University of South Bohemia
Faculty of Science
České Budějovice, Czech Republic

Johannes Kepler University
Faculty of Engineering & Natural Sciences
Linz, Austria

Comparison of bioinformatics pipelines for eDNA metabarcoding data analysis of fish populations in Czech reservoirs

Bachelor Thesis

Rômulo Acácio dos Santos

Supervisor: RNDr. Petr Blabolil, Ph.D.

České Budějovice
2021

Bibliographical Detail

dos Santos R., 2021: Comparison of bioinformatic pipelines for eDNA metabarcoding data analysis of fish populations in Czech reservoirs. Bc. Thesis, in English. – 201 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic and Faculty of Engineering and Natural Sciences, Johannes Kepler University, Linz, Austria.

Annotation

The aim of the study is a comparison of five distinct pipelines for environmental DNA (eDNA) metabarcoding using data collected in three reservoirs (Klíčava, Římov, and Žlutice) in the summer and autumn seasons. The results are analysed by comparing the number of reads assigned, number of species detected, and ecological indices (alpha and beta diversity). Finally, statistical analysis is applied to corroborate the results using analysis of variance (ANOVA), post-hoc Tukey, and permutational multivariate analysis of variance (PERMANOVA).

Declaration

I hereby declare that I have worked on my bachelor‘s thesis independently and used only the sources listed in the bibliography. I hereby declare that, in accordance with Article 47b of Act No. 111/1998 in the valid wording, I agree with the publication of my bachelor thesis, in full form to be kept in the Faculty of Science archive, in electronic form in publicly accessible part of the STAG database operated by the University of South Bohemia in České Budějovice accessible through its web pages.

Further, I agree to the electronic publication of the comments of my supervisor and thesis opponents and the record of the proceedings and results of the thesis defense in accordance with aforementioned Act No. 111/1998. I also agree to the comparison of the text of my thesis with the Theses.cz thesis database operated by the National Registry of University Theses and a plagiarism detection system.

In České Budějovice, April 13, 2021,

Rômulo Kacio dos Santos

Signature

Abstract

Environmental DNA (eDNA) metabarcoding has been increasing in popularity as a method for biodiversity monitoring in ecology. The number of new tools and pipelines developed every year is increasing in parallel. However, a proper validation of the results must be conducted to validate metabarcoding is a reliable method and avoid incorrect ecological assessment caused by dissimilar results. The aim of this study is to make a comparison of eDNA metabarcoding pipelines.

The study was conducted in three reservoirs in the Czech Republic (Klíčava, Římov, and Žlutice) using data collected in the summer and autumn seasons. Samples plus negative and positive controls were processed in the laboratory and sequenced. The 12S rRNA gene was chosen as the genetic marker used as the short barcode DNA section. Five distinct pipelines were selected to be compared. Anacapa, Barque, metaBEAT, MiFish, and SEQme comprises the main tools used in eDNA metabarcoding. A reference database file was created by updating the one developed by colleagues at the University of Hull with either sequences downloaded from public databases or *de novo* sequences included to overcome the lack of or low-quality sequences. Sequences representing fish species with possibility to be present in the reservoirs were included.

Alpha (richness and Shannon index) and beta (Jaccard index) diversities ecological indices together with number of species detected and number of reads assigned were used in the comparison. ANOVA, post-hoc Tukey, and PERMANOVA were applied to analyze the similarity of the results. Pipelines had high similarity with consistent statistical results. Species assigned were also compared to species detected by conventional methods. Pipelines demonstrated to have a higher sensitive in species detection than conventional methods. Finally, the study indicated a similarity in the results, thus corroborating eDNA metabarcoding as a reliable method for biomonitoring, which can help in taking decisions on wildlife management and help to save our planet for current and future generations.

Acknowledgements

I would like to express my deepest gratitude to my family, Ana, and all my friends for their patience, support, and love. I also would like to thank and express my gratitude to my supervisor Petr Blabolil for guiding and supporting me, and for being an inpiration as a researcher. A warmly thank to my professor Marta Vohnoutová, whose has supported, advised, and encouraged me throughout the period of my studies. Many thanks to my new friends resulted from the friendship built during the Bioinformatics course, and to my brazilian "family" in the Czech Republic. Finally, I would like to thank the colleagues from Fish Ecology Unit at the Institute of Hydrobiology, Biology Centre CAS (www.fishecu.cz) and Evo Hull group at University of Hull (www.evohull.org) for help with samples collection and laboratory processing. The work was supported by the projects QK1920011 "Methodology of predatory fish quantification in drinking-water reservoirs to optimize the management of aquatic ecosystems" and MSM200961901 "The true picture of eDNA", and by the CAS within the program of the Strategy AV 21 (VP21).

Contents

Introduction

Environmental DNA (eDNA) has been recently increasing in popularity in the field of molecular ecology for monitoring the ecosystem biodiversity [Seymour, 2019]. Environmental DNA consists of mutiple genomic DNA from different organisms found in a sample (water, soil, *etc.*) collected from the environment [Taberlet, Coissac, Hajibabaei, & Rieseberg, 2012]. The barcoding method was proposed by Hebert, Cywinska, Ball, and deWaard [2003] as a technique of identification of single species by a genetic marker (barcode). The meta prefix was added to the barcoding method to indicate a barcoding of multi-taxa identification. Metabarcoding is a non-invasive technique derived from barcoding where all taxas in a sample are attempted to be identified without capturing the organisms [Thomsen & Willerslev, 2015]. However, multiple species identification adds an additional challenge as primers tend to amplify in favor of some taxas, being more efficiently for some species over the others [Deagle, Jarman, Coissac, Pompanon, & Taberlet, 2014]. In addition, closely related species may not have a high variability in the DNA sequences, which could induce species to be identified in a lower level as genus or family [Taberlet, Bonin, Zinger, & Coissac, 2018b]. The advances in technology have improved the sequencing capabilities and reduced the cost per sample, providing a cost-effective approach for biodiversity research [Reuter, Spacek, & Snyder, 2015; Taberlet, Coissac, Pompanon, Brochmann, & Willerslev, 2012]. In addition, the reduced amount of people to perform an eDNA metabarcoding study provides a labour-effective alternative to conventional methods [Bohmann et al., 2014].

Fishes detection using eDNA metabarcoding was shown to outperform conventional methods [Hänfling et al., 2016]. Each species of fish have distinct preferences for the habitat they live, from the darker, colder, poorer in oxygen content, and higher in pressure deep water zone to the lighter, warmer, richer in oxygen content, and lower in pressure in the region close to surface [Kottelat & Freyhof, 2007]. Four different factors influence the characteristics and presence of eDNA in freshwater ecosystems. First, fishes release a large amount of eDNA in the habitat. Second, the eDNA can survive in freshwater for only a short period of time (few weeks). Finally, transport and diffusion distances are influenced by the concentration of eDNA and speed of the current [Taberlet, Bonin, Zinger, & Coissac, 2018d]. The four aspects make eDNA metabarcoding an appropriate technique to assess the presence or the absence of species in freshwater. However, environmental conditions (temperature, UV-B

radiation, microbial activity, *etc.*) can affect the eDNA degradation [Kasai, Takada, Yamazaki, Masuda, & Yamanaka, 2020]. Among the conditions, temperature has the strongest influence on degradation rate [Strickler, Fremier, & Goldberg, 2015]. Considering all the aspects, all major habitats with sampling in the summer and autumn seasons (as in the winter the ice cover would block the sampling) must be considered to maximize the detection of the species and to analyse their influence in the final result of the study.

The data collected in the field is analysed in silico by using a sequence of specialized modules. A common pipeline workflow is composed of trimming, merging, filtering, dereplication, clustering/inferring, and taxonomic assignment [Taberlet, Bonin, Zinger, & Coissac, 2018c]. In the trimming step, low quality bases, adapters, and/or primers are removed from the sequences [Martin, 2011; Bolger, Lohse, & Usadel, 2014]. In the merging step, paired reads (forward and reverse), generated when sequencing both end in a Illumina sequencer machine to improve the sequence quality and detect sequencing errors, are merged into a single sequence [Magoč & Salzberg, 2011; Aronesty, 2013]. In the filtering step, reads with length below the threshold, chimeric sequences, and/or sequences with the overall quality below the threshold are removed [Rognes, Flouri, Nichols, Quince, & Mahé, 2016; Bolger et al., 2014]. In the dereplication step, identical sequences are combined into a single one to avoid redundant comparison and improve the speed, the number of reads in the group is annotated [Rognes et al., 2016]. In the clustering/inferring step, related sequences are grouped into operational taxonomic unit (OTU) or amplicon sequence variant (ASV) are inferred [Xiong & Zhan, 2018]. Finally, sequences are assigned to a taxonomic level [Holovachov, Haenel, Bourlat, & Jondelius, 2017]. However, different bioinformatics pipelines are being used for the high-throughput sequence data analysis in different studies and this makes the studies hardly comparable.

A comparison of distinct approaches in each step of the bioinformatic data processing must be performed to evaluate the importance of each stage of the pipeline in the final result. In addition, divergent parameters of the same program would also contribute for different results as demonstrated by Pauvert et al. [2019]. They also found that the more flexible the pipeline is, the more taxa will be detected, but there will be more false positives (species detected in the pipeline but not present in the study site). On the other hand, more specialized pipeline will select more trustworthy taxa, but there will be more false negatives (species not detected in the pipeline but present in the study site). A comparison of bioinformatic pipelines and the role of each step in the analysis of the eDNA metabarcoding of fish populations in Czech reservoirs has not been not fully conducted. Five distinct pipelines covering the main tools used in eDNA metabarcoding were chosen to be compared.

Anacapa pipeline was developed at the University of California, USA, to process multilocus metabarcode sequence data [Curd et al., 2019]. The Anacapa workflow starts by trimming adapters, primers at the 3' end, and any subsequent bases using Cutadapt [Martin, 2011]. FASTX-Toolkit is used to trim sequences based on quality score [Gordon, Hannon, et al., 2010].

Cutadapt is again used to trim any primer and subsequent bases [Martin, 2011]. DADA2 is used to filter, trim, dereplicate, merge, remove chimeras, and infer amplicon sequence variant [Callahan et al., 2016]. Bowtie 2 is used to perform a global alignment to the sequences [Langmead & Salzberg, 2012]. Finally, Bayesian Lowest Common Ancestor (BLCA) method with MUSCLE alignment is used to assign taxonomy and generate the bootstrap confidence scores [Gao, Lin, Revanna, & Dong, 2017; Robert C. Edgar, 2004].

Barque pipeline was developed at the Laval University, Canada [Normandeau, n.d.]. The pipeline annotates reads instead of Operational Taxonomic Unit (OTU) as the main difference. The Barque workflow starts by using Trimmomatic to trim and filter raw reads [Bolger et al., 2014]. Paired-end (PE) sequencing forward and reverse files are merged using FLASH [Magoč & Salzberg, 2011]. A custom Python script is used to remove primers from 5' and 3' end of the sequences. VSEARCH is used to dereplicate and remove chimeric sequences [Rognes et al., 2016]. A custom Python script is used to merge identical reads and sum up for the unique sequences the number of sequences found in each cluster created in the dereplication step. Finally, VSEARCH is used to execute a global pairwise alignment between the reads and the reference database for taxonomic assignment.

MetaBEAT pipeline is a metabarcoding and environmental DNA analysis tool developed at the University of Hull, United Kingdom [Hahn & Lunt, n.d.]. The metaBEAT workflow starts by trimming low quality bases from the sequences using Trimmomatic [Bolger et al., 2014]. FLASH is used to merge forward and reverse reads [Magoč & Salzberg, 2011]. Identical sequences are groupped (dereplicated), chimeras are removed, and a clustering of OTU is performed using VSEARCH [Rognes et al., 2016]. Finally, BLAST is used to align the sequences to the reference database for taxonomic assignment [Camacho et al., 2009].

MiFish pipeline was developed at the University of Tokyo, Japan, to be used with the sequences amplified by the set of primers created by the same research group [Miya et al., 2015; Sato, Miya, Fukunaga, Sado, & Iwasaki, 2018]. MiFish workflow starts by checking the sequence quality using FastQC [Andrews et al., 2010]. Read sequences are trimmed using DynamicTrim from SolexaQA package [Cox, Peterson, & Biggs, 2010]. Sequences are merged using FLASH [Magoč & Salzberg, 2011]. A custom Perl script is used to remove sequences with ambiguous bases represented by the letter N. Sequences are filtered based on length using a custom Perl script. Primers sequences are removed by using TagCleaner [Schmieder, Lim, Rohwer, & Edwards, 2010]. USEARCH is used to dereplicate (group identical sequences), filter, and align the sequences [Robert C. Edgar, 2010]. Finally, sequences are aligned to the reference database using BLAST for taxonomic assignment [Camacho et al., 2009].

SEQme pipeline was created by the SEQme private company and presented during the metabarcoding and metagenomics workshop [SEQme, 2018]. SEQme workflow starts by merging forward and reverse reads into single sequences using fastq-join [Aronesty, 2013]. FASTX-Toolkit command fastq_quality_filter is used to filter sequences based on quality

[Gordon, Hannon, et al., 2010]. Sequences are filtered based on length using command lines read_fasta, grab, and write_fasta from Biopieces bioinformatic framework [Hansen, Oey, Fernandez-Valverde, Jung, & Mattick, 2008]. USEARCH is used to dereplicate, remove chimeras, and cluster OTU [Robert C. Edgar, 2010]. Finally, the taxonomic classification is done by a Bayesian classifier from the Ribosomal Database Project (RDP) [Wang, Garrity, Tiedje, & Cole, 2007].

Work Aims

The aim of this study was to compare the results of the taxonomic assignment of the eDNA metabarcoding of five different pipelines. Sampling was conducted by the Institute of Hydrobiology, Biology Centre CAS, in the summer and autumn seasons to evaluate the influence of the water temperature in the final results. All major habitats were considered for sampling as different fish species have distinct nature. The number of sequence reads for each step in the workflow execution was counted to check any deviation from the average. The total number of reads assigned to species for each pipeline was measured to evaluate the variation among the pipelines. The total number of species detected in all pipelines, also known as gamma diversity, was calculated to be compared with the list of species detected by conventional methods and check the reliability of the results. The number of sequence reads assigned to each species was measured to evaluate the relative abundance. The alpha diversity was computed to compare detection among the pipelines. It describes the number of species (i.e. richness) detected in a determined group such as pipeline. The inequality of the number of reads assigned between species was assessed by applying the Shannon index in the alpha diversity. It was calculated to check if the relation number of species detected and number of reads assigned between species was similar between the groups. In addition to the alpha diversity, the beta diversity was also considered as the species composition of two groups can be completely different even with identical alpha diversities. It was computed to assure that the species composition of groups were similar. The Jaccard index was used to calculate beta diversity, it considers only the presence of species and ignore the number of reads assigned. Positive and negative controls detection were checked to ensure data reliability. All these validations are essential to make certain that the pipelines provide similar outcomes. These data are crucial for ecological studies. For this reason, an incorrect ecological assessment by scientists, water and fisheries managers, or nature protection agencies on creating effective protection of nature and wildlife could be catastrophic. This study aim to improve conditions to keep natural heritage for our descendants and somehow help to save the world.

Material and Methods

3.1 In the field and laboratory

3.1.1 Study site

The study was conducted in three reservoirs in the Czech Republic built as drinking water storages and restricted to public access. The three reservoirs (Klíčava, Římov, and Žlutice) possess different characteristics, but similar canyon-shape morphology with one main inflow and one side bay (Table 3.1).

Parameter	Klíčava	Římov	Žlutice
Trophic state	oligotrophic	eutrophic	eutrophic
Dam geographical coordinates	50°3'52.166"N 13°56'2.356"E	48°51'0.257"N 14°29'27.409"E	50°5'12.113"N 13°7'36.681"E
Elevation above sea level [m]	294	470	509
Volume [mil.m^3]	8.3	34	14
Flooded area [km^2]	0.62	2.1	1.6
Maximum depth [m]	34	42	23
Average depth [m]	13	16	9

Table 3.1: Trophic state, geographical, and morphological parameters of studied reservoirs.

Fish communities in the reservoirs have been monitored by convetional methods repeatedly (benthic and pelagic gillnets, continuous electrofishing, trawling and other methods), keeping a stability in the species detected and being dominated by cyprinid species (Table 3.2).

Species	Klíčava	Římov	Žlutice
Lampetra planeri		X	
Acipenser baerii		X	
Anguilla anguilla	X	X	X
Rutilus rutilus	X	X	X
Chondrostoma nasus		X	
Squalius cephalus	X	X	X
Alburnus alburnus	X	X	X
Blicca bjoerkna		X	
Abramis brama	X	X	X
Leuciscus idus		X	X
Leuciscus leuciscus	X	X	X
Leuciscus aspius	X	X	X
Scardinius erythrophthalmus	X	X	X
Pseudorasbora parva	X	X	X
Gobio gobio	X	X	
Tinca tinca	X	X	X
Hypophthalmichthys molitrix			X
Hypophthalmichthys nobilis	X		
Ctenopharyngodon idella		X	X
Cyprinus carpio	X	X	X
Carassius auratus	X	X	X
Barbatula barbatula		X	
Esox Lucius	X	X	X
Sander lucioperca	X	X	X
Perca fluviatilis	X	X	X
Gymnocephalus cernua	X	X	X
Lepomis gibbosus		X	
Oncorhynchus mykiss		X	
Salmo trutta			X
Coregonus maraena		X	
Silurus glanis	X	X	X
Lota lota		X	X

Table 3.2: List of species detected by traditional methods by FishEcU members (www.fishecu.cz) in the studied reservoirs in the last 3 years (2018, 2019, and 2020).

3.1.2 eDNA sampling and laboratory processing

The water sampling was conducted in the summer (August) and autumn (November/December) seasons of 2018. The total number of samples collected in each reservoir in the summer and autumn seasons was 29 and 30 in Klíčava; 38 and 35 in Římov; and 28 and 29 in Žlutice, respectively. The distribution of sampling was design to cover all major habitats (littoral, surface, deep water and inflows). The water was sampled in five locallities in Klíčava and Žlutice, and eight localities in Římov, with intervals of approximately 1 km between the locallities to cover the heterogeneity of species composition. One more locality was sampled for each reservoir in a side bay. Water from both banks in the littoral region, from the surface of open water (pelagic), and from deep layers (5, 10, and 20 meters dependent on the depth at the locality) were collected. In Klíčava and Žlutice the samples were collected in all localities, whereas in Římov deep water samples were only considered at the localities 1, 4, 6, and the side basin. During the autumn campaign, water was not sampled at the locality 8 in Římov due to ice cover. In addition, in Klíčava and Žlutice bays deep water samples were not collected in the summer season.

Two liters of water were sampled at each locality and pre-filtered in the field to prevent clogging from excessive seston. The water sampled was stored inside sterile labeled bottles and kept cold inside a box with ice until being processed. Within 24 hours after sampling, one litre of each sample was filtered through open filters in the laboratory [for details see Blabolil et al., 2020]. Two field blanks were included in each sampling event and processed together with the reservoir water samples. The Mu-DNA water protocol [Sellers, Muri, Gómez, & Hänfling, 2018] was used to extract the DNA. PCR amplicons were produced using the primers (forward ACTGGGATTAGATACCCC and reverse TAGAACAGGCTCCTCTAG) designed by Riaz et al. [2011]. Negative (molecular grade water) and positive (*Maylandia zebra* DNA 0.05 ng μl^{-1}) controls were included during PCR to detect possible contamination and inhibition. Finally, the sequencing library was generated from PCR amplicons and run on an Illumina MiSeq sequencer. For methodological detail see Blabolil et al. [2020].

3.2 At the keyboard

All the analyses, from the reference database creation to the taxonomic classification and statistical tests, were conducted in a Linux Ubuntu Mate Server computer with Intel Xeon CPU E5-2620 v2 2.10 GHz x 12 and 24 GB RAM. A common pipeline workflow is described on Figure 3.1.

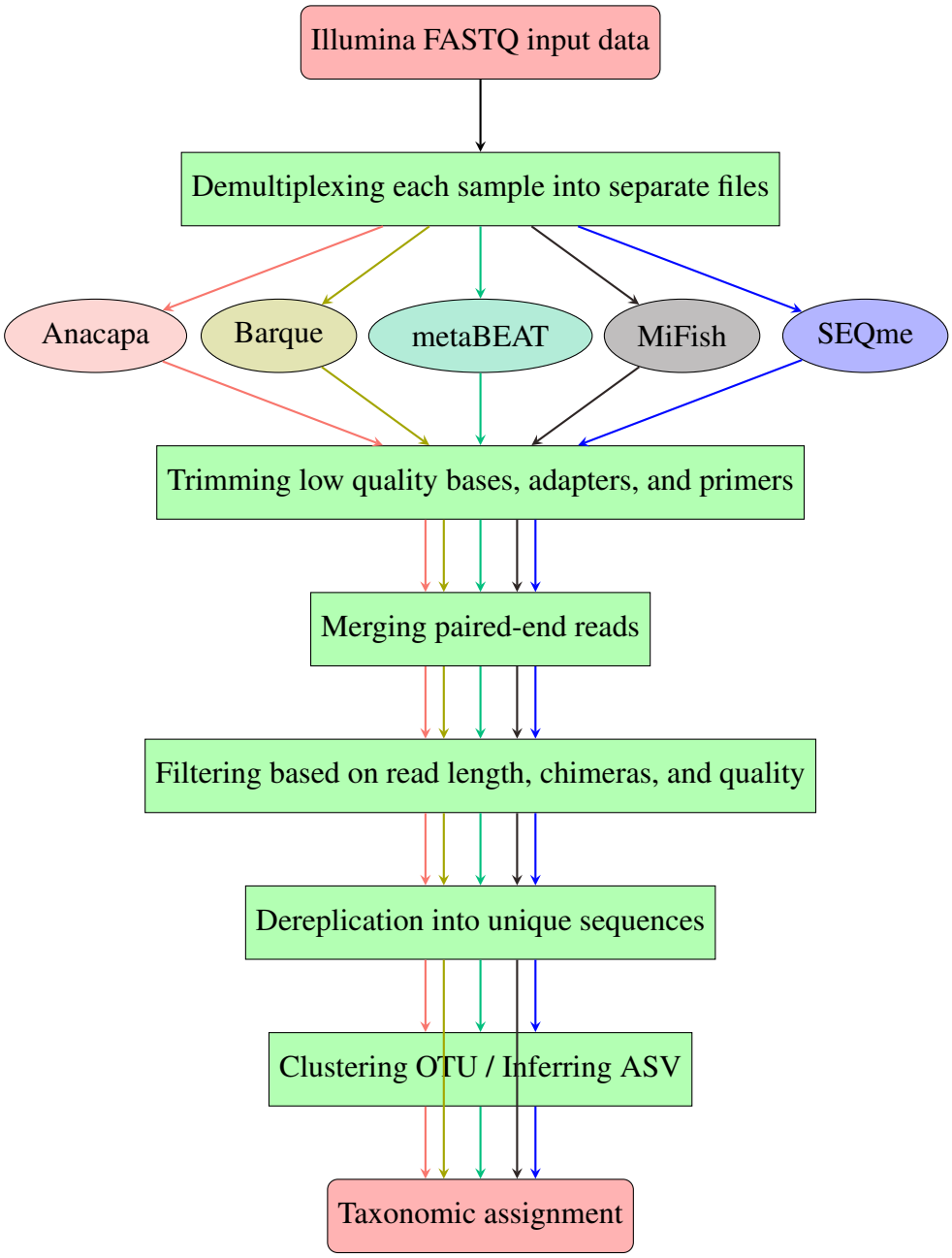

Figure 3.1: Metabarcoding workflow for the pipelines.

3.2.1 Reference database

A custom reference database based on the molecular marker 12S rRNA gene was created by adding *de novo* sequences and sequences from GenBank [Clark, Karsch-Mizrachi, Lipman, Ostell, & Sayers, 2015] to the reference database developed at the University of Hull [Hänfling et al., 2016] (Table 3.3). The reference database was created to represent all fish species detected by conventional methods in the reservoirs where the samples were collected and additional non-detected species that could possibly be present. *De novo* sequences were included to overcome the lack of sequences for the 12S gene in public databases or species represented by low-quality sequences [Weigand et al., 2019]. The *de novo* sequences were submitted to

the genbank NCBI database and can be found with the accession numbers from MW652796 to MW652804.

Species	Accession number
Lampetra planeri	MW652804
Acipenser-sp.	AY442351.1, AY544140.1
Lepomis gibbosus	MF621724.1
Cottus poecilopus	AB188187.1, AB188185.1, EU332750.1
Chondrostoma nasus	MW652798
Cobitis elongatoides	KF926686.1
Cobitis taenia	MW652799, MW652800
Sabanejewia balcanica	AY887776.1
Cyprinus carpio	MW652797
Hypophthalmichthys nobilis	MF180233.1
Aspius+Scardinius	AB239597.1
Phoxinus phoxinus	MW652802
Romanogobio albipinnatus	MW652796
Rutilus rutilus	AP010775.1
Proterorhinus marmoratus	MT484059.1
Ameiurus nebulosus	MF621733.1
Barbatula barbatula	MW652803, MW652801
Gymnocephalus baloni	AY372795.1
Oncorhynchus mykiss	MF621750.1
Salvelinus fontinalis	AF154850.1
Hucho hucho	KM588351.1
Umbra krameri	AY430269.1

Table 3.3: Sequences added to the reference database developed at the University of Hull.

The reference database was curated by keeping only sequences from the 12S gene locus, removing redundant sequences, filtering sequences by length, and correcting taxonomically mislabelled sequences. The curation process was done based on the descriptions from the Curated reference databases github repository of the Evolutionary and Environmental Genomics Group, University of Hull, United kingdom (https://github.com/HullUni-bioinformatics/Curated_reference_databases). VSEARCH version 2.14.2 [Rognes et al., 2016] was used to cluster and remove redundant sequences. A variant of the UCLUST algorithm that maximize the speed [Robert C. Edgar, 2010] was applied. A threshold of 100 % for the identity and the query cover was applied for the sequences clustering. The reverse complement of the sequence was also considered when clustering the sequences. Finally, the result was saved in a tab-separated file with information about the clusters and used to discard redundant sequences from the reference

file (Source Codes A.1 and 3.1).

```
python Clustering.py reference_database.gb --num_threads 10
```

Source Code 3.1: Execution of the Source Code A.1.

A custom Python script (Python version 3.7.3 [Van Rossum & Drake, 2009]) was used to discard sequences with length smaller than 200 bases (Source Codes A.2 and 3.2). MAFFT version 7.310 [Katoh & Standley, 2013] was used to align the sequences using Smith-Waterman algorithm local sequence alignment with 1,000 iterations to refine the alignment. Reverse complement of divergent sequences was created as an attempt to improve the alignment quality (Source Codes A.3 and 3.3). TrimAl version 1.4.rev15 [Capella-Gutiérrez, Silla-Martínez, & Gabaldón, 2009] was used to remove large gaps in the sequences of the multiple alignment for phylogenetic analyses (Source Codes A.4 and 3.4).

```
python Filter_by_Length.py reference_database.gb --threshold 200
```

Source Code 3.2: Execution of the Source Code A.2.

```
python Alignment.py reference_database.gb --program mafft --rank superkingdom
```

Source Code 3.3: Execution of the Source Code A.3.

```
python Trim_Alignment.py reference_database.aln
```

Source Code 3.4: Execution of the Source Code A.4.

SATIVA (https://github.com/amkozlov/sativa) was used to identify taxonomically mislabelled sequences [Kozlov, Zhang, Yilmaz, Glöckner, & Stamatakis, 2016]. The reference database genbank file was used to create a taxonomy file (tab-separated text file) with accession numbers in the first column and the taxonomic levels separated by semicolon in the second column (from superkingdom to species). Previously aligned sequences trimmed by trimAl together with the taxonomy file were used as input to sativa (Source Codes A.5 and 3.5). Mislabelled sequences identified by SATIVA were removed from the reference database file using a Python (Python version 3.7.3 [Van Rossum & Drake, 2009]) custom script (Source Codes A.6 and 3.6).

```
python Sativa.py reference_database.gb reference_database.phy
```

Source Code 3.5: Execution of the Source Code A.5.

```
python Remove_Mislabelled.py reference_database.gb sativa.mis
```

Source Code 3.6: Execution of the Source Code A.6.

After correcting taxonomically mislabelled entries, MAFFT version 7.310 [Katoh & Standley, 2013] and TrimAl version 1.4.rev15 [Capella-Gutiérrez et al., 2009] were again used to align the database file and trim the sequences alignment (Source Codes A.3, 3.3, A.4 and 3.4). RAxML (Randomized Axelerated Maximum Likelihood) version 8.2.12 [Stamatakis, 2014] was used to infer a maximum likelihood phylogenetic tree from the aligned sequences. The multi-threaded version of RAxML (*raxmlHPC-PTHREADS-SSE3*) was used in the tree creation. In addition, the option for a rapid Bootstrap and a search for the best-scoring was used. The tree was inferred using GTRGAMMA substitution model. The parsimony inference seed for the generation of the starting tree was set to 765 and the seed to start the heuristic search was set to 498 (seeds are used to obtain the same results every time the tree is created using the same seeds). A total of 100 execution on different starting trees were used to build the phylogenetic tree (Source Codes A.7 and 3.7). The new tree was used to check if the sequences in the reference file reflect accurate relationships among the species, where closely related fishes were grouped sharing a common ancestor [Pavlopoulos, Soldatos, Barbosa-Silva, & Schneider, 2010]. Finally, MAFFT version 7.310 [Katoh & Standley, 2013] was used to align the primer sequences (forward ACTGGGATTAGATACCCC and reverse TAGAACAGGCTCCTCTAG) to the sequences from the reference file (Source Codes A.3 and 3.8).

```
python Build_Tree.py reference_database.aln
```

Source Code 3.7: Execution of the Source Code A.7.

```
python Alignment.py reference_database.gb --program mafft --rank species --primers
    ↪ primers.fasta
```

Source Code 3.8: Execution of the Source Code A.3 with the addition of primers in the alignment.

A manual curation was applied to the reference database based on the alignment. Sequences without any bases inside the primers amplification region (in the region between the forward and reverse primers in the alignment) were removed from the database. Sequences with identical subsequences for the region in between the primers were either discarded if

from the same species, or the name of the species were joined to avoid multiple different species assigning a read sequence. In both cases only one sequence was kept to represent the group (Table 3.4). Two Acipenser species with no identical subsequence between the primers, *Acipenser sturio* and *Acipenser ruthenus*, were also represented as *Acipenser-sp.* as *Acipenser* species were reassigned to genus level.

Species with identical subsequences	Species representing the group
Acipenser gueldenstaedtii, *Acipenser nudiventris*, *Acipenser stellatus*, and *Huso huso*	*Acipenser-sp.*
Coregonus lavaretus and *Coregonus maraena*	*Coregonus-sp.*
Leuciscus idus and *Leuciscus leuciscus*	*L.idus+leuciscus*
Blicca bjoerkna and *Vimba vimba*	*Blicca+Vimba*
Leuciscus aspius, *Pelecus cultratus*, and *Scardinius erythrophthalmus*	*Aspius+Scardinius*
Perca fluviatilis and *Sander Lucioperca*	*Sander+Perca*

Table 3.4: List of species with identical subsequences in the region between the primers in the alignment and the species that represents the group in the reference database after joining the identical sequences to a unique entry in the reference file.

For methodological detail on how the curation process is done see the descriptions on the Curated reference databases github repository from the University of Hull (https://github.com /HullUni-bioinformatics/Curated_reference_databases). For the precise reproducibility of the taxonomic classification analysis, the curated reference database and the python codes were uploaded to a Github repository and are available by accessing https://github.com/RomuloAS/ eDNA_metabarcoding_pipelines_comparison.

3.2.2 Read sequences curation and taxonomic classification

Illumina MiSeq raw sequence reads were demultiplexed using a Python script developed by the Evolutionary and Environmental Genomics Group at the University of Hull (http: //www.evohull.org/). In addition, a custom Python script was created to automatize the process for all files (Source Codes A.8, A.9 and 3.9). Demultiplexed files were submitted to NCBI genbank database and can be downloaded using NCBI Sequence Read Archive accession number PRJNA611963.

```
1 python Demultiplex.py FASTQ_files_folder/ Demultiplex_tables_folder/
```

Source Code 3.9: Execution of the Source Code A.9.

Before the pipelines being executed, adapters in the 3' end of the read fragment were removed using Cutadapt version 1.18 [Martin, 2011] (Source Codes A.10 and 3.10). The tables

with information about the adapters to demultiplex the FASTQ files and remove the adapter in the 3' end of the read sequence can be downloaded by accessing https://github.com/RomuloAS/ Github repository. In addition, FASTQ raw reads for each PCR product can be downloaded under the accession number PRJNA611963.

```
python Remove_Adapter.py FASTQ_files_folder/ Demultiplex_tables_folder/
```

Source Code 3.10: Execution of the Source Code A.10.

Finally, the curated reference database was converted to FASTA format considering each pipeline requirements for the description line (Source Code 3.11). An additional table with taxonomic information was created for Anacapa and SEQme pipelines.

```
python genbank2Fasta.py reference_database.gb --pipeline anacapa --rank superkingdom
```

Source Code 3.11: Execution of the Source Code A.11.

3.2.2.1 Anacapa

The Anacapa eDNA toolkit (https://github.com/limey-bean/Anacapa) [Curd et al., 2019] from the Univeristy of California was used to process the FASTQ files and to assign taxonomy to sequence data (Source Codes 3.12 and 3.13). A full description about the installation process of the pipeline and dependencies can be found on the Github repository. Forward, reverse, and minimum length for the overlapping region files used in the execution of the Anacapa read sequences curation script were created. Forward and reverse primer files (forward_primers.txt and reverse_primers.txt) were filled out with the gene locus 12S in the description line and the respective primer sequences (forward ACTGGGATTAGATACCCC and reverse TAGAACAG GCTCCTCTAG) in the sequence data line. The minimum length for the overlapping region file (metabarcode_loci_min_merge_length.txt) was set to 126 as the minimum overlap required (LENGTH_12S="126").

```
bash anacapa_QC_dada2.sh -i path_to_input_data_folder -o path_to_output_data_folder -
    ↪ d path_to_Anacapa_db -a nextera -t MiSeq -l -f forward_primers.txt -r
    ↪ reverse_primers.txt -q 20 -m 90 -x 0 -y 0 -e
    ↪ minimum_length_for_the_overlapping_region.txt
```

Source Code 3.12: Execution of the Anacapa read sequences curation script. For the complete script code see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/anacapa_QC_dada2.sh.

```
bash anacapa_classifier.sh -o path_to_output_data_folder -d path_to_Anacapa_db -l -b
    ↪ 1 -p 0.85 -n 1000
```

Source Code 3.13: Execution of the Anacapa taxonomic assignment script. For the complete script code see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/anacapa_classifier.sh.

Cutadapt version 1.18 [Martin, 2011] was used to remove the nextera adapter and any previous bases from the 5' end of the sequences for R1 and R2 reads from paired-end (PE) sequencing. Primer, nextera adapter, and any subsequent bases from to the 3' end of the sequence for R1 and R2 reads were also removed. A maximum error rate of 30 % (the number of mismatches, insertions, and deletions in a match divided by the length of the adapter matching region) was allowed (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/anacapa_QC_dada2.sh#L256-L258).

FASTX-Toolkit version 0.0.14 [Gordon, Hannon, et al., 2010] command line fastq_quality_trimmer was used to trim sequences based on quality using phred 33 quality score (the lowest score starts at the position 33 of the ASCII table). Starting from the 3' end of the sequence, bases with quality below 20 were cut off until a base with quality equal or above the threshold was found (if the stop condition was found in the first iteration, the sequence was returned intact). After trimming the end of the sequence, sequences smaller than 90 bases were discarded (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/anacapa_QC_dada2.sh#L261-L266).

Primers were removed from to the 5' end of the reads for forward (R1) and reverse (R2) sequences using Cutadapt version 1.18 [Martin, 2011]. A maximum error (mismatches, insertions, and deletions) rate of 30 % calculated by dividing the number of errors by the number of bases in the matching region was considered (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/anacapa_QC_dada2.sh#L270-L278).

A custom Python script was used to check if the sequences from forward (R1) and reverse (R2) files matched (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/check_paired.py). Four output files were generated, two representing sequences that matched from forward (R1) and reverse (R2) and two representing solitary sequences from both forward (R1) and reverse (R2) files (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/anacapa_QC_dada2.sh#L327-L328).

DADA2 version 1.6.0 [Callahan et al., 2016] was used to filter, trim, dereplicate, and merge the sequences. In addition, a sequence table analogous to an OTU table was constructed and chimera sequences were removed. DADA2 was executed inside R language environment version 3.4.3 [R Core Team, 2020] (for the R code see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/dada2_unified_script.R). During the filtration and trimming step, sequences with length smaller than 10 bases were removed. Sequences with

any incidence of a N base (representing an ambiguity of any base according to the IUPAC nomemclature code [Cornish-Bowden, 1985]) were also discarded. Additionally, reads with an expected error allowed in the read higher than 2 were discarded. Sequences were also removed if matched against the bacteriophage phix genome. Finally, sequence identifier from forward (R1) and reverse (R2) sequences were compared and only when matching up the identifiers the sequences were saved to the output data. The dereplicated data and error rates (errors introduced by PCR amplification and sequencing) learned from the result after filtering and trimming the sequences were used to apply the divisive amplicon denoising algorithm (dada) and denoise the reads, resulting in inferred sequence variants in each sample. After applying the inference algorithm, forward (R1) and reverse (R2) sequences were merged considering a minimum length of 20 bases and a maximum of 2 mismatches in the overlap region. An amplicon sequence variant (ASV) table was generated from the merged sequences (table similar to an OTU table), where the rows represent the samples and columns represent the inferred unique sample sequence. Finally, chimeras sequences were removed using consensus sequence created across samples. DADA2 steps were applied for paired files and both forward (R1) and reverse (R2) files representing unmerged sequences (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/anacapa_QC_dada2.sh#L353-L362).

The previously converted reference database in FASTA format and a taxonomic table generated in the conversion process (Source Code 3.11), where the first column has the name of the read and the second column has taxonomic rank from superkingdom to species separated by semicolon, were used to build the Bowtie 2 reference database using Bowtie 2 version 2.3.5.1 [Langmead & Salzberg, 2012] command line bowtie2-build (Source Code 3.14). For a detailed description about Bowtie 2 reference database creation see https://github.com/limey-bean/CRUX_Creating-Reference-libraries-Using-eXisting-tools/blob/master/Manual_addition_of_reads_to_CRUX.txt.

```
1 bowtie2-build -f 12S_.fasta ../12S_bowtie2_database/12S_bowtie2_index
```

Source Code 3.14: Bowtie database creation using the reference database in FASTA format and the taxonomic table with the first column having the name of the read and the second column having taxonomic rank from superkingdom to species separated by semicolon.

Bowtie 2 was used to perform a global alignment of the merged sequences in FASTA format to the reference database. The alignment was executed in very sensitive mode, where speed is given in exchange for sensivity and accuracy, being slower but more sensitive and more accurate in the relation speed, sensitivity, and accuracy. Only alignments where the entire read sequence is aligned (from one end to the other) without any bases being ignored in the alignment were considered. For each read sequence a total of 100 distinct alignments were considered. The result of the global alignment was saved in a SAM (Sequence Alignment/Map)

format, which consists of a header section and a alignment section for each aligned sequence [Li et al., 2009]. The header lines metadata, reference sequence dictionary, and information for unaligned reads were suppressed. In addition, sequences that were rejected in the alignment to the reference database were saved to a FASTA file. Finally, the rejected sequences were used to perform a local alignment using the same parameters used to run the global alignment (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/run_bowtie2_blca.sh#L105-L110).

The global and local alignments were repeated for unmerged sequences, considering together the forward and the reverse FASTA files. The sequences were aligned to the reference database and saved in a SAM format. The header lines metadata, reference sequence dictionary, and information for unaligned reads were ignored. Alignments where only a sequence from the forward file or only a sequence from the reverse file was aligned alone to a particular sequence in the reference database were ignored. Paired-end alignment when both a sequence from the forward file and a sequence from the reverse file were aligned to a particular sequence in the dabatase were considered when met the criteria for a concordant alignment for the pair of sequences. An alignment was considered valid by the concordant alignment conditions when either a sequence from the forward file was aligned in the 5' end and the reverse complement of the sequence from the reverse file was aligned in the 3' or a sequence from the reverse file was aligned in the 5' end and the reverse complement of the sequence from the forward file was aligned in the 3' end. Additionally, the alignment was also considered valid when the reverse complement of the sequence from the forward file was aligned in the 5' end and the sequence from the reverse file was aligned in the 3' end. The very sensitive mode was used in the alignment to maximize the sensitivity and accuracy in exchange of speed. In addition, 100 distinct alignments for each sequence where the whole read sequence was aligned to the reference database were considered. Sequences ignored in the previous step were saved to a FASTA file. Finally, the sequences rejected in the preceding global alignment were aligned using the same parameters but applying a local alignment algorithm (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/run_bowtie2_blca.sh#L129-L132).

Custom Python scripts were used to join the amplicon sequence variant (ASV) tables resulted after DADA2 execution (Merged, unmerged, forward, and reverse files) in a unique file. The first column represent the identifier of the ASV, the column names (from the second to the last one) represent each sample identifier, and each cell of the table represent the number of sequences in the group of the amplicon sequence variant for the intersection ASV and sample identifiers (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/merge_asv1.py). A detailed version of the previous table was also created, with the inclusion of 3 columns for merged, only forward, and only reverse sequences identified in samples and 3 columns for the number of sequences for merged, forward, reverse

identified in each sample (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/merge_asv.py). In addition, information from Bowtie results (if single or multiple hit, global or local alignment, the maximum percent identity from the alignments, and the sequence length) were included in the detailed version of the table (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/append_bowtie_to_summary.py).

Taxonomy was assigned to Bowtie SAM alignment result using a custom Python script. A Bayesian Lowest Common Ancestor (BLCA) method [Gao et al., 2017] with MUSCLE (MUltiple Sequence Comparison by Log-Expectation) alignment [Robert C. Edgar, 2004] was applied. The previously converted curated reference database FASTA file and taxonomy table were used in the BLCA script. Only alignments with a sequence identity equal to 100 % and a query cover (the percentage of bases included in the alignment from the total of bases in the query sequence) equal or higher than 85 % were considered in the assignment. The boostrapping iteration was repeated 1,000 times [Efron, 1979]. The result was saved to a table with the sequence identifier, taxonomic rank, the percentage probability of each taxonomic rank being correctly assigned to the query sequence, and the identifier of the sequence from the database involved in the alignment (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/blca_from_bowtie.py).

A custom Python script was used to add the BLCA result to the previously created brief and detailed tables (for details see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/append_blca_to_summary.py). Finally, a custom R script (R language environment version 3.4.3 [R Core Team, 2020]) was used to keep only taxonomic ranks with a taxonomy confidence level of at least 60 %, removing any taxonomic rank below the threshold, and to group by identical taxonomic ranks while summing up their values (for the R code see https://github.com/limey-bean/Anacapa/blob/New-Master/Anacapa_db/scripts/sum_blca_for_R_by_taxon.R).

3.2.2.2 Barque

Barque pipeline version 1.7.2 [Normandeau, n.d.] (https://github.com/enormandeau/barque) was used with commands being executed in parallel using GNU parallel version 20201122 [Tange, 2020]. The configuration file is described on Source Code 3.14. Primer table was filled out with the respective forward and reverse primer sequences (forward ACTGGGATTAGATACCCC and reverse TAGAACAGGCTCCTCTAG), an amplicon size of 90 and 110 for the minimum and maximum, respectively, the reference database name, and a threshold of 100 % required for the species identity in the assignment.

Trimmomatic version 0.36 [Bolger et al., 2014] was used for trimming and filtering the reads. Bases with quality below 20 starting from the 5' end of the read were cut off until a base with the quality equal or above 20 being found. Bases with quality below 20 starting from the 3'

end of the read were cut off until a base with the quality equal or above 20 being found. Starting at the 5' end and moving one base each time, when the average quality of a group (window) of 20 bases dropped below the threshold of 20, the sequence was cut off from the rightmost base within the group with quality equal or above the threshold (ignoring the last position that is always removed) to the 3' end. All reads with length smaller than 90 were removed. Bases at the 3' end of the read were cut off regardless of quality and only 126 bases were kept (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/util/trimmomatic.sh).

Forward (R1) and reverse (R2) reads from paired-end (PE) sequencing were merged using FLASH (Fast Length Adjustment of SHort reads) version 1.2.11 [Magoč & Salzberg, 2011] with a minimum required overlap of 15 bases and a maximum expected overlap of 126 bases between the two sequences. Overlaps longer than 126 were still considered, but the maximum allowed mismatch ratio (ratio between the number of mismatches and overlap) was calculated over the maximum overlap rather than the overlap between the sequences (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/02_merge.sh).

A custom Python script was used to remove primers (forward ACTGGGATTAGATACC CC and reverse TAGAACAGGCTCCTCTAG) from both ends of the sequences with maximum number of mismatches between primer and sequence equal to 2 (sequences with higher number of differences were discarded) and only keeping the sequences when both primers were found. A reverse complement of the sequence was created if primer was found in the reverse order. In addition, sequences shorter than 90 and larger than 110 bases after removing primers were discarded (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/util/split_amplicons_one_file.py). Finally, the files were converted from FASTQ to FASTA format (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/util/fastq_to_fasta.py).

VSEARCH version 2.14.2 [Rognes et al., 2016] was used to group identical sequences (sequences with the same nucleotide in each position and same length), while keeping only one sequence representing the group. During the full length dereplication sequences smaller than 20 bases were discarded. Grouped sequences were written in a file sorted by decreasing abundance with the header of the first sequence. The number of sequences found in each cluster (abundance) was also written at the end of the respective FASTA header line. Finally, the width of the lines of the new FASTA files were not wrapped (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/05_chimeras.sh#L23-L25).

VSEARCH [2016] was also used to remove chimeric sequences. Chimera detection was performed without using a reference database. Chimeric, non-chimeric, and borderline sequences were outputted to the respective FASTA file without wrapping the line width (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/05_chimeras.sh#L30-L33). After removing chimeric sequences, VSEARCH [2016] was used to convert the curated reference database FASTA file to a binary UDB database file in order to speed up

searching. Sequences smaller than 20 bases were discarded and the new UDB file was saved with the same name with the addition of the suffix .vsearchdb (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/06_vsearch.sh#L23). A custom Python script was used to convert from the VSEARCH [2016] unique FASTA format to the Barque FASTA format, where the header was changed to a sequence identification and the number of sequences found in each cluster (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/util/fasta_format_non_chimera.py).

Reads surviving previous steps were compared to the reference database by using a global pairwise alignment implemented in VSEARCH [2016]. The previously converted UDB database was used in the global alignment, using 10 CPU cores to arrange the sequences. Low-complexity sequences (simple sequence repeats [Orlov & Potapov, 2004]) were not masked in both the read sequences and the reference database sequences. A threshold of 100 % for the sequence identity was used (the number of matching nucleotides is divided by the alignment length minus terminal gaps, which are gaps filling the whole extension of the alignment in either 5', 3' or both ends) when aligning the query sequence to the reference database. A threshold of 85 % for the sequence query cover was used (the percentage of the read sequence included in the alignment with the sequence in the reference database) when aligning the query sequence to the reference database. After pairwise alignment, the search was stopped when either 20 alignments met the accept criteria of having 100 % for the sequence identity and 85 % for the sequence query cover, or 20 alignments did not meet the accept criteria. The best 20 hits between query sequences and the reference database sorted by decreasing identity were written to the output files. The result of the global alignment was written to a file in a twelve fields blast like format, where each line is a match between the query sequence and the reference database sequence. Sequences from the reference database were written to a FASTA file if matched at least one query sequence. The width of the lines of the new FASTA files were not wrapped and sequences smaller than 20 bases were discarded (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/06_vsearch.sh#L45-L50).

Tables with read counts per sample at species, genus, and phylum level, and a table showing multiple hits (read sequences that cannot be unambiguously classified to just one species and were assigned to all the possibilities [Normandeau, n.d.]) were generated using a custom Python script (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/07_summarize_results.py). A table with number of reads dropout after each step was generated using a custom shell script (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/08_summarize_read_dropout.sh). A graphical representation figure showing the dropout was generated using a R script executed in R version 3.6.1 [R Core Team, 2020] (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/util/create_read_dropout_figure.R).

A FASTA file with the 1,000 most frequent non-annotated sequences for all samples and

FASTA files with unique non-chimeric sequences, FASTA files with unique non-annotated sequences, and lists with identification of annotated sequences for each sample were generated (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/09_get_most_frequent_non_annotated_sequences.sh). Fasta files were created with all sequences involved in a multiple hit (sequences from the reference database and read sequences that cannot be unambiguously classified) using a custom Python script (for details see https://github.com/enormandeau/barque/blob/master/01_scripts/12_extract_multiple_hit_sequences.py).

3.2.2.3 MetaBEAT

FASTQ raw read data was processed and taxonomy assigned using a docker (version 19.03.6) container [Merkel, 2014] of the metaBEAT pipeline version 0.97.10 (https://github.com/HullUni-bioinformatics/metaBEAT) [Hahn & Lunt, n.d.]. Data processing workflow was done using a jupyter notebook [Kluyver et al., 2016] based on Hänfling et al. [2016] workflow (Source Code A.13).

The list of FASTQ file names was parsed and used to create a tab-separated text file where the first, second, third, fourth, and fifth columns represents the file name, forward (R1) file path, reverse (R2) file path, length of the forward primer, and length of the reverse primer, respectively (Source Code A.13). The previously created file was parsed (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L1722-L1779).

Trimmomatic version 0.32 [Bolger et al., 2014] was used for quality trimming using phred 33 (quality zero starting with the character 33 in the ascii table [Cock, Fields, Goto, Heuer, & Rice, 2009]) quality score and 10 CPU cores of the computer. Log files showing information about the execution were generated. Bases were cut off until a base with the quality equal or above 20 being found for both 5' end and 3' end. The average quality of a group (window) of 5 bases was assessed. Starting at the 5' end of the sequence, when the average of a window dropped below 20, the quality inside the window was checked and the sequence was cut off from the rightmost base within the group with quality equal or above the threshold to the 3' end (the last position is always removed and ignored on cheking). The sequences were shortened to a length of 110 bases cutting off the 3' end. Read sequences smaller than 90 bases were discarded. The first 18 bases (primer size) were cropped out of the reads for the forward (R1) and reverse (R2) files (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L2135-L2177).

FLASH (Fast Length Adjustment of SHort reads) version 1.2.11 [Magoč & Salzberg, 2011] was used to merge forward (R1) and reverse (R2) reads from paired-end (PE) next-generation sequencing. A maximum overlap of 106 (expected in 90 % of read pairs) was applied (the ratio between the number of mismatches and overlap was calculated based on maximum overlap instead of the real overlap of the sequences). A total of 10 CPU cores working

in parallel were adopted to merge the sequences. Phred 33 quality score was considered when choosing the base with the higher quality when a mismatch was found and for taking the higher quality of the two bases in the overlap to be saved to the merged file. The new sequences were saved to files using the prefix of the trimmed file and compressed using the gzip format. The merged reads, forward (R1) reads after the trimming process, and forward (R1) unmerged reads were combined into a single file, keeping only sequences with length between 85 and 127, which is a length of 106 bases, but allowing a deviation of 20 %. A csv table file with the number of reads after each previous step was created (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L2244-L2444).

All samples FASTA files were merged into a single data and a global clustering was performed using VSEARCH version 1.1.0 [Rognes et al., 2016] applying a variant of the UCLUST algorithm [Robert C. Edgar, 2010]. A threshold of 100 % for the identity was considered when clustering the sequences. In addition, both strands (sequence and reverse complement) were taken into consideration. The clustering process was executed using 10 CPU cores working in parallel. A FASTA file was created with representative sequences (centroid) for each cluster. Clustering results were saved in a 10 columns tab-separated uclust-like format with the information about the clusters. The uclust-like format file was parsed and denovo operational taxonomic units (OTU) biom tables, both in tab-separated value format and biom format, were created (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L2452-L2500). Finally, VSEARCH version 1.1.0 [Rognes et al., 2016] was used to remove chimeric sequences. The sequences were compared to the curated reference database FASTA file. Chimeric and non-chimeric sequences were saved to the respective FASTA file (Source Code A.13).

A tab-separated text file containing the list of nonchimeras FASTA files was created (Source Code A.13). The first, second, and third columns represent the name of the file, the format, and the path to the file, respectively. The newly text file was parsed (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L1722-L1779). VSEARCH version 1.1.0 [2016] was used to cluster sequences for each FASTA file. A method based on the UCLUST algorithm [Robert C. Edgar, 2010] created to maximize the speed was used in the clustering process. A threshold of 100 % for the identity was applied and both strands of the sequence were considered. During the clustering execution 10 CPU cores working in parallel were used. New FASTA files were created with the clustered sequences, keeping only the centroid sequences representing the clusters. A tab-separated file with information about the clusters (uclust-like format) were also generated. In addition, The number of sequences in each cluster was counted and two new FASTA files for each sample were created representing centroid sequences with clusters of size above or equal 3 and clusters of size smaller than the threshold. Finally, a csv table file with information about the clustering was created with the identity threshold, the total number of

clusters, the threshold for removing small size cluster, the number of cluster after removing small size cluster, and the number of sequences that represent clusters of size above or equal the threshold (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L2340-L2393).

After filtering the centroids, all FASTA sample files were joined into a unique global file and VSEARCH version 1.1.0 [Rognes et al., 2016] was used to cluster the sequences. A similar algorithm to UCLUST [Robert C. Edgar, 2010] modified to maximize the speed was executed in 10 CPU cores working in parallel for clustering, with a identity of 100 %. Both sequence and reverse complement of the sequence were considered. A FASTA file and a tab-separated text file (uclust-like format) with information about the clusters was generated. The uclust-like file was parsed and denovo operational taxonomic units (OTU) biom tables were created in TSV and biom formats (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L2452-L2500).

Biopython [Cock, Antao, et al., 2009] "NcbiblastxCommandline" interface to the BLAST+ suite version 2.2.28+ [Camacho et al., 2009] command line blastn (nucleotide query compared to a nucleotide database) was used to align the sequences from the global FASTA file to the BLAST database. A total of 10 CPU cores (*–threads 10*) working in parallel were used in the alignment. The first 50 hits for each sequence alignment that were smaller or equal than an E-value (a value calculated based on query sequence length, length of the database, and alignment score that symbolize the number of hits expected to be found by chance) [Fassler & Cooper, 2011] of 1×10^{-20} were saved to an output file in an extensible markup language (XML) format, a text file that use custom tags to represent the data [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, et al., 2000] (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L2560-L2588).

The result from BLAST alignment was parsed using Biopython [Cock, Antao, et al., 2009] "NCBIXML" parser and filtered. Alignments were filtered out if bit score (the space to search before fiding an identical or better score by chance) was smaller than 80, query coverage (percentage of the read sequence aligned to the database) was smaller than 85 %, sequence identity (how identical are the sequences aligned, based both in the number of mismatches and length of the sequence) was smaller 100 %, and if bit score was smaller than the maximum value of the bit score for the alignments of the sequence [Fassler & Cooper, 2011]. The taxonomic identifier from the species involved in the hits were extracted. After, the taxtastic suite version 0.8.5 (https://github.com/fhcrc/taxtastic) [Fred Hutchinson Cancer Research Center, Computational Biology, n.d.] was used to summarize the taxonomic information in a table representing the taxonomic lineages for the taxonomic identifiers. In addition, the taxonomic rank of hits based on the lowest common ancestor (LCA) method was determined (when multiple hits to different species happens in the alignment, the closest taxonomic rank level mutually shared by the hits is assigned to the sequence). Finally, three pair of files (tab-separated

table and biom formats) were generated. One with the number of clusters for each taxonomic rank in each sample identifier, another with the number of reads for each taxonomic rank in each sample identifier, and the last one with the number of reads sequences in each cluster for each cluster identifier in each sample identifier (for details see https://github.com/HullUni-bioinformatics/metaBEAT/blob/v0.97.10/scripts/DEVEL/metaBEAT_global.py#L2590-L2630).

3.2.2.4 MiFish

The publicly available bioinformatics MiFish pipeline (https://doi.org/10.5061/dryad.54v2q) [Miya et al., 2015; Sato et al., 2018] was used for the data processing and taxonomic assignments. Forward (R1) and reverse (R2) FASTQ files of all samples were merged in two unique records and the sequence quality was assessed by the program FastQC version 0.11.9 [Andrews et al., 2010] (Source Code 3.15). Read sequences were trimmed to the longest contiguous subsequence for which bases quality were greater than 20 using DynamicTrim.pl version 1.13 from SolexaQA software package [Cox et al., 2010] (Source Code 3.16).

```
1 fastqc *fastq
```

Source Code 3.15: Example of execution of the tool used by the MiFish J01_Fastqc.sh script (https://doi.org/10.5061/dryad.54v2q).

```
1 perl DynamicTrim.pl input_file.fastq -h 20 -d output_directory
```

Source Code 3.16: Example of execution of the tool used by the MiFish J02_TailTrimming.sh script (https://doi.org/10.5061/dryad.54v2q).

Forward (R1) and reverse (R2) reads from paired-end (PE) sequencing were merged using FLASH (Fast Length Adjustment of SHort reads) version 1.2.11 [Magoč & Salzberg, 2011] considering both "innie" (overlap between the 3' end of the forward sequence and 5' end of the reverse sequence) and "outie" (overlap between the 5' end of the forward sequence and 3' end of the reverse sequence) orientations. A minimum overlap of 15 bases between R1 and R2 sequences was required. For the maximum overlap a upper limit of 150 bases was expected. For overlaps longer than 150, the ratio between the number of mismatches and overlap was calculated over the maximum overlap option ignoring the overlap of the alignment. Merged FASTQ files were saved using the same sample name as prefix inside a new folder for the merged sequences (Source Code 3.17). Sequences with at least one ambiguous base represented by the letter N (which could be any nucleotide according to the IUPAC nomenclature code [Cornish-Bowden, 1985]) were removed (Source Code 3.18). Sequences with length smaller than 90 bases or larger than 150 bases (120 ± 30) were also removed (Source Code 3.19).

```
flash input_file_R1.fastq.trimmed input_file_R2.fastq.trimmed -O -m 15 -M 150 -o
    ↪ prefix_name -d output_directory
```

Source Code 3.17: Example of execution of the tool used by the MiFish J03_PE_read_assembly.sh script (https://doi.org/10.5061/dryad.54v2q). The options -O (–allow-outies), -m (–min-overlap), and -M (–max-overlap) were included to the original version of the script.

```
perl Fastq_Nread_trim.pl input_file.fastq >output_file.Ntrimmed.fastq
```

Source Code 3.18: Example of execution of the tool used by the MiFish J04_RemoveN.sh script (https://doi.org/10.5061/dryad.54v2q). The custom Perl script Fastq_Nread_trim.pl can be found on https://doi.org/10.5061/dryad.54v2q.

```
perl check_seq_length_MiFish.pl input_file.fastq >output_file.MiFish.fastq
```

Source Code 3.19: Example of execution of the tool used by the MiFish J05_Length_check_MiFish.sh script (https://doi.org/10.5061/dryad.54v2q). The custom Perl script check_seq_length_MiFish.pl can be found on https://doi.org/10.5061/dryad.54v2q.

TagCleaner version 0.16 [Schmieder et al., 2010] was used to remove primers sequences (forward ACTGGGATTAGATACCCC and reverse TAGAACAGGCTCCTCTAG). The forward primer was applied in the original direction (*-tag5 ACTGGGATTAGATACCCC*), whereas the reverse primer was applied using the reverse complement direction of the primer sequence (*-tag3 CTAGAGGAGCCTGTTCTA*). During the execution the status was printed to the terminal. The FASTQ files were transformed into FASTA format and saved in a new folder after removing primers. Reads not matching the primer sequences at either end were filtered out, with a maximum allowed mismatches of 4 bases at the 5' end and also the 3' end. Finally, log files were generated showing for each primer all different number of mismatches found between the primer and the read sequences (from zero to the maximum detected). For each different number of mismatch, the number of sequences and the percentage of sequences found were also informed (Source Code 3.20). For the original data and all the files resulted from the previous steps, the number of reads was counted and saved in a text file (Source Code 3.21).

```
perl tagcleaner.pl -verbose -fastq input_file.fastq -out_format 1 -out output_file.
    ↪ MiFish_processed -nomatch 3 -mm3 4 -mm5 4 -tag3 CTAGAGGAGCCTGTTCTA -tag5
    ↪ ACTGGGATTAGATACCCC

perl tagcleaner.pl -verbose -fastq input_file.fastq -out_format 1 -stats
    ↪ output_directory -out output_file.MiFish_processed -nomatch 3 -mm3 4 -mm5 4 -
    ↪ tag3 CTAGAGGAGCCTGTTCTA -tag5 ACTGGGATTAGATACCCC > output_file.log
```

Source Code 3.20: Example of execution of the tool used by the MiFish J06_Primer_removal_MiFish.sh script (https://doi.org/10.5061/dryad.54v2q).

```
ls *MiFish_processed.fasta | sort -d | sed s/'.*\/'//g | sed s/'.MiFish_processed.
    ↪ fasta'//g >> output_file_NAMES.txt

grep -c '^>' *MiFish_processed.fasta | sort -d | sed s/'.*:'//g >>
    ↪ output_file_READ_counts.txt
```

Source Code 3.21: Example of execution of the tools used by the MiFish J07_Processed_read_counter.sh script (https://doi.org/10.5061/dryad.54v2q).

Sequences having the same length and same nucleotides in each position were grouped (dereplicated) using USEARCH version 11.0.667 [Robert C. Edgar, 2010], keeping only one sequence representing the group. Dereplicated sequences were sorted by a decreasing order considering the cluster abundance and saved to FASTA files with the number of sequences for the cluster being written at the FASTA header. Sequences with the cluster size smaller than 10 was extracted and a new FASTA file was created to save cluster size sequences smaller than the threshold. Sequences were sorted by cluster size using USEARCH [2010] and saved to a new FASTA file, discarding sequences with size smaller than 10. A global pairwise alignment implemented in USEARCH [2010] was used to compare the FASTA file containing sequences with cluster size smaller than 10 and the FASTA file containing sequences with cluster size equal or higher than 10. The sequences were only compared in the forward orientation. A threshold of 99 % for the sequence identity (99 % identical) was used when aligning the sequences. In addition, the information about the sequences that matched were saved in a USEARCH cluster format, a tab-separated text file with 10 fields. The size of the cluster with less than 10 sequences that resulted in a identity of 99 % or higher was summed up to the size of the cluster with 10 or more sequences involved in the alignment. The smaller group was summed up to the larger cluster and a new text file was generated with the header of the larger cluster and the new size. Additionally, a new FASTA file was generated after changing the header of the dereplicated FASTA file with new clusters sizes. The uc_size_fas_integrator.pl and uc_size_processor.pl scripts were slightly modified from the original version to deal with illumina header special characters, */$OTUname/* on fas_integrator and */$otuname/* on processor were changed to */\Q$OTUname\E/* and */\Q$otuname\E/*, respectively. Finally, the new FASTA file was sorted by cluster size using USEARCH [2010] and a new FASTA file was created (Source Code 3.22).

```
usearch -fastx_uniques input_file.fasta -fastaout output_file.derep.fasta -sizeout
```

```
perl size_extracter_def.pl output_file.derep.fasta > output_file.derep.size.fasta
usearch -sortbysize output_file.derep.fasta -fastaout output_file.sizetrim.derep.
    ↪ fasta -minsize 10
usearch -usearch_global output_file.derep.size.fasta -db output_file.sizetrim.derep.
    ↪ fasta -strand plus -id 0.99 -uc output_file.size.uc
perl uc_size_processor.pl output_file.size.uc > output_file.rempd.otunmsz.txt
perl uc_size_fas_integrator.pl output_file.sizetrim.derep.fasta output_file.rempd.
    ↪ otunmsz.txt > output_file.sizetrim.sum.fasta
usearch -sortbysize output_file.sizetrim.sum.fasta -fastaout output_file.sizetrim.sum.
    ↪ fasta
```

Source Code 3.22: Example of execution of the tools used by the MiFish J10_Uclust_derep_trim.sh script (https://doi.org/10.5061/dryad.54v2q). The options -derep_fulllength and -output from the original script USEARCH version were renamed to -fastx_uniques and -fastaout, respectively, in the USEARCH version 11.0.667 used to execute the pipeline. Custom Perl scripts size_extracter_def.pl, uc_size_processor.pl, and uc_size_fas_-integrator.pl can be found on https://doi.org/10.5061/dryad.54v2q.

After being processed, reads were aligned to the reference database using the command line blastn (nucleotide query compared to a nucleotide database) from NCBI BLAST+ suite (Basic Local Alignment Search Tool) version 2.10.0+ [Camacho et al., 2009]. The read query sequence was compared to the reference database using a threshold of 100 % for the sequence identity (the original 97 % was changed to 100 %), returning the first 5 hits that were smaller or equal than an E-value of 0.00001. The output file name was informed and the results were saved in a tabular format with information about the id of the sequence in the reference database involved in the match, percentage of identity, alignment length, number of mismatches, number of gap openings, expect value (e-value represents the number of hits expected to be found by chance and it is calculated based on query sequence length, length of the database, and alignment score), bit score (the size of the database that would make the alignment being found by chance), and the aligned part of query sequence [Fassler & Cooper, 2011] (Source Code 3.23).

```
blastn -query input_file.fasta -db reference_database -max_target_seqs 5 -
    ↪ perc_identity 100 -evalue 0.00001 -outfmt "7 sseqid pident length mismatch
    ↪ gapopen evalue bitscore qseq" -out blastn_res.txt -html
```

Source Code 3.23: Example of execution of the tool used by the MiFish J11_Blastn.sh script (https://doi.org/10.5061/dryad.54v2q).

The result was parsed and a new file was generated with the list of hits from the BLAST result (cluster size in the first column, followed by the species information or no hit in the second column, and the query sequence identifier information in the third column). Hits for the same species were summed up and new file was generated (species information or not hit in the

first column, cluster size in the second one, and sequence for hits or sequence identifier for no hit in the third column). Finally, the LOD (logarithm of the odds) score was calculated. The LOD score was calculated to compare the genetic linkage (likelihood of two genetic loci being linked) between the sequences involved in a hit [Risch, 1992] (Source Code 3.24). LOD score files were parsed and a list with all species detected was created (Source Code 3.25). Finally, a table with samples identifier in the first column, the species names as columns, and the number of reads classified in the cells intersection representing species classified for the sample was created (Source Code 3.26).

```
perl blastres_parser_v5.pl input_file > blastn.deprep.list.txt
perl blastres_parse_counter_v4.pl blastn.deprep.list.txt > blastn.deprep.counts.txt
perl blastres_parser_LODs_v2.pl input_file > blastn.LODlist.txt
```

Source Code 3.24: Example of execution of the tools used by the MiFish J12_Blastres_counts.sh script (https://doi.org/10.5061/dryad.54v2q). Custom Perl scripts blastres_parser_v5.pl, blastres_parse_counter_v4.pl, and blastres_parser_LODs_v2.pl can be found on https://doi.org/10.5061/dryad.54v2q.

```
less -f blastn.LODlist.txt | awk -F "\t" '{print $2}' >> specieslist.temp.txt
less -f specieslist.temp.txt | grep -v '^No hits found.*' | sort | uniq > specieslist.
    ↪ txt
```

Source Code 3.25: Example of execution of the tools used by the MiFish J13_Allspecies_list_make.sh script (https://doi.org/10.5061/dryad.54v2q).

```
perl allsamples_nameprinter_v1.pl specieslist.txt >> species_table.tsv
perl allsamples_species.counter_v2.pl blastn.deprep.counts.txt specieslist.txt >>
    ↪ species_table.tsv
```

Source Code 3.26: Example of execution of the tools used by the MiFish J14_Allsamples_table_make.sh script (https://doi.org/10.5061/dryad.54v2q). Custom Perl scripts allsamples_nameprinter_v1.pl and allsamples_species.counter_v2.pl can be found on https://doi.org/10.5061/dryad.54v2q.

3.2.2.5 SEQme

The metabarcoding data analysis was presented by the SEQme private company during the Microbiome and Metagenome Data analysis workshop [SEQme, 2018]. A python script was created to automatize each step of the analysis (Source Code A.14). Forward (R1) and reverse (R2) reads from paired-end (PE) sequencing were merged using fastq-join version 1.3.1 [Aronesty, 2013]. Sequences were verified if the forward identifier and the reverse identifier matched up from the beginning of the header to the space character (Illumina reads use space

before the read number, which is 1 for forward or 2 for reverse in a paired-end sequencing). Everything after the space was ignored in the verification. Overlapping sequences with a number of mismatches higher than 15 % were discarded. A minimum overlap of 15 bases between the forward and the reverse sequences was required. Three new files were generated after merging the reads, one for the merged sequences and two (forward and reverse) for not merged sequences (for details see Source Code A.14 line 46).

Command line fastq_quality_filter from FASTX-Toolkit version 0.0.14 [Gordon, Hannon, et al., 2010] was used for quality filtering of the reads. Fastq files use one symbol per quality value. The quality score value plus the phred type defines the symbol used to represent the quality. For a score equal to zero and a phred 33 a exclamation (!) mark is used, because exclamation has the code 33 in the ASCII table [Cock, Fields, et al., 2009]. Read sequences with less than 50 % of the bases having quality higher or equal to 20 for phred 33 were discarded (for details see Source Code A.14 lines 47 and 48). Command line fastq_to_fasta from FASTX-Toolkit version 0.0.14 [Gordon, Hannon, et al., 2010] was used for the conversion of the FASTQ files to FASTA format (for details see Source Code A.14 line 49).

Command lines read_fasta, grab, and write_fasta from Biopieces bioinformatic framework version 2.0 [Hansen et al., 2008] were used to remove short and long sequences. First of all, the FASTA files were read using read_fasta, which results in the sequence identifier, nucleotides sequence, and the sequence length. Secondly, the result was filtered using the grab command line with the option -e, which evaluates the key (sequence length), the operator (>= and <=), and the value (value to be kept). Sequences shorter than 90 and larger than 150 were discarded. Finally, the sequences were saved to a FASTA file using the command line write_fasta without being printed to the terminal (for details see Source Code A.14 lines from 50 to 53).

USEARCH version 11.0.667 [Robert C. Edgar, 2010] was used to dereplicate sequences with identical length and nucleotide composition. The sequences were grouped and only one sequence having the cluster size at the end of the respective FASTA header line was kept to represent the group. The unique sequences were saved to a FASTA file following a decreasing order of abundance. A tab-separated text file in a USEARCH cluster format with 10 fields was created with the information about clusters and sequences that matched. Finally, the sequence identifier was renamed to "Uniq" followed by a integer representing the position in the FASTA file (for details see Source Code A.14 lines from 54 to 56).

After the dereplication step, closely related sequences with 97 % of identity were clustered into operational taxonomic units (OTU) using UPARSE [Robert C Edgar, 2013] OTU clustering algorithm from USEARCH version 11.0.667 [Robert C. Edgar, 2010]. Chimeric sequences were also removed during the clustering step. Fasta output file were saved with the OTU sequences without the number of sequences for the cluster in the sequence identifier. Finally, the sequence identifier was renamed to "Otu" followed by a integer representing the position in the FASTA file (for details see Source Code A.14 lines 57 and 58).

Reads resulted after removing short and long sequences were mapped to OTUs with the highest identity higher or equal than a threshold of 97 % using USEARCH version 11.0.667 [2010]. The sequences were mapped to the corresponding FASTA OTU file. If a tie was found, the first OTU in the increasing order was taken. The OTU identifiers and the number of reads mapped to each OTU were saved to an OTU table file in QIIME classic format, a tab-separated text with the header representing the OTU identifier in the first column (#OTU ID) and read sequences identifier in each remaining columns. The header is followed by a unique OTU identifier in each new line in the first column and the number of sequences in the OTU for each read sequences identifier in the remaining columns. The information of mapping were also saved to a map file, a tab-separated text file where the first column represents the read sequence identifier and the second column represents the OTU identifier (for details see Source Code A.14 lines from 59 to 61).

The Ribosomal Database Project (RDP) classifier version 2.11 [Wang et al., 2007] was used for the taxonomic classification of the read sequences. The RDP classifier is a naïve Bayesian classifier that assigns data into labeled classes based on Bayes theorem (probability of an event happening based on prior observed data) assuming independence of the features [Rish et al., 2001; Puga, Krzywinski, & Altman, 2015]. The classifier uses all possible subsequences of 8 bases as features [Wang et al., 2007]. The curated reference database FASTA and the hierarchical taxonomy information files previously generated using a custom Python script (Source Code 3.11) were used to train the classifier (Source Code 3.27). The result was saved to a new folder named "Classifier". A classifier properties file was created inside the new folder to set the path to each respective file (Source Code 3.28).

```
1 classifier train -o Classifier -s reference\_database.fasta -t reference\_database\
    ↪ _taxid.txt
```

Source Code 3.27: Trainning classifier using the reference database and the taxonomic table.

```
1  # Sample ResourceBundle properties file
2  bergeyTree=bergeyTrainingTree.xml
3
4  probabilityList=genus_wordConditionalProbList.txt
5
6  probabilityIndex=wordConditionalProbIndexArr.txt
7
8  wordPrior=logWordPrior.txt
9
10 classifierVersion=RDP Naive Bayesian rRNA Classifier Version 2.5, May 2012
```

Source Code 3.28: Classifier properties file.

The RDP classifier trained with the curated database was used to classify the OTUs to species level. For each sequence, all subsequences of 8 bases were created. In each bootstrap iteration (resampling the dataset) [Efron, 1979], 150 subsequences (comprising all subsequences) were used to calculate the joint probability. The boostrapping step was repeated 100 times. The number of times a taxonomic rank was classified out of all iterations were used to estimate the confidence level [Wang et al., 2007]. The result was saved to tab-delimited text files with the first column representing the OTU identifier followed by taxonomic rank name, taxonomic rank, and the confidence level for each taxonomic rank from superkingdom to species level. In addition, taxonomic hierarchical tab-delimited files showing only taxonomic rank with 100 % of confidence were also created (for details see Source Code A.14 lines from 66 to 68). Finally, for each sample all OTUs with 100 % of confidence in species level were parsed. For each OTU having 100 %, the number of sequences in the cluster was parsed from the OTU file. OTUs classified to the same species were summed up and a tab-delimited text file was created with species in the first column, sample identifiers as columns, and the number of reads classified put in the cell intersection between each species and sample (for details see Source Code A.14 lines from 508 to 590).

3.2.3 Data Analysis

The number of sequence reads after each step in the workflow of the pipelines execution was counted using a python script (Source Codes 3.29 and A.15). Data analyses were conducted inside R language environment version 3.6.3 [R Core Team, 2020]. A threshold was applied in each sample to remove false positive species assignment where the number of reads assigned fell below 0.1 % considering the sample total of reads [Hänfling et al., 2016] (Source Code A.16). A custom R script was used to create new tables for data analysis (Source Codes A.17 and A.18). Number of reads was calculated by aggregating and summing up the values for pipelines, reservoirs and seasons. The same was applied to obtain the number of species (Source Code A.19).

```
1 python Count_Reads.py path_to_folder fastq "new file row name" --
    ↪ pattern_of_the_file_name_to_be_searched
```

Source Code 3.29: Execution of the Source Code A.15.

Alpha and beta diversity were calculated using Vegan community ecology package version 2.5-6 [Oksanen et al., 2019]. For the alpha diversity richness the number of species was counted (Source Code A.20). The alpha diversity describe the number of species in a determined group [Whittaker, 1972]. The shannon index, which accounts not only for the richness of species but also the number of reads for each one (the higher the richness and the evenness, the higher the

index) [Shannon, 1948], was calculated using *diversity* function from Vegan package [Oksanen et al., 2019] (Source Code A.21). In addition to the alpha diversity, which considers only the diversification within a particular group, the beta diversity also quantifies the difference in species composition from one group to another [Whittaker, 1972]. The beta diversity was calculated using *vegdist* function from Vegan package [Oksanen et al., 2019]. The Jaccard index, which accounts for the presence or absence of species among groups [Koch, 1957], was the method used to compute the dissimilarity indices, with the smallest and the largest showing the most and the least similar distances between two groups, respectively (Source Code A.22).

An analysis of variance (ANOVA) and post-hoc Tukey tests were performed to test whether the difference in alpha diversity among the groups was statistically significant and compare the diversity of each group against each other. Functions *aov*, *anova*, and *TukeyHSD* from R language version 3.6.3 [R Core Team, 2020] stats package were used to perform the analysis of variance and calculate Tukey's Honest Significant Difference (Source Codes A.20 and A.21). Regarding beta diversity, *adonis* function from Vegan package [Oksanen et al., 2019] was used to perform a permutational multivariate analysis of variance (PERMANOVA) to test whether the species composition among the groups had statistically significant difference. Finally, *cmdscale* function from stats package in R language version 3.6.3 [R Core Team, 2020] was used to apply the principal coordinates analysis (PCoA) method to better represent the distances and relationships among pipelines dissimilarity indices in a low-dimensional visualization (Source Code A.22).

For each pipeline, it was calculated the percentage of assigned reads to *Maylandia zebra* based on the initial total of reads from demultiplexed samples used as the input data. An ANOVA test was applied using the *anova* function from R language version 3.6.3 [2020] stats package to test wether pipelines had a statistically significant difference for the detection of the positive control. The *TukeyHSD* function from stats package was also used to perform a Tukey test to assess the statistical significant difference between each pair of pipelines (Source Code A.23).

Results

4.1 Number of sequence reads

4.1.1 Number of sequence reads after each pipeline execution

From the 220 samples, including positive and negative controls, collected in Klíčava, Římov, and Žlutice in autumn and in summer, sequencing the libraries with Illumina Miseq generated 22.46 million raw sequence reads. Out of the reads, 93.08 % (20,910,517 reads) remained after demultiplexing. The demultiplexed reads were used as the input data. With the execution of the pipelines, 85.95 % (19,307,168 reads on average, ranged from 18,513,853 to 20,910,517) remained after trimming, 81.81 % (18,377,199 reads on average, ranged from 17,145,436 to 19,384,073) after merging, 75.92 % (17,054,961 on average, ranged from 13,876,672 to 18,271,442) after filtering and chimera removal, and 46.33 % (10,407,476 on average, ranged from 8,940,480 to 11,112,721) were assigned to species after applying a false positive sequence threshold of 0.1 % to remove for each sample any read frequencies below the threshold (Table 4.1).

Data Processing Steps	Anacapa	Barque	MetaBEAT	MiFish	SEQme *
Total from original data	22,464,147	22,464,147	22,464,147	22,464,147	22,464,147
Total after demultiplexing	20,910,517	20,910,517	20,910,517	20,910,517	20,910,517
Trimmed and filtered	19,095,153	18,632,248	18,513,853	20,910,517	19,384,070
Merged	18,944,446	18,619,523	17,792,519	17,145,436	19,384,073
Filtered and chimera removed	18,271,442	17,889,794	17,782,513	13,876,672	17,454,382
Assigned	10,676,765	11,112,721	10,347,227	8,940,480	10,960,189
Unassigned (original data)	11,787,382	11,351,426	12,116,920	13,523,667	11,503,958
Unassigned (demultiplexed data)	10,233,752	9,797,796	10,563,290	11,970,037	9,950,328

Table 4.1: Number of reads after each step on data processing, including positive and negative controls.
* SEQme pipeline applies merging before trimming.

4.1.2 Number of sequence reads assigned to pipelines, reservoirs, and seasons

The average number of sequence reads assigned to species taking into account all pipelines was 7,821,428, excluding positive and negative controls. The pipeline with the highest number of sequence reads was Barque with 8,410,037, whereas MiFish assigned the lowest amount with 6,820,393 reads (Table 4.2).

Pipeline	Number of reads
Anacapa	7,816,625
Barque	8,410,037
MetaBEAT	7,859,744
MiFish	6,820,393
SeqME	8,200,342

Table 4.2: Number of reads assigned to species for each pipeline, excluding positive and negative controls.

The average number of sequence reads assigned to all pipelines in each reservoir was 1,205,830 in Klíčava, 4,994,656 in Římov, and 1,620,942 in Žlutice. Regarding seasons, 2,330,853 and 5,490,575 were the number of sequence reads on average assigned in autumn and summer, respectively. When considering pipelines, reservoirs, and seasons together, the number of sequence reads assigned ranged from 113,122 (in the MiFish pipeline in Klíčava in autumn) to 3,610,686 (in the Barque pipeline in Římov in summer) (Table 4.3).

	Autumn	Summer
Klíčava	152,115	1,027,223
Římov	1,748,946	3,207,005
Žlutice	509,967	1,171,369

(a) Anacapa

	Autumn	Summer
Klíčava	139,839	1,125,257
Římov	1,826,973	3,610,686
Žlutice	535,515	1,171,767

(b) Barque

	Autumn	Summer
Klíčava	129,156	1,060,167
Římov	1,697,086	3,417,605
Žlutice	452,972	1,102,758

(c) MetaBEAT

	Autumn	Summer
Klíčava	113,122	916,496
Římov	1,490,978	2,927,711
Žlutice	432,617	939,469

(d) MiFish

	Autumn	Summer
Klíčava	135,439	1,230,336
Římov	1,729,903	3,316,388
Žlutice	559,636	1,228,640

(e) SEQme

Table 4.3: Number of reads assigned to species considering pipelines, reservoirs, and seasons, excluding positive and negative controls.

4.2 Species detection and diversity

4.2.1 Number of species detected

From a total of 58 species in the reference library, 37 species were detected considering all pipelines (Table 4.4), excluding *Maylandia zebra* positive control, and 21 were not detected in any of them (Table 4.5). A few species were removed from the detections after applying a false positive threshold to discard read frequencies below 0.1 % of the total of reads assigned in the sample (Table 4.6).

Family	Species
Petromyzontidae	*Lampetra planeri*
Acipenseridae	*Acipenser-sp.*
Anguillidae	*Anguilla anguilla*
Centrarchidae	*Lepomis gibbosus*
Cottidae	*Cottus gobio*, *Cottus poecilopus*, *Abramis brama*, *Alburnus alburnus*, *Barbus barbus*, *Carassius auratus*, *Carassius carassius*, *Chondrostoma nasus*, *Ctenopharyngodon idella*
Cyprinidae	*Cyprinus carpio*, *Gobio gobio*, *Hypophthalmichthys molitrix*, *Hypophthalmichthys nobilis*, *Aspius+Scardinius*, *L.idus+leuciscus*, *Phoxinus phoxinus*, *Pseudorasbora parva*, *Rhodeus amarus*, *Rutilus rutilus*, *Squalius cephalus*, *Tinca tinca*, *Blicca+Vimba*
Esocidae	*Esox lucius*
Gasterosteidae	*Gasterosteus aculeatus*
Nemacheilidae	*Barbatula barbatula*
Percidae	*Gymnocephalus cernua*, *Sander+Perca*
Salmonidae	*Coregonus-sp.*, *Oncorhynchus mykiss*, *Salmo trutta*, *Salvelinus fontinalis*, *Thymallus thymallus*
Siluridae	*Silurus glanis*

Table 4.4: Species in the reference library detected in at the least one of the pipelines.

Family	Species
Centrarchidae	*Micropterus salmoides*
Clupeidae	*Alosa alosa*
Cobitidae	*Cobitis elongatoides*, *Cobitis taenia*, *Misgurnus fossilis*, *Sabanejewia balcanica*
Cyprinidae	*Leucaspius delineatus*, *Romanogobio albipinnatus*
Gobiidae	*Neogobius melanostomus*, *Pomatoschistus minutus*, *Ponticola kessleri*, *Proterorhinus marmoratus*
Ictaluridae	*Ameiurus melas*, *Ameiurus nebulosus*
Lotidae	*Lota lota*
Percidae	*Gymnocephalus baloni*
Petromyzontidae	*Petromyzon marinus*
Pleuronectidae	*Platichthys flesus*
Salmonidae	*Hucho hucho*, *Salmo salar*
Umbridae	*Umbra krameri*

Table 4.5: Species in the reference library not detected in any of the pipelines.

Pipeline	Species
Anacapa	*Romanogobio albipinnatus*, *Squalius cephalus*, *Gymnocephalus cernua*
Barque	*Leucaspius delineatus*
metaBEAT	*Leucaspius delineatus*, *Squalius cephalus*, *Gymnocephalus cernua*, *Lampetra planeri*
MiFish	*Lampetra planeri*, *Leucaspius delineatus*
SEQme	*Lampetra planeri*, *Lota lota*, *Neogobius melanostomus*

Table 4.6: Species removed from the pipeline detections after discarding number of reads assigned smaller than a threshold of 0.1 % of the total of reads in the sample.

The number of species detected was 32 in Anacapa and metaBEAT, and 33 in Barque, MiFish, and SEQme pipelines. The total of species detected in each reservoir was 21, 34, and 23 in Klíčava, Římov, and Žlutice, respectively, whereas 36 species were detected in autumn and 26 in summer. When considering pipelines, reservoirs, and seasons together, the number of species detected ranged from 10 (Anacapa and SEQme pipelines in Klíčava in summer) to 29 (Barque, MiFish, and SEQme pipelines in Římov in autumn) (Table 4.7).

	Autumn	Summer
Klíčava	15	10
Římov	27	18
Žlutice	19	11

(a) Anacapa

	Autumn	Summer
Klíčava	16	12
Římov	29	22
Žlutice	20	12

(b) Barque

	Autumn	Summer
Klíčava	15	11
Římov	28	21
Žlutice	19	11

(c) MetaBEAT

	Autumn	Summer
Klíčava	16	12
Římov	29	21
Žlutice	20	12

(d) MiFish

	Autumn	Summer
Klíčava	17	10
Římov	29	19
Žlutice	18	11

(e) SEQme

Table 4.7: Number of species detected considering pipelines, reservoirs, and seasons, excluding positive and negative controls.

4.2.2 Number of sequence reads assigned to species

Based on the number of sequence reads assigned to each species in all pipelines divided by the number of pipelines (average) without positive and negative controls, *Rutilus rutilus* had the largest number with 2,437,600 reads on average, followed by *Sander+Perca* with 2,128,746 reads on average. By contrast, the smallest number was assigned to *Lampetra planeri* with 50 reads on average (the species was detected only in Anacapa with 248 reads), followed by *Barbus barbus* with 697 reads on average (Figure 4.1).

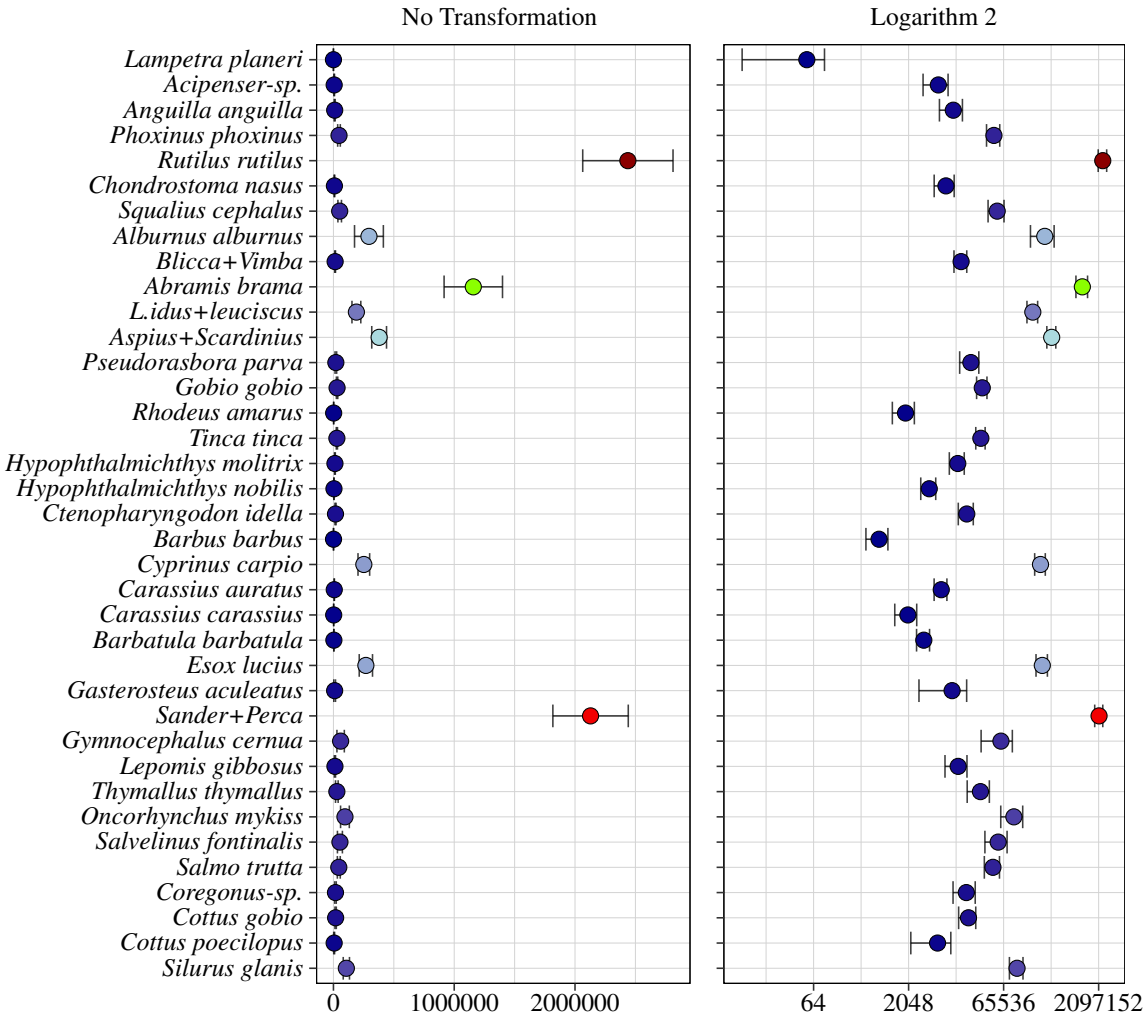

Figure 4.1: Average of number of reads assigned to species considering all pipelines where the error bars indicate the standard deviations, excluding positive and negative controls. The left plot shows the data without axis transformations, whereas the right plot shows the data with logarithm base 2 transformation applied to x axis

For each pipeline, the largest and smallest number of sequence reads assigned to species was determined. *Rutilus rutilus* (2,498,382 reads) and *Lampetra planeri* (248 reads) were found in Anacapa; *Rutilus rutilus* (2,341,339 reads) and *Barbus barbus* (708 reads) in Barque; *Rutilus rutilus* (2,225,684 reads) and *Barbus barbus* (671 reads) in metaBEAT; *Rutilus rutilus* (1,873,906 reads) and *Barbus barbus* (593 reads) in MiFish; and *Rutilus rutilus* (3,248,689 reads) and *Acipenser-sp.* (483 reads) in SEQme (Figure 4.2).

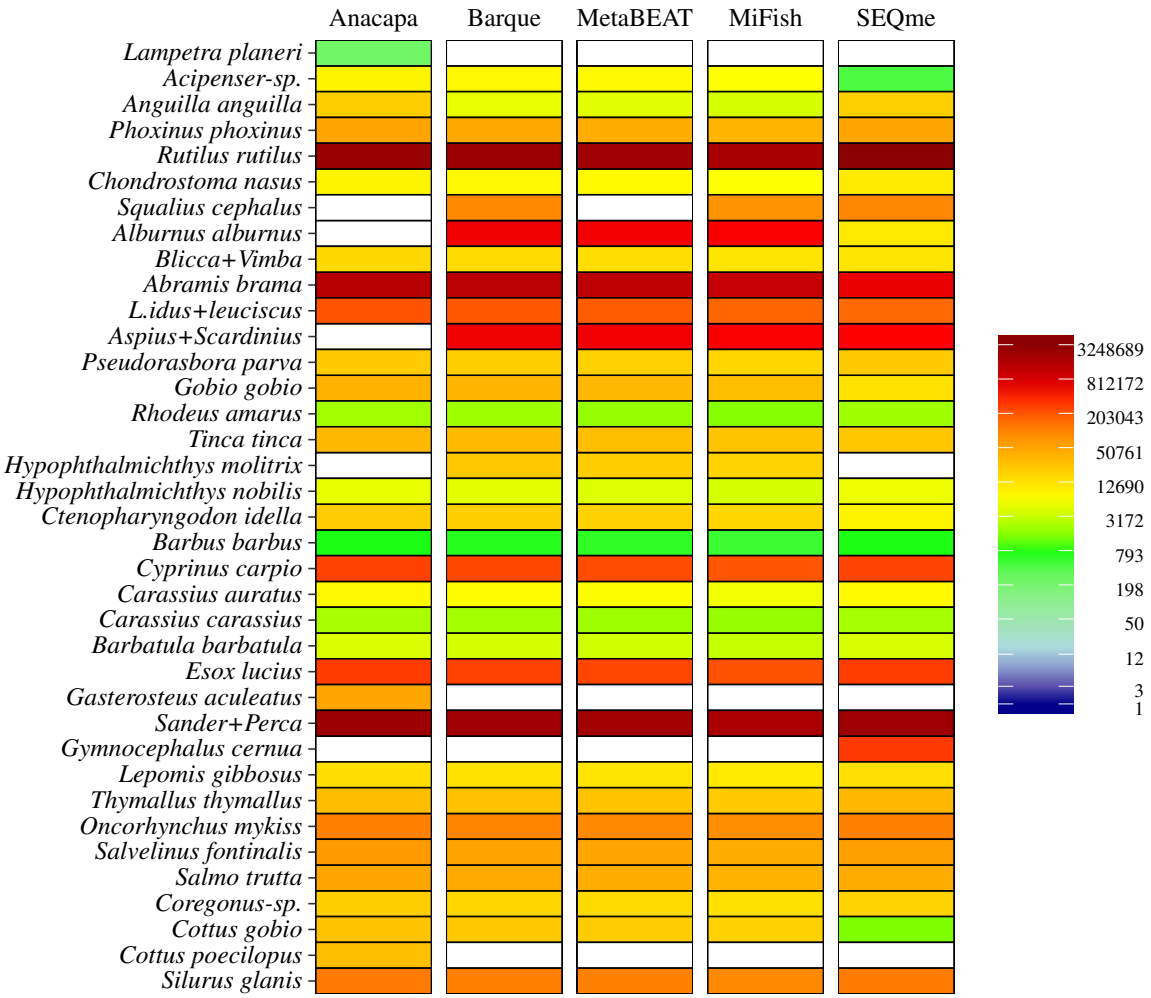

Figure 4.2: Number of reads assigned to species for each pipeline, excluding positive and negative controls.

Regarding reservoirs, the largest and smallest amounts on average in all pipelines were assigned to *Sander+Perca* (394,679 reads) and *Blicca+Vimba* (525 reads) in Klíčava, *Rutilus rutilus* (1,489,246 reads) and *Lampetra planeri* (50 reads) in Římov, and *Rutilus rutilus* (598,052 reads) and *Gasterosteus aculeatus* (618 reads) in Žlutice (Figure 4.3).

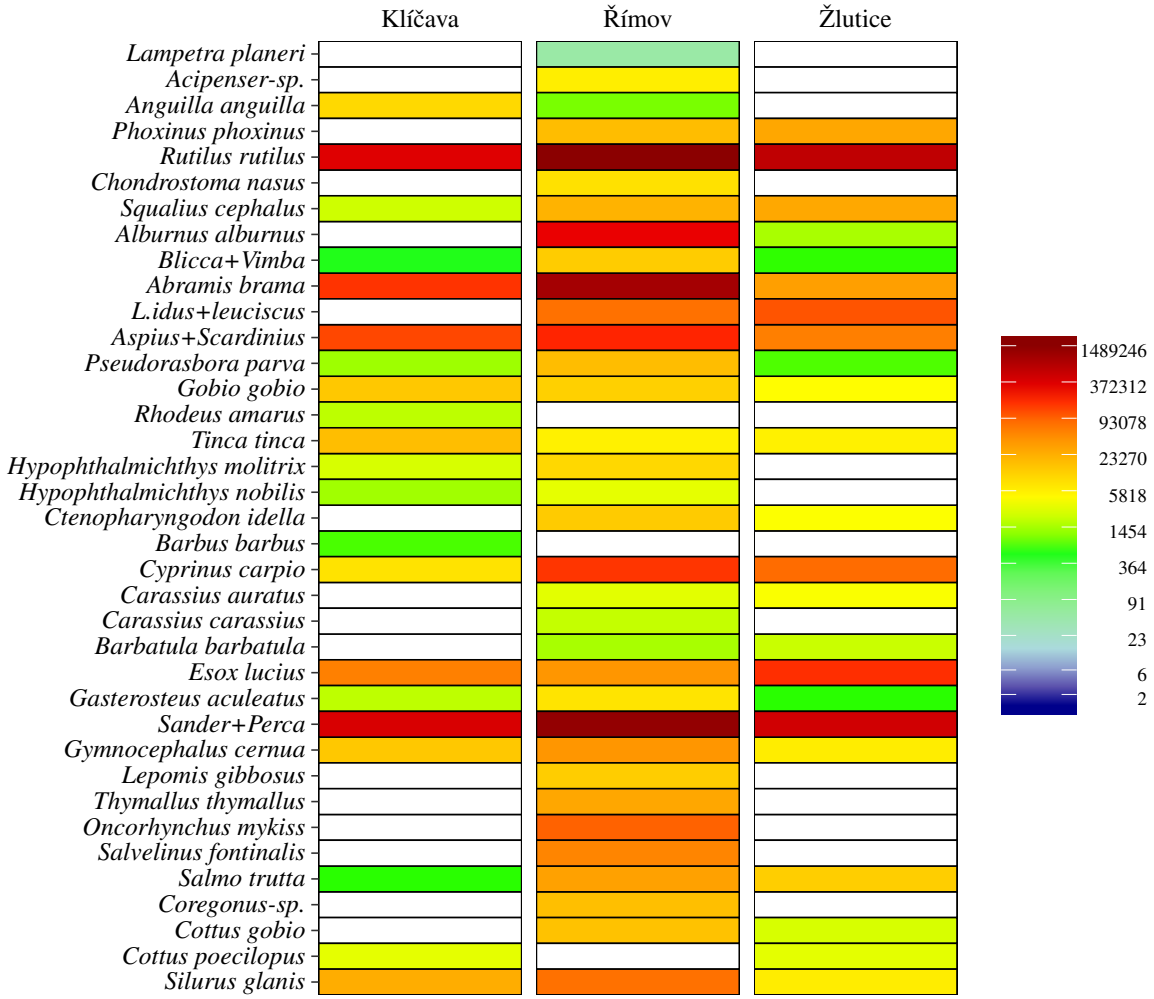

Figure 4.3: For each reservoir, average of number of reads assigned to species considering all pipelines, excluding positive and negative controls.

Regarding seasons, *Rutilus rutilus* (549,523 reads) had the largest number of sequence reads assigned to species in autumn, while *Lampetra planeri* (50 reads) had the smallest number. In summer the largest number of sequence reads was assigned to *Rutilus rutilus* (1,888,077 reads) and the smallest was assigned to *Cottus poecilopus* (102 reads) (Figure 4.4).

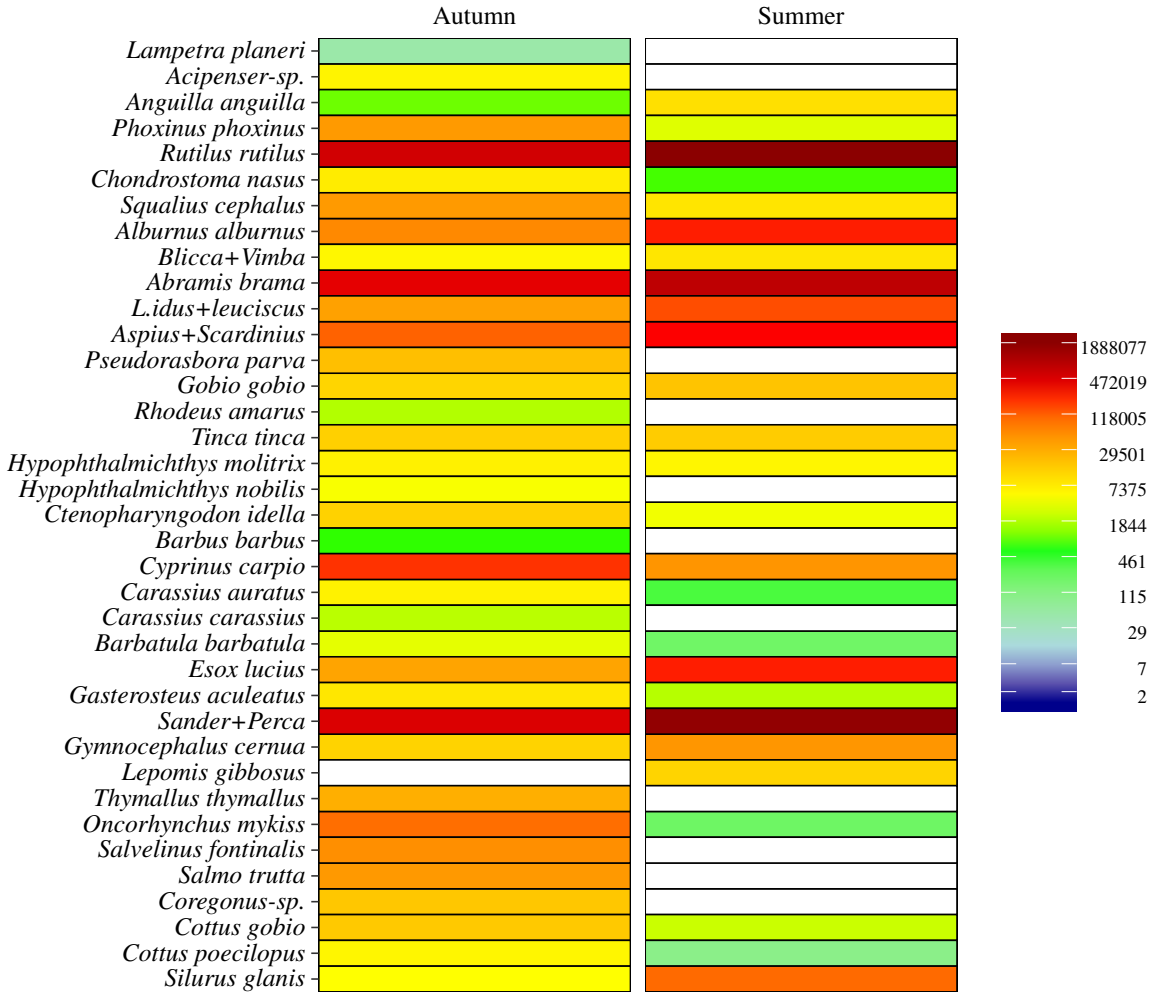

Figure 4.4: For each season, average of number of reads assigned to species considering all pipelines, excluding positive and negative controls.

When considering pipelines, reservoirs, and seasons together, *Rutilus rutilus* (1,614,121 reads) had the largest number of sequence reads. It was detected in the SEQme pipeline, in the Římov reservoir in summer. The same species also had the second largest number with 1,113,465 reads. It was detected in the same reservoir and season but in the Anacapa pipeline. As for the smallest number of sequence reads, *Barbatula barbatula* (176 reads) was detected in MiFish in Římov in Summer, whereas *Oncorhynchus mykiss* with 195 reads was found in the same pipeline, resevoir, and season (Figure 4.5).

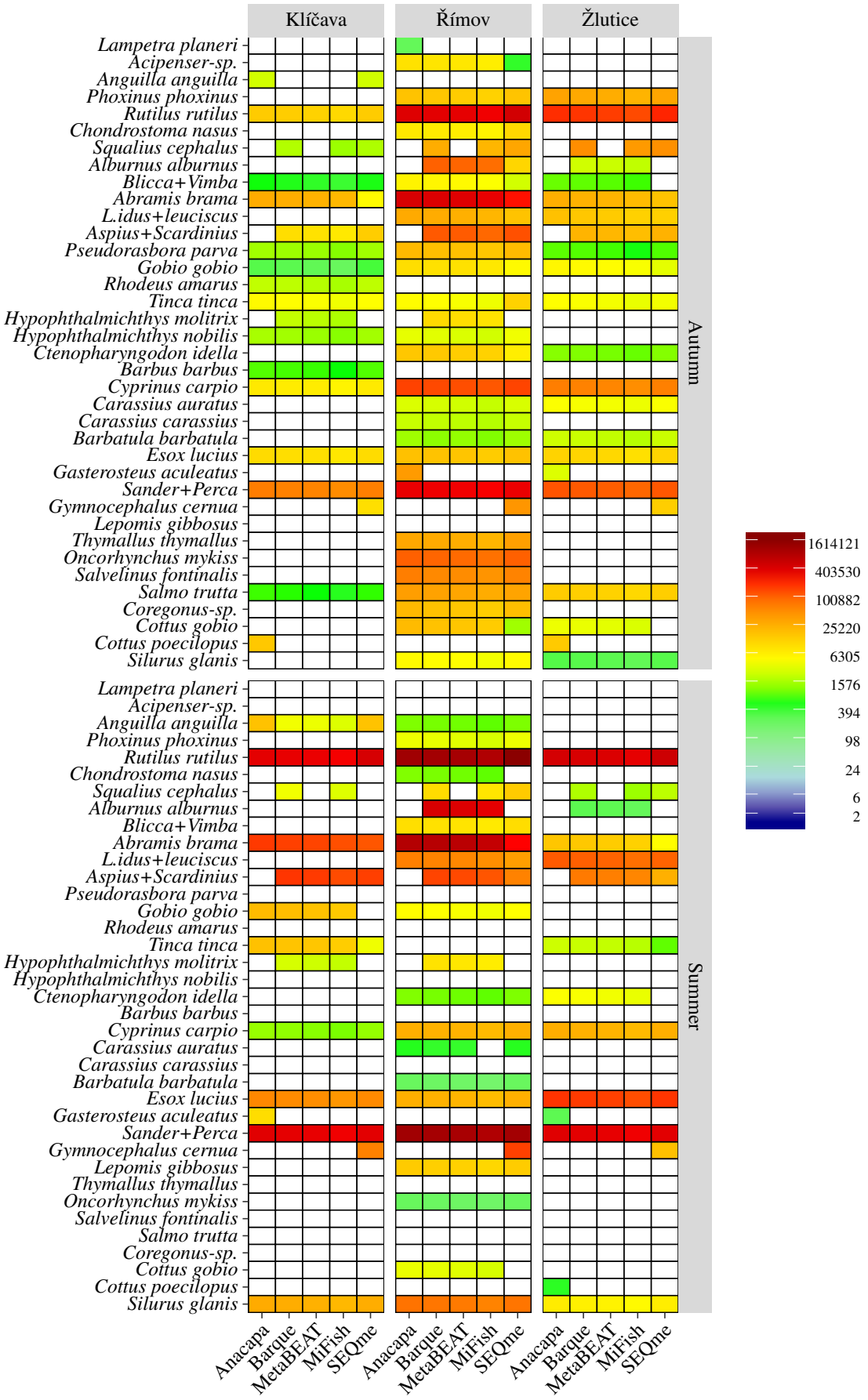

Figure 4.5: Number of reads assigned to species considering pipelines, reservoirs, and seasons, excluding positive and negative controls.

4.2.3 Alpha diversity

4.2.3.1 Species richness

The alpha diversity species richness describes the number of species in a single group (a group could be a pipeline, reservoir, or season). Considering pipelines, reservoirs, and seasons together, the alpha diversity ranged from 10 to 29 (Figure 4.6). The smallest richness was detected in the Anacapa and SEQme pipelines in the Klíčava reservoir both in the summer season, whereas the largest richness was detected in the Barque, MiFish, and SEQme pipelines in the Římov reservoir in the autumn season.

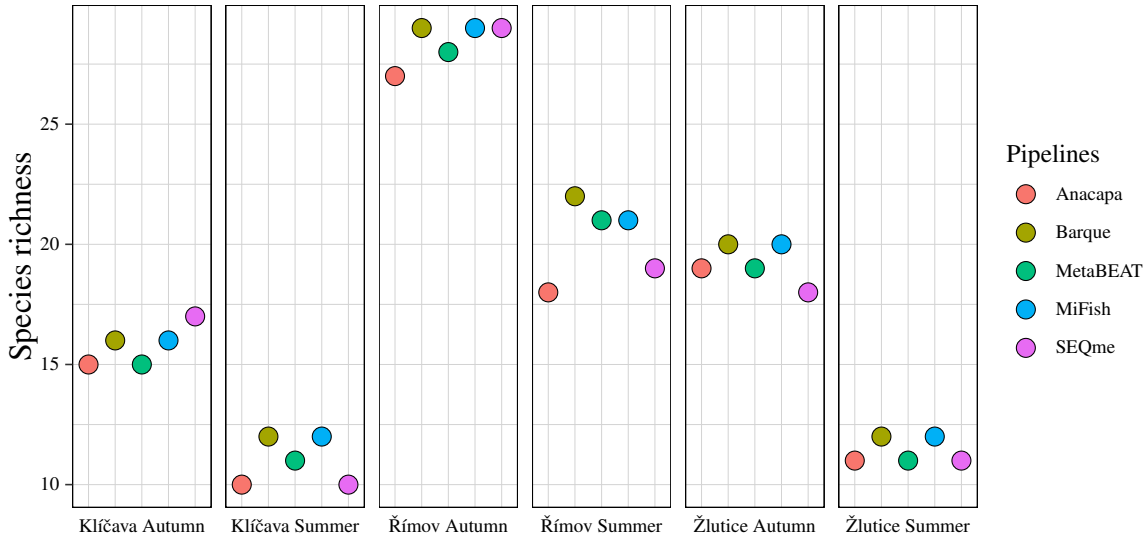

Figure 4.6: Alpha diversity species richness considering pipelines, reservoirs, and seasons, excluding positive and negative controls.

For each pipeline, the alpha diversity species richness was determined. In Anacapa it ranged from 10 to 27; in Barque and in MiFish it ranged from 12 to 29; in metaBEAT it ranged from 11 to 28; and in SEQme it ranged from 10 to 29. In the Anacapa and SEQme pipelines the smallest was observed only in Klíčava, whereas in Barque, metaBEAT, and MiFish the smallest richness was observed in Klíčava and Žlutice, all in the summer season. In contrast, the largest was observed in Římov in autumn (Figure 4.7). The number of species between the pipelines were similar (ANOVA: $F_{4,25} = 0.080$, $p = 0.988$).

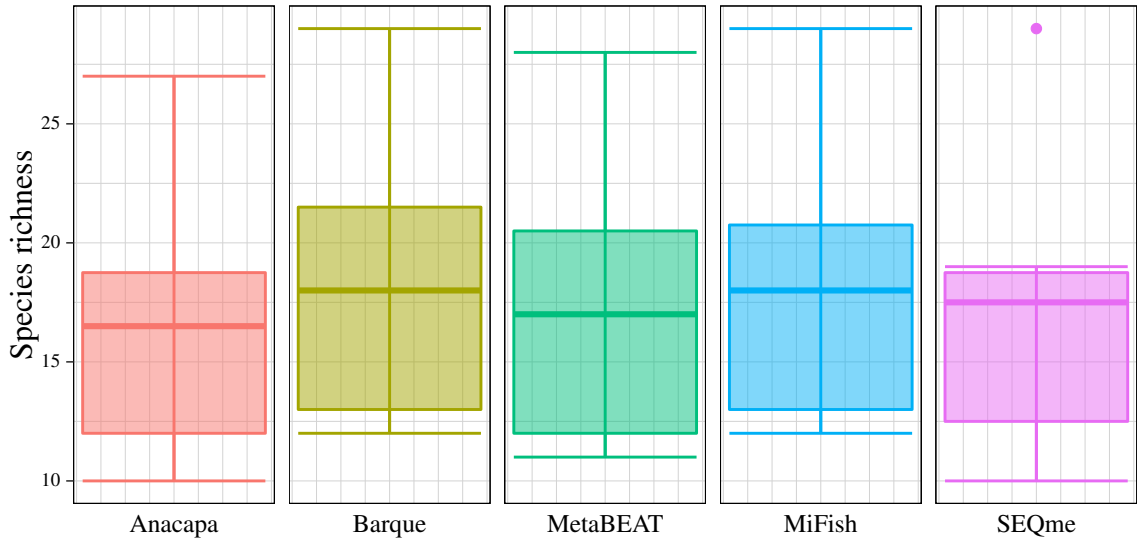

Figure 4.7: Alpha diversity species richness considering pipelines, excluding positive and negative controls. Median values (thick lines), upper and lower quartiles (boxes), maximum and minimum values (whiskers) and outlier (full circle) are shown.

Regarding reservoirs, in Klíčava it ranged from 10 to 17; in Římov it ranged from 18 to 29; and in Žlutice it ranged from 11 to 20. The smallest was detected in Anacapa and SEQme in Klíčava; in Anacapa in Římov; and in Anacapa, metaBEAT, and SEQme in Žlutice; all in the summer season. The largest was detected in SEQme in Klíčava; in Barque, MiFish, and SEQme in Římov; and in Barque and MiFish in Žlutice; all in the autumn season (Figure 4.8).

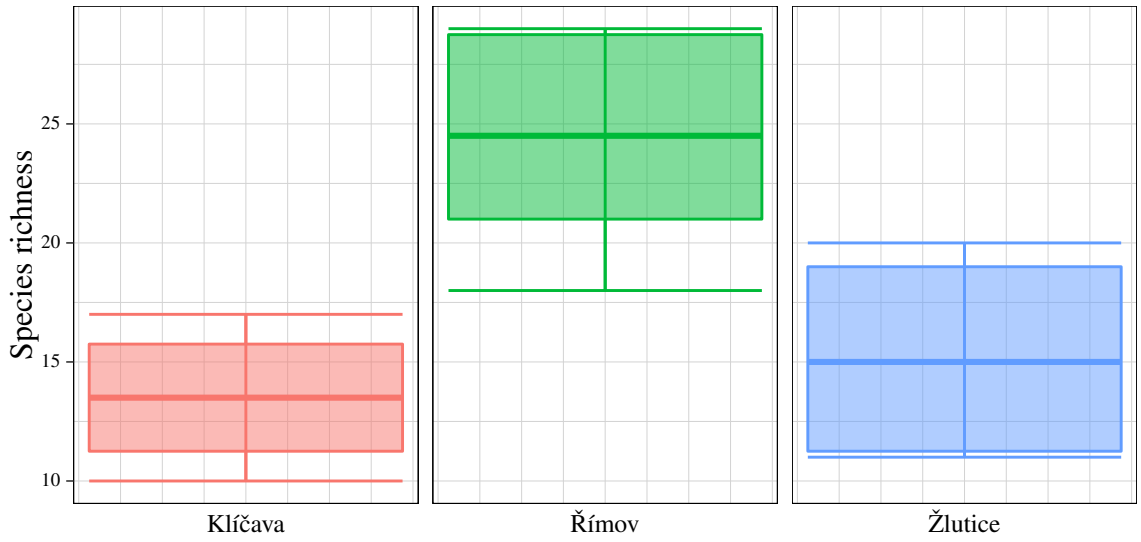

Figure 4.8: Alpha diversity species richness considering reservoirs, excluding positive and negative controls. Median values (thick lines), upper and lower quartiles (boxes), maximum and minimum values (whiskers) are shown.

A statistical test showed a statistically significant difference between the richness of the reservoirs (ANOVA: $F_{2,27} = 22.737$, $p < 0.05$). In addition, a post-hoc Tukey showed a statistically significant difference between Římov and Klíčava (p-value adjusted < 0.05), and

between Římov and Žlutice (p-value adjusted < 0.05), whereas it showed a similarity between Klíčava and Žlutice (p-value adjusted = 0.522).

For the seasons, it ranged from 15 to 29 in autumn and it ranged from 10 to 22 in summer. In autumn, the smallest richness was detected in Anacapa and metaBEAT in Klíčava, whereas in summer the smallest was observed in the Anacapa and SEQme pipelines in the Klíčava reservoir. The largest richness was detected in Barque, MiFish, and SEQme in Římov in Autumn, whereas in Barque in Římov in the summer season (Figure 4.9). A statistical test between the seasons showed a statistically significant difference between the richness of autumn and summer (ANOVA: $F_{1,28} = 14.018$, $p < 0.05$).

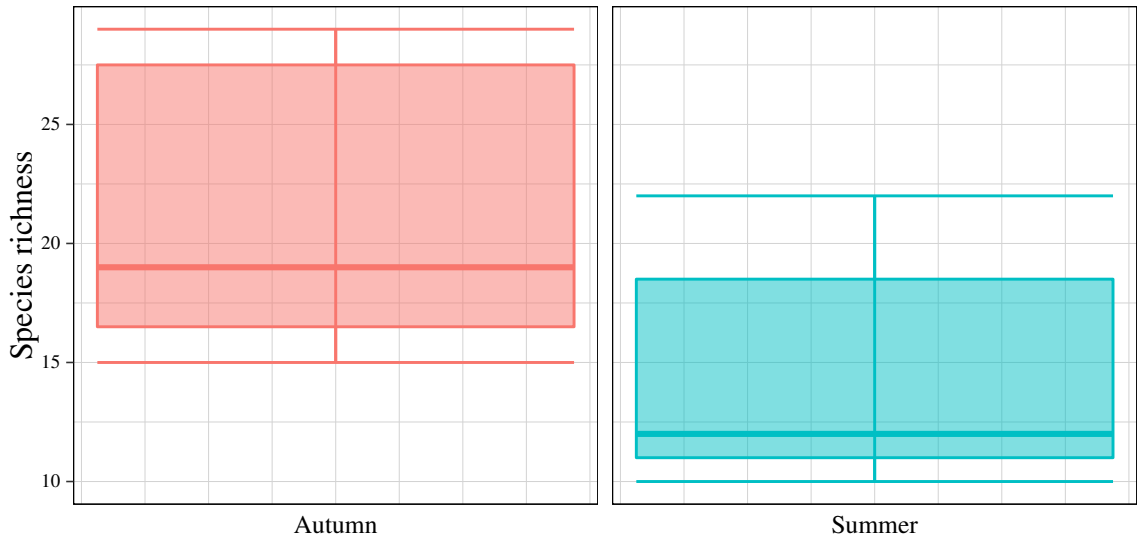

Figure 4.9: Alpha diversity species richness considering seasons, excluding positive and negative controls. Median values (thick lines), upper and lower quartiles (boxes), maximum and minimum values (whiskers) are shown.

4.2.3.2 Shannon index

In addition to the richness, the alpha diversity Shannon index also take into consideration if the number of sequence reads assigned is evenly distributed among all species (evenness). The higher the richness and the evenness, the higher the index. For a richness of 37, which is the total number of species (richness) detected in all pipelines in all reservoirs in the summer and autumn seasons, the diversity calculation would result a shannon index of 3.61 if all species had the number of reads equally distributed (evenness). Considering pipelines, reservoirs, and seasons together, the Shannon indices ranged from 1.395 to 2.425 (Figure 4.10). The smallest shannon index was detected in Anacapa in the Římov reservoir in the summer season, whereas the largest was detected in the MiFish pipeline in the Římov reservoir in the autumn season.

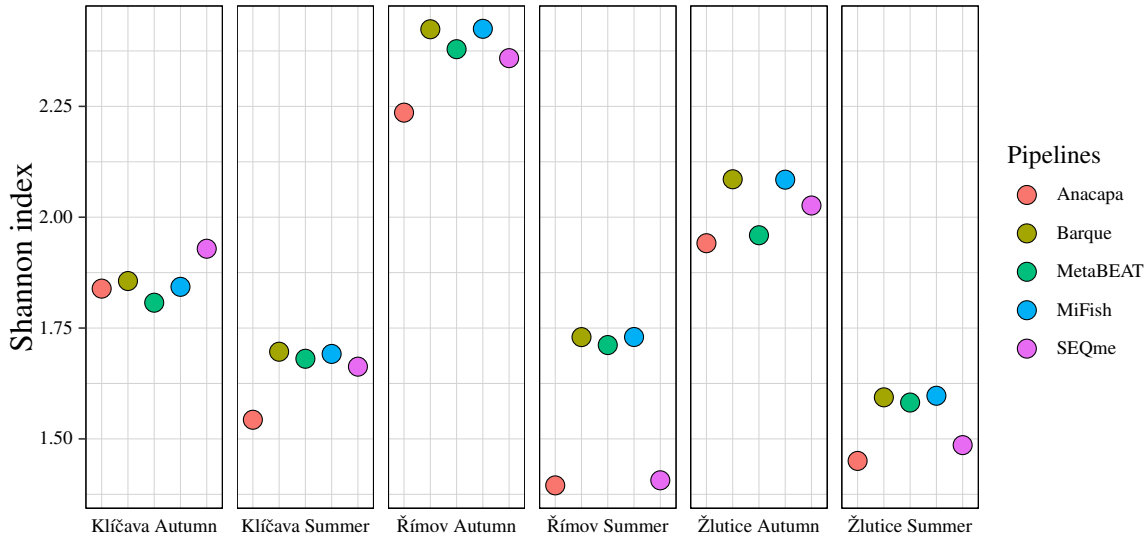

Figure 4.10: Alpha diversity shannon index considering pipelines, reservoirs, and seasons, excluding positive and negative controls.

For each pipeline, the shannon index was determined. In Anacapa it ranged from 1.395 (in Římov in summer) to 2.236 (in Římov in autumn); in Barque it ranged from 1.594 (in Žlutice in summer) to 2.424 (in Římov in autumn); in metaBEAT it ranged from 1.582 (in Žlutice in summer) to 2.379 (in Římov in autumn); in MiFish it ranged from 1.597 (in Žlutice in summer) to 2.425 (in Římov in autumn); in SEQme it ranged from 1.407 (in Římov in summer) to 2.359 (in Římov in autumn) (Figure 4.11). The Shannon indices between the pipelines were similar (ANOVA: $F_{4,25} = 0.272$, $p = 0.893$).

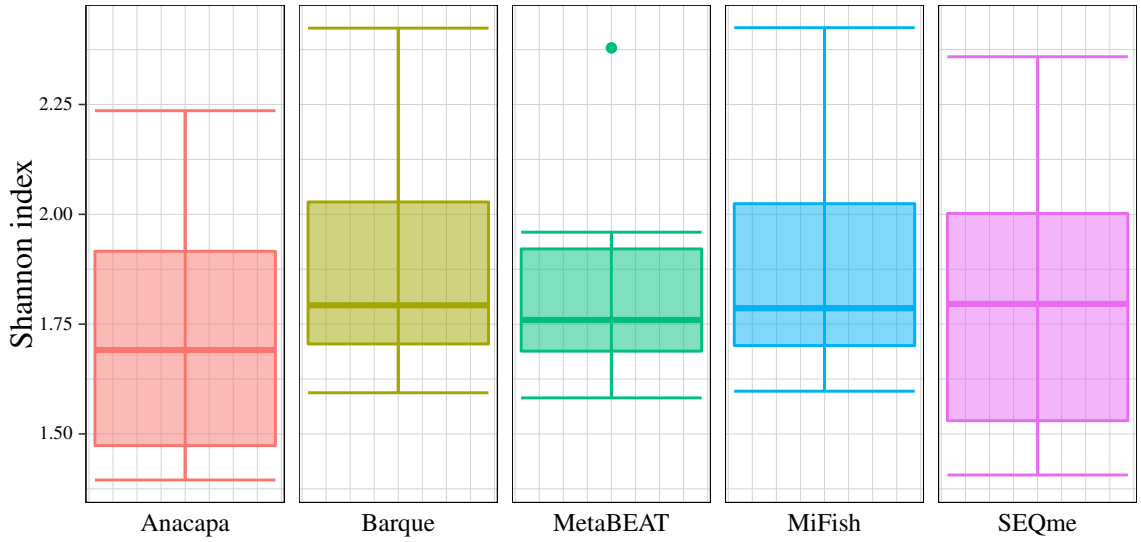

Figure 4.11: Alpha diversity shannon index considering pipelines, excluding positive and negative controls. Median values (thick lines), upper and lower quartiles (boxes), maximum and minimum values (whiskers) and outlier (full circle) are shown.

Regarding reservoirs, in Klíčava it ranged from 1.543 to 1.929; in Římov it ranged from 1.395 to 2.425; and in Žlutice it ranged from 1.45 to 2.086. The smallest shannon index was

detected in Anacapa in the summer season in all reservoirs. The largest was detected in SEQme in Klíčava; in MiFish in Římov; and in Barque in Žlutice; all in the autumn season (Figure 4.12). The Shannon indices between the reservoirs were not statistically significant different (ANOVA: $F_{2,27} = 1.726$, $p = 0.197$).

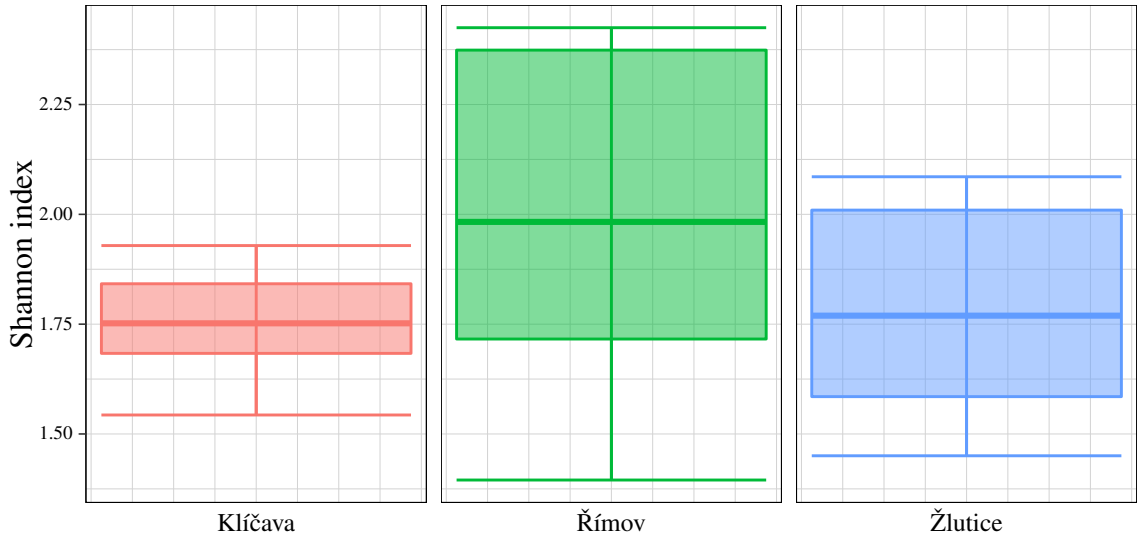

Figure 4.12: Alpha diversity shannon index considering reservoirs, excluding positive and negative controls. Median values (thick lines), upper and lower quartiles (boxes), maximum and minimum values (whiskers) are shown.

For each season, in autumn it ranged from 1.807 to 2.425 and in summer it ranged from 1.395 to 1.73. In autumn the smallest shannon index was detected in metaBEAT in Klíčava, whereas the largest was detected in MiFish in Římov. In summer the smallest shannon index was detected in Anacapa in Římov, whereas the largest was detected in MiFish in Římov (Figure 4.13). A statistical test between the seasons showed a statistically significant difference between the shannon indices of autumn and summer (ANOVA: $F_{1,28} = 53.149$, $p < 0.05$).

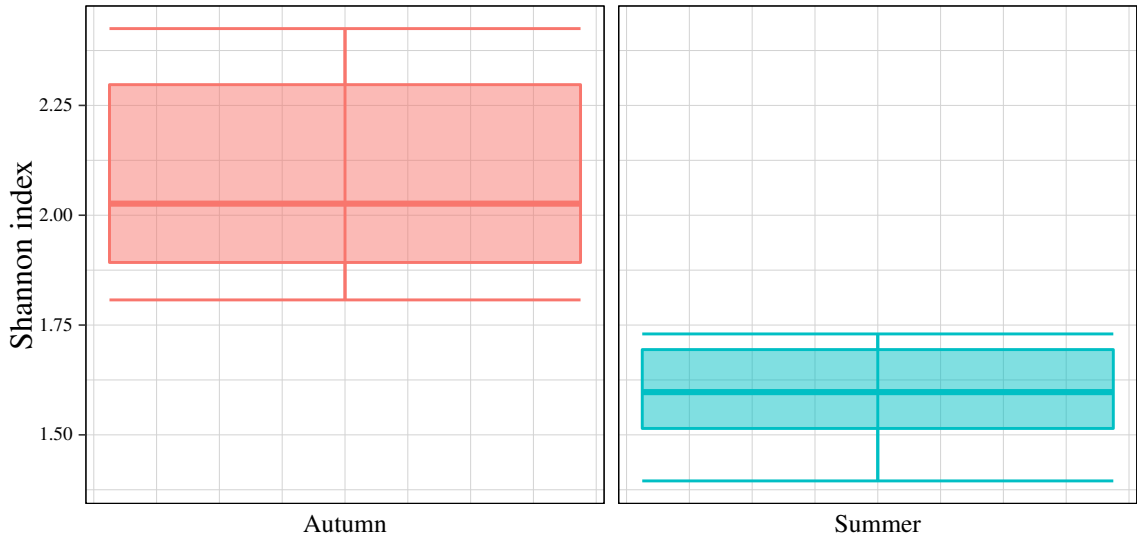

Figure 4.13: Alpha diversity shannon index considering seasons, excluding positive and negative controls. Median values (thick lines), upper and lower quartiles (boxes), maximum and minimum values (whiskers) are shown.

4.2.4 Beta diversity

4.2.4.1 Jaccard index

The beta diversity compares the species composition (diversity) among different groups, as two groups can have identical alpha diversity (richness), but different species composition. The Jaccard index only considers presence or absence of the species to measure the dissimilarity among groups without considering the number of sequence reads assigned (relative abundance). The index range from 0 % for identical composition to 100 % for completely different composition. Considering pipelines, reservoirs, and seasons together, the Jaccard dissimilarity indices ranged from 0.053 to 0.971. The highest similarity (dissimilarity of 0.053) was detected between Barque and metaBEAT pipelines both in Římov in the summer season. In contrast, the smallest similarity (dissimilarity of 0.971) was detected between Barque in Římov in summer and MiFish in Klíčava in autumn (Figure 4.14).

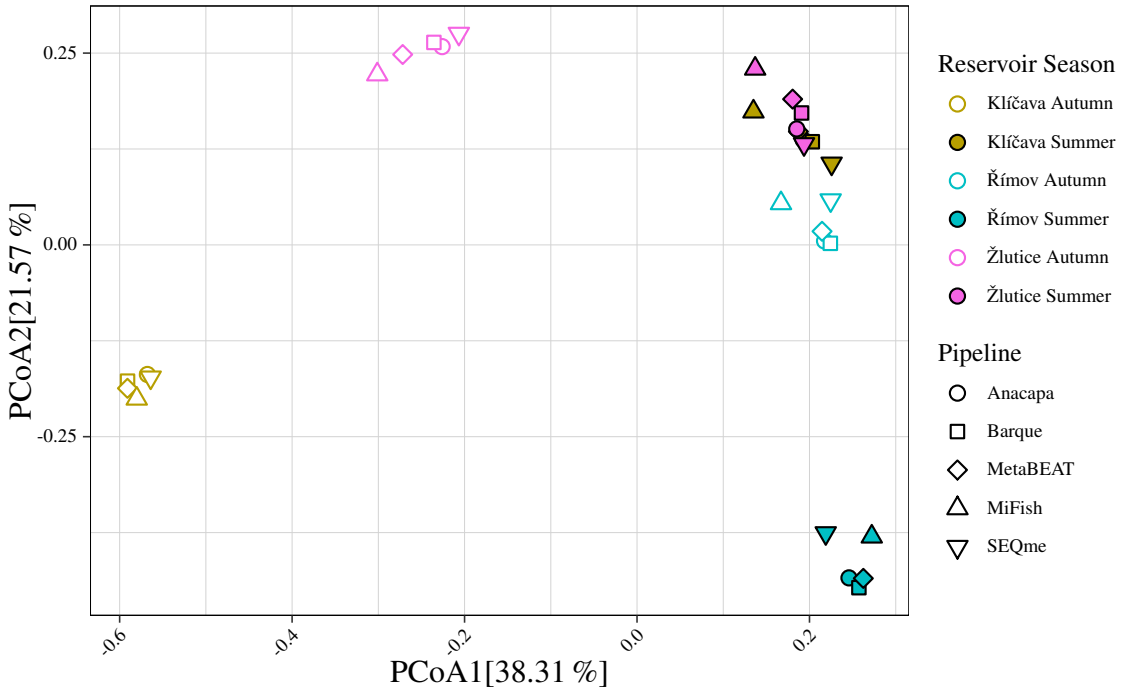

Figure 4.14: Beta diversity Jaccard index considering pipelines, reservoirs, and seasons, excluding positive and negative controls. The X axis indicates a variance of 38.31 % in the data observed in the X direction, whereas the Y axis represents a variance of 21.57 % in the Y direction.

Regarding pipelines, the strongest similarity was detected between Barque and metaBEAT (dissimilarity of 0.065) and the weakest was detected between MiFish and SEQme (dissimilarity of 0.373) (Table 4.8). For reservoirs, 0.507 was the dissimilarity distance between Klíčava and Žlutice, 0.77 between Klíčava and Římov, and 0.725 between Římov and Žlutice. Finally, 0.706 was the dissimilarity between autumn and summer.

	Anacapa	**Barque**	**MetaBEAT**	**MiFish**
Barque	0.20			
MetaBEAT	0.23	0.07		
MiFish	0.33	0.19	0.15	
SEQme	0.29	0.30	0.33	0.37

Table 4.8: Beta diversity Jaccard dissimilarity indices considering pipelines.

A test showed a statistically significant similarity between the pipelines (PERMANOVA: $F_{4,25} = 0.277$, $R^2 = 0.042$, $p = 0.998$). Regarding Reservoirs, the test showed a statistically significant difference (PERMANOVA: $F_{2,27} = 7.365$, $R^2 = 0.353$, $p < 0.05$). When considering each pair, a pairwise test showed a statistically significant difference between all pairs of reservoirs (pairwise PERMANOVA: $p < 0.05$). For seasons, a statistically significant difference was showed between autumn and summer (PERMANOVA: $F_{1,28} = 8.630$, $R^2 = 0.236$, $p < 0.05$).

4.2.5 Species detection consistency and inconsistency

From a total of 37 species detected, 4 species were detected in only one pipeline. Out of the 4, three were detected in Anacapa and one in SEQme. In contrast, 29 species were detected in all pipelines. Regarding reservoirs, 11 species were detected in only one reservoir, two in Klíčava and 9 in Římov. In contrast, 15 were detected in all reservoirs. Regarding seasons, 12 species were detected in only one season, with 11 in autumn and only one (*Lepomis gibbosus*) in the summer season. In contrast, 25 were detected in both seasons.

When considering pipelines, reservoirs, and seasons together, only one species was detected just once (*Lampetra planeri*), in Anacapa in Římov in the autumn season. In contrast, 5 species were detected in all pipelines, reservoirs, and seasons (*Abramis brama*, *Cyprinus carpio*, *Esox lucius*, *Rutilus rutilus*, and *Sander+Perca*) (Figure 4.5).

4.2.6 Positive and negative controls detection

Regarding positive and negative controls, after removing for each sample any potential false readings of the total of reads demultiplexed where the number of reads assigned were smaller than a threshold of 0.1 %, only reads from the positive controls were assigned to species. Sequences from the postive controls were only assigned to *Maylandia zebra* (species used as control) in all pipelines.

Based on the number of reads after demultiplexing as total, the percentage of assigned reads to the positive control ranged from 70.11 % to 96.76 %, with 91.2 % for the median and 87.20 % for the mean (Figure 4.15).

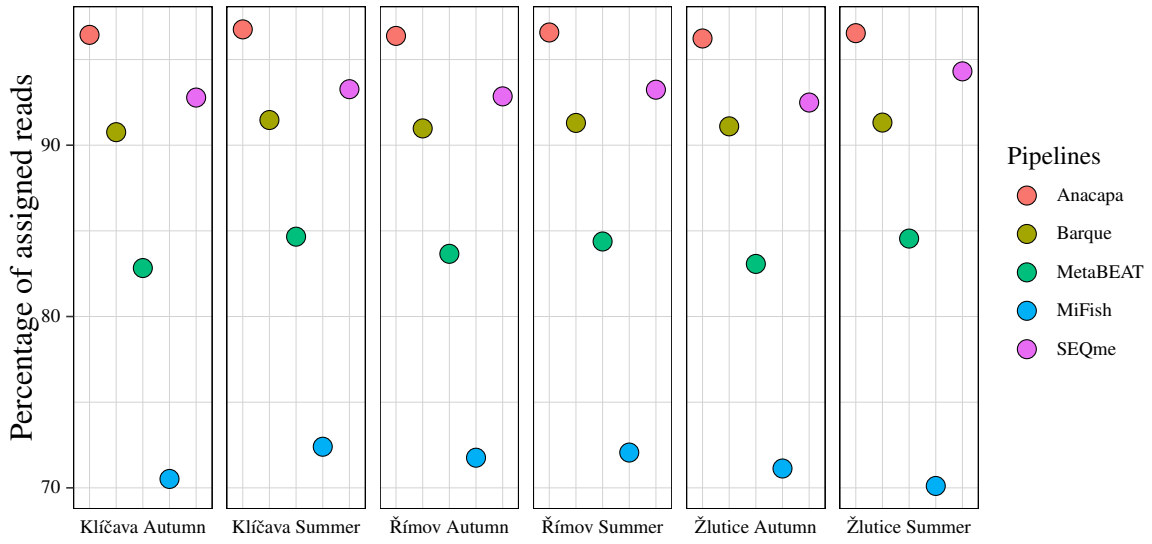

Figure 4.15: Percentage of assigned reads to the positive control Maylandia zebra based on the initial total of reads (demultiplexed reads) used as the input data.

The smallest percentage was detected in the MiFish pipeline in the Žlutice reservoir in the summer season, whereas the highest percentage was detected in the Anacapa pipeline in Klíčava

in summer. A statistical test showed a statistically significant difference in the percentages of positive control detection among the pipelines (ANOVA: $F_{4,25} = 1549.3$, $p < 0.05$). In addition, a post-hoc Tukey showed a strong statistically significant difference in the percentages between each pair of pipelines (Tukey: $p < 0.05$).

For each pipeline, in Anacapa the percentage of assigned reads to the positive control ranged from 96.23 % to 96.76 %; in Barque it ranged from 90.76 % to 91.47 %; in metaBEAT it ranged from 82.83 % to 84.66 %; in MiFish it ranged from 70.11 % to 72.4 %; in SEQme it ranged from 92.49 % to 94.31 %. All pipelines showed no variance greater than 5.

4.3 Execution time of the pipelines

The execution time for each pipeline was determined, seconds were not considered. In the Anacapa pipeline the time of execution was 2 hours and 59 minutes. In Barque it was 21 minutes. In metaBEAT the time was 12 hours and 45 minutes. The MiFish pipeline run time was 1 hour and 51 minutes. Finally, the execution time of SEQme pipeline was 23 minutes.

Discussion

This study is the first comparison of the entire workflow of five distinct eDNA metabarcoding pipelines. The number of reads after each step in the pipelines execution, the total number of reads assigned, the number of species detected, the number of sequence reads assigned to species, and community indices were compared. High similarities and consistent statistical results were found despite different approaches. The main differences were in MiFish considering the number of reads assigned and in Acanapa considering the number of true positive species detected. The alpha and beta diversity demonstrated very similar results among the pipelines. The results from the pipelines were also compared to the fish community composition detected by conventional methods. Metabarcoding demonstrated to be very efficient in species detection performing better than conventional methods when considering the number of species detected. All species observed in conventional methods were detected in at least one of the pipelines (*Lota lota* was detected in SEQme, but discarded after removing sequences with frequencies below 0.1 % threshold). Finally, autumn outperformed summer in the number of species detected, but summer had a higher number of reads assigned in total.

5.1 Comparison of pipelines and conventional methods species detection

The outputs of the pipelines were highly similar in both number of species detected and species composition. Anacapa resulted the most dissimilar detections among the pipelines. Three species were only detected in Anacapa, *Lampetra planeri*, *Gasterosteus aculeatus*, and *Cottus poecilopus*. With exception of Barque, before discarding detections where the number of reads assigned were smaller than 0.1 % of the total of reads in the sample [Hänfling et al., 2016; Lawson Handley et al., 2019], *Lampetra planeri* was present in all pipelines in Římov in autumn. However, after applying the threshold only the Anacapa pipeline preserved the species. *Lampetra planeri* was also detected in conventional methods in Římov, which corroborate the species as a true positive. On the other hand, *Gasterosteus aculeatus* is probably a false positive as the species is only present in the east region of the Czech Republic and in the northern district near Liberec [IUCN, 2018]. *Cottus poecilopus* is probably a misinterpretation of *Cottus gobio* as *Cottus poecilopus*. Five species were not detected in Anacapa, but detected in at least one of

the pipelines. *Alburnus alburnus* and *Aspius+Scardinius* are common species in the reservoirs, but not detected in the pipeline. *Squalius cephalus* was not detected in Anacapa, but it was also not present in metaBEAT. Equivalent behaviour happened for *Hypophthalmichthys molitrix*, it was not detected in Anacapa and SEQme. The Anacapa pipeline uses DADA2 [Callahan et al., 2016] to infer Amplicon Sequence Variants (ASV) instead of Operational taxonomic unit (OTU) clustering. DADA2 ASV inference was demonstrated to be less sensitive in species detection than clustering algorithms (USEARCH [Robert C. Edgar, 2010] and VSEARCH [Rognes et al., 2016]), the latter should be preferred when sensitivity on species detection is the goal [Pauvert et al., 2019]. The BLCA classifier [Gao et al., 2017] applied after DADA2 was set to 60 % of confidence to diminish the number of false negatives. However, the precision given in exchange of sensitivity increases the possibility of false positives [Pauvert et al., 2019]. The combination of a less sensitive algorithm and a lower confidence score resulted on false positives and negatives. The confidence could be reduced to attempt the detection of species not detected.

Barque and MiFish pipelines were identical in number and composition of the species detected. One distinction was found when the detections in different reservoirs and seasons were considered as *Carassius auratus* was removed from MiFish in summer in Římov after applying the threshold (0.1 %) to remove species with low number of detections. Barque and MiFish have in common the alignment-based approach used for the taxonomic assignment and they both ignore the creation of Operational Taxonomic Unit (OTUs) or Amplicon Sequence Variants (ASV). The intermediate steps did not influence in the species composition detected, but only in the number of reads assigned. MetaBEAT pipeline had almost identical species composition as Barque, the only difference is the nondetection of *Squalius cephalus*. The species detected in each reservoir and season were also equal in both pipelines. With exception of the taxonomic assignment, which is done using BLAST [Camacho et al., 2009], the metaBEAT pipeline has an identical selection of programs as Barque, with additional python scripts and slightly different parameters used in both pipelines.

For the SEQme pipeline, the species composition detected was also analogous to Barque and Mifish detections, with the addition of *Gymnocephalus cernua* detected in SEQme and the reduction of *Hypophthalmichthys molitrix* not detected in the pipeline. SEQme was the unique pipeline to detected *Lota lota*, but the species was removed when applying the 0.1 % threshold. *Lota lota* was detected by conventional methods within the last 3 campaigns (2018, 2019, and 2020) in Římov and Žlutice, but each time only one specimen was detected. The eDNA of rare species, because of the small number of specimen, is found in low concentration in the environment [Sepulveda et al., 2019]. In addition, *Lota lota* has nocturnal habits and prefer cold temperatures [Eick, 2013; Blabolil et al., 2018]. Therefore, sampling considering these characteristics would increase the possibility of detection. Besides SEQme, *Gymnocephalus cernua* was also detected in Anacapa and metaBEAT, but it was likewise removed when

applying the threshold due to the low number reads assigned to this species. Hybrids resulted of artificial spawning of *Hypophthalmichthys molitrix* and *Hypophthalmichthys nobilis* are frequent [Nosova, Kipen, Tsar, & Lemesh, 2020], which may cause misinterpretation. SEQme pipeline rely on RDP classifier [Wang et al., 2007], a naïve Bayesian machine learning approach, for the taxonomic classification. Machine learning was demonstrated to outperform alignment-based approaches with higher species-level accuracy and lower number of false-positives [O'Rourke, Bokulich, Jusino, MacManes, & Foster, 2020; Bokulich et al., 2018]. Although having similar species composition, the same similarity was not present in the number of reads assigned to species. The amount detected in SEQme for *Alburnus alburnus* was lower when compared to Barque, metaBEAT, and MiFish, as the species was only detected in Římov in the autumn season.

Seven species not observed by conventional methods in the last three years (2018, 2019, and 2020) were detected in all pipelines (*Barbus barbus*, *Carassius carassius*, *Cottus gobio*, *Phoxinus phoxinus*, *Rhodeus amarus*, *Salvelinus fontinalis*, and *Thymallus thymallus*). *Barbus barbus* and *Carassius carassius* were already caught by conventional methods in the past in 2007 in Klíčava and 2003 in Římov, respectively. These two species were detected in the pipelines in the same reservoirs (Klíčava and Římov) at very low reads proportions, less than 0.1 % of the total of reads assigned. *Barbus barbus* was detected in two sampling locality, whereas *Carassius carassius* was detected in only one. The other five unrecorded species were detected in higher proportion of reads assigned. *Cottus gobio* and *Phoxinus phoxinus* are present in habitats difficult to be captured by conventional methods, *Rhodeus amarus* has a small size hard to be observed, *Salvelinus fontinalis* is dependent on restocking, and *Thymallus thymallus* is a rare species with preference for running currents [Blabolil et al., 2020]. In addition, eDNA could have been washed from the catchment and new species are introduced with stocking as contamination of predatory species stocked every year.

Only *Maylandia zebra* was detected in PCR positive controls. In the negative controls, there was no detection of any species, which indicates no contamination during the process [Taberlet, Bonin, Zinger, & Coissac, 2018a]. Anacapa pipeline, with 96.49 on average, had the largest number of reads assigned to *Maylandia zebra* positive control, whereas MiFish, with 71.33 on average, had the lowest. Although having the highest number of possibles false negatives detected, the percentage of positive control detected in Anacapa demonstrates a consistency in the number reads assigned to species. On the other hand, MiFish had the lowest number reads assigned in general, which was also reflected on positive control detections.

5.2 Alpha and beta diversities comparison

The alpha and beta diversities were statistically similar among the pipelines. The Alpha diversity was calculated to evaluate the similarity in the number of species identified and the beta

diveristy was calculated to check the species composition among the pipelines, as identical alpha diversities can have completely different species compositions [Whittaker, 1972]. The number of species detected in the pipelines (32 in Anacapa and metaBEAT, and 33 in Barque, MiFish, and SEQme) was similar to the number of species observed by conventional methods in the last three years (2018, 2019, and 2020) in all reservoirs (29 when considering *L.idus+leuciscus*, *Aspius+Scardinius*, and *Sander+Perca* joined and 32 without considering the joining). The beta diversity species composition was statistically similar among the pipelines and traditional methods. However, although having similar species compositions, seven species were not identified by conventional methods, but detected in all pipelines. In addition, when pipelines are considered together, all species detected by conventional methods were also detected in the pipelines (*Lota lota* was detected in SEQme, but removed after applying threshold). A few species were missed in each one when considering the pipelines individually. Environmental DNA (eDNA) metabarcoding was demonstrated to be highly sensitive in detecting common and rare species [Valentini et al., 2016; Hänfling et al., 2016]. A threshold (0.1 %) to remove possible false positive detections discarded potential true positives, a smaller percentage for the threshold could be considered.

The number of reads assigned was also considered to evaluate the alpha diversity using Shannon index calculation [Shannon, 1948]. The Shannon indices among the pipelines were statistically similar. Pipelines applying alignment-based approach for taxonomic assignment, Barque, metaBEAT, and MiFish, were comparable in number of reads assigned to each species. With exception of *Squalius cephalus* not detected in metaBEAT, the detection followed a pattern, with the highest, middle, and lowest assigned to Barque, metaBEAT and MiFish, respectively. Pipelines that apply a Bayesian classifier for taxonomic assignment, Anacapa and SEQme, also presented a pattern in the detection between the pipelines. For example, *Alburnus alburnus* species not detected in Anacapa, similarly in SEQme the detection was much lower than alignment-based pipelines.

The number of species observed in each reservoir by the pipelines were higher than conventional methods. Římov had the largest (34), folowed by Žlutice (23), and Klíčava had the lowest (21). The fish community composition is dependent on lake morphology and trophic state, the higher is the area, volume, and trophic state the higher is the fish population [Mehner, Diekmann, Brämick, & Lemcke, 2005; Willemsen, 1980]. The number of species detected in each reservoir confirms the pattern. Římov the largest reservoir, with the largest volume, and eutrophic trophic state detected the highest number of species. Žlutice, which had the second highest number of species observed, is the second in area and volume, and also have an eutrophic state. Finally, the smallest reservoir with an oligotrophic state, Klíčava detected the lowest number of species. Environmental DNA has a decay rate faster in oligotrophic than eutrophic [Eichmiller, Best, & Sorensen, 2016], which could contribute to a lower detection in Klíčava. The number of reads assigned in each reservoir also followed the same arrangement,

the highest to Římov, Žlutice in the middle, and the lowest to Klíčava.

Pipelines detected all species obeserved by conventional methods in Římov. In Žlutice two species (*Hypophthalmichthys molitrix* and *Anguilla anguilla*) were not detected in the pipelines but observed using traditional methods. In Klíčava three (*Alburnus alburnus*, *L.idus+leuciscus*, and *Carassius auratus*) were not detected. *Hypophthalmichthys molitrix* and *Anguilla anguilla* were detected in small number in Žlutice by conventional methods. Because of the small number, the eDNA is found in low concentration [Sepulveda et al., 2019], which increases the possibility of being missed in the sampling and molecular processing [Kelly, Shelton, & Gallego, 2019]. The same for *Alburnus alburnus*, *L.idus+leuciscus*, and *Carassius auratus*, which were detected in small quantity by conventional methods in Klíčava. The number of species not detected in each reservoir by conventional methods but detected for the first time by the pipelines was 5 in Klíčava, 8 in Římov, and 6 in Žlutice (*Gasterosteus aculeatus* and *Cottus poecilopus* detected only in Anacapa pipeline were not considered). A higher number of species detected in the pipelines supports the higher sensitivity in species detection by eDNA metabarcoding than conventional methods [Valentini et al., 2016; Hänfling et al., 2016].

Regarding seasons, 36 out of 37 species detected in total were observed in the autumn season and 26 in summer. Only *Lepomis gibbosus* was not detected in autumn, but in the summer season. Fishes are ectothermic (cold-blooded) animals dependent on the environment to regulate their body temperature [van de Pol, Flik, & Gorissen, 2017]. They are very sensitive to changes in temperature and can sense even really small variations [Bardach & Bjorklund, 1957]. Each species has a temperature range preference, which can have an influence in the physiology, bioenergetic, and behaviour of the species [Leuven et al., 2011]. The eDNA detection probability is influenced by the preferred temperature range, which says when the species is more active, the higher the activity the higher the detection, therefore the season of sampling directly affects the detection probability of the environmental DNA [de Souza, Godwin, Renshaw, & Larson, 2016]. The detection of *Lepomis gibbosus* only in the summer season corroborates the preference of the species for higher water temperature and being more active in the warm season [Blabolil et al., 2020]. Higher number of species detected in autumn can be explained by the influence that the temperature and UV radiation have on environmental DNA (eDNA) degradation. The eDNA in colder temperatures persist for longer time as demonstrated by Strickler et al. [2015]. The degradation is caused by either direct action of temperature or indirect by exonuclease and microbial activity. In addition, it was also demonstrated that UV radiation can direct affect the degradation or indirect influence the impact of the temperature. With the exception of *Carassius carassius*, which can tolerate temperatures ranging from 2 °C to 38 °C, most of the species (lamprey, sturgeon, trout, whitefish, *etc*) detected only in the autumn season can not tolerate high temperatures and have preference for cold water [Leuven et al., 2011]. Finally, autumn has a higher discharge resulting in higher eDNA washing from upstream [Blabolil et al., 2020].

Although showing higher number of species detected in the autumn season, both the total number of reads and the number of reads assigned was higher in the summer season. The number of reads assigned to species in the summer season was more than twice the number assigned in autumn. The reservoirs are dominated by cyprinid species, which in general prefer warm water and are more active in the summer season [Cherry, Dickson, Cairns Jr., & Stauffer, 1977; Cherry & Cairns, 1982]. The number of reads in the summer season was inflated by a huge detection of abundant common cyprinid species in the reservoirs.

5.3 Pipelines analogy and recommendations

In the present study, all pipelines were statistically high similar in the alpha and beta diversities. They were also consistent in terms of the number of reads assigned and the number of species detected. The choice on which one should be used for eDNA metabarcoding data analysis is driven by the goal of the project. A study when the aim is to detect the number of different species in a study site could even have the step of OTU or ASV creation ignored as demonstrated by Barque result. A complete reference database including all species possibly to be present in the study site must be created, otherwise the lack of sequences could result in potential false negatives [Schenekar, Schletterer, Lecaudey, & Weiss, 2020]. On the other hand, studies not requiring taxonomic identification, where the only need is the categorization of groups of related individuals based on sequence similarity, for example for microbial communities diversity where most of microbial diversity was not identified yet [Lladó Fernández, Větrovský, & Baldrian, 2019; Locey & Lennon, 2016], must apply the OTU or ASV creation step. Two approaches (OTU and ASV) were developed to group related sequences while minimizing errors caused by PCR and sequencing that could lead to false misassignment [Nearing, Douglas, Comeau, & Langille, 2018]. An operational taxonomic unit (OTU) is a cluster of similar sequences, often clustered with a threshold of 97 %, to avoid errors which could create slightly different sequences that could be interpreted as a separate taxonomic unit [Robert C. Edgar & Flyvbjerg, 2015; Huse, Welch, Morrison, & Sogin, 2010]. However, OTU tends to overestimate species richness by creating a number of OTUs higher than real [Barnes et al., 2020]. An amplicon sequence variants (ASV) applies an approach for correcting amplicon errors, which is more effective on creating the real composition of the community [Callahan et al., 2016; Pauvert et al., 2019]. Considering the two methods, when the sensitivity on species detection must be maximized an OTU approach must be considered, the number of species is overestimated with possible false positives generated, but a larger number of species is detected. On the other hand, ASV should be considered when composition of the community is the goal, but ignoring possible false negatives generated.

Trimming, merging and filtering demonstrated small variations in the number of sequences after each step, but the number for each pipeline are still statistically similar. Besides the

clustering step, the quality control curation also help to correct errors created during PCR and sequencing (e.g., low-quality, adapter contamination, small sequences), which could lead to false positive and negatives in taxonomic assignment, and has a significant impact on the final result [He et al., 2020]. Although the steps of quality control had an influence in the final number of reads assigned, there is still no consensus wether the read counts reflects the abundance and biomass, even though studies already pointed out the correlation between them [Lamb et al., 2019; Muri et al., 2020; Hänfling et al., 2016]. Regarding the taxonomic assignment, two different methods were used by the pipelines, alignment-based classification and Bayesian classifier. The number of species and species composition were statistically similar among the pipelines. However, machine learning Bayesian classifiers were more sensitive to detect rare species. For this reason, machine learning should be considered when rare species must be detected. Considering the execution time of the pipelines, Barque and SEQme were the fastest pipelines (in both pipelines the jobs are done in parallel), completing the execution in less than 23 minutes, which is approximately five times faster than MiFish, eight times faster than Anacapa, and more than 33 times faster than metaBEAT. However, Barque applies a alignment-based method for taxonomic assignment, which may be too slow if the number of sequences (in the database and reads) and the length of the sequences are huge [Wangensteen & Turon, 2017; Zielezinski, Vinga, Almeida, & Karlowski, 2017]. Therefore, SEQme is preferable when a vast volume of data must be analysed and time must be taking into consideration. Finally, there is no optimal pipeline, in the best scenario we should run more than one simultaneously to extract the quality of each one (i.e, species composition, number of reads assigned, and detection of rare species), and apply a post-processing to compare the results.

Conclusion

This study demonstrated high similarity in the results of five distinct pipelines concerning the number of species detected, number of reads assigned, and species composition. The similarity was corroborated by consistent statistical results. The data are essential for ecological studies, different pipelines having dissimilar results would lead to incorrect ecological assessment that could have very serious consequences, as the information is used for taking decisions on wildlife management. This research will help to speed up the validation of eDNA metabarcoding as a reliable method for biodiversity monitoring. As a consequence, traditional methods that use mostly invasive and destructive sampling will be used less often and save animals.

Environmental DNA (eDNA) metabarcoding has several advantages over conventional methods. First of all, it is a non-invasive technique, thus stress or death is not caused in contrast with conventional methods. In addition, the technique demonstrated to be more sensitive with a higher detection probability than conventional methods. Metabarcoding is also more efficient in detecting rare species and in recognizing species that are difficult to distinguish by even an experienced taxonomist [Elbrecht, Vamos, Meissner, Aroviita, & Leese, 2017; Harper et al., 2019]. However, eDNA metabarcoding has also disadvantages compared to conventional methods and problems that need to be solved in the technique. First of all, eDNA metabarcoding can not be used to collect several informations of a specimen, such as life stage, size, weight, sex ratio, hybrids, fish condition, or age structure. Environmental DNA metabarcoding also needs more research wether read counts could be correlated to abundance and biomass, but studies already found a possible correlation [Muri et al., 2020; Hänfling et al., 2016]. In addition, although being a cost-effective method, laboratory and sequencing costs are still high. Furthermore, primer bias, PCR inhibitors and errors introduced in PCR amplification and sequencing could cause a misinterpretation of the data, thus application of more primers is recommended. Finally, relativelly fast eDNA degradation in the environment (depending on the particularity of the environment) and in the sample processing could lead to a inaccurate species detection [Elbrecht et al., 2017; Harper et al., 2019].

Other problems can get in the way during a eDNA metababarcoding analysis. Species sequences are underrepresented in public databases or represented by low-quality sequences [Weigand et al., 2019]. In the present study, the DNA of a few species not present in public databases were sequenced and included in the reference library. Although the genetic marker

is chosen to have a distinction between all species in the study site, sometimes it can not distinguish related species. The current one used in the research can not distinguish perch and pikeperch, Coregonus and Acipensers, thus more genetic markers should be used. Other sequencing technology (e.g., Oxford Nanopore technology) could also be used to get longer reads and help distinguishing related species and increase the precision of the taxonomic identification [Santos, van Aerle, Barrientos, & Martinez-Urtaza, 2020; Doorenspleet, Jansen, Oosterbroek, & Nijland, 2021]. In addition, internal positive control (IPC) is recommended to deal with the PCR inhibition problem [Goldberg et al., 2016].

With the increase in the popularity of eDNA metabarcoding, new pipelines are being developed really fast. However, validation and comparison of the results between the pipelines is rare. For future perspectives, new pipelines can be included in the comparison, such as PEMA [Zafeiropoulos et al., 2020] and BIOCOM-PIPE [Djemiel et al., 2020]. In addition, pipelines are almost in totality command line tools without graphical interface. Pipelines could be more user-friendly and flexible in terms of tools to be chosen with either web or native GUI (graphical user interface) for both the metabarcoding bioinformatics workflow (e.g., SLIM [Dufresne, Lejzerowicz, Perret-Gentil, Pawlowski, & Cordier, 2019]) and post-processing statistical data analysis (e.g., TaxonTableTools [Macher, Beermann, & Leese, 2021], ranacapa [Kandlikar et al., 2018]). There is also a new trend toward using machine learning algorithms in eDNA metabarcoding [Cordier et al., 2018; Nugent & Adamowicz, 2020].

More information could be targeted in future eDNA research beyond biodiversity. Population genetics studies could be conducted using eDNA, such as analysis of haplotypes, deep genetic diversity, sex chromosome, interaction between species [Sigsgaard et al., 2020; Adams et al., 2019; Djurhuus et al., 2020]. The genetic marker used in the study is vertebrate-specific, thus the data could be used to analyse other vertebrate groups present in the reservoirs. Human DNA signal may help to find places with illegal activity (e.g., poachers, swimming people). Endangered, invasive, and rare species could be monitorated using eDNA metarcoding following the IUCN red list of threatened species (https://www.iucnredlist.org/). The same list may be used to check for endangered and invasive species that were missed in the pipelines. Even operational taxonomic unit (OTU) or amplicon sequence variant (ASV) could be sufficient indicators of diversity in samples without assigning a taxonomic rank to sequences. Finally, the comparison of the pipelines helped to validate eDNA metabarcoding as a reliable method. The outcomes of this study will be restructured and submitted in a scientific journal, thus supporting wildlife conservation and protecting our planet for future generations.

Bibliography

Adams, C. I., Knapp, M., Gemmell, N. J., Jeunen, G.-J., Bunce, M., Lamare, M. D., & Taylor, H. R. (2019). Beyond Biodiversity: Can Environmental DNA (eDNA) Cut It as a Population Genetics Tool? *Genes*, *10*(3). doi:10.3390/genes10030192

Andrews, S. et al. (2010). FastQC: a quality control tool for high throughput sequence data. Retrieved December 23, 2020, from https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Aronesty, E. (2013). Comparison of sequencing utility programs. *The open bioinformatics journal*, *7*(1).

Bardach, J. E. & Bjorklund, R. G. (1957). The Temperature Sensitivity of Some American Freshwater Fishes. *The American Naturalist*, *91*(859), 233–251. doi:10.1086/281982

Barnes, C. J., Rasmussen, L., Asplund, M., Knudsen, S. W., Clausen, M.-L., Agner, T., & Hansen, A. J. (2020). Comparing DADA2 and OTU clustering approaches in studying the bacterial communities of atopic dermatitis. *Journal of Medical Microbiology*, *69*(11), 1293–1302. doi:https://doi.org/10.1099/jmm.0.001256

Blabolil, P., Duras, J., Jůza, T., Kočvara, L., Matěna, J., Muška, M., … Peterka, J. (2018). Assessment of burbot *Lota lota* (L. 1758) population sustainability in central European reservoirs. *Journal of Fish Biology*, *92*(5), 1545–1559. doi:https://doi.org/10.1111/jfb.13610

Blabolil, P., Harper, L., Říčanová, Š., Sellers, G., Di Muri, C., Jůza, T., … Hänfling, B. (2020). Environmental DNA metabarcoding uncovers environmental correlates of fish communities in spatially heterogeneous freshwater habitats. doi:10.22541/au.159284919.96257497

Bohmann, K., Evans, A., Gilbert, M. T. P., Carvalho, G. R., Creer, S., Knapp, M., … de Bruyn, M. (2014). Environmental DNA for wildlife biology and biodiversity monitoring. *Trends in Ecology & Evolution*, *29*(6), 358–367. doi:https://doi.org/10.1016/j.tree.2014.04.003

Bokulich, N. A., Kaehler, B. D., Rideout, J. R., Dillon, M., Bolyen, E., Knight, R., … Gregory Caporaso, J. (2018). Optimizing taxonomic classification of marker-gene amplicon sequences with QIIME 2's q2-feature-classifier plugin. *Microbiome*, *6*(1), 90. doi:10.1186/s40168-018-0470-z

Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*, *30*(15), 2114–2120. doi:10.1093/bioinformatics/btu170

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., et al. (2000). Extensible markup language (XML) 1.0. W3C recommendation October.

Callahan, B. J., McMurdie, P. J., Rosen, M. J., Han, A. W., Johnson, A. J. A., & Holmes, S. P. (2016). DADA2: High-resolution sample inference from Illumina amplicon data. *Nature Methods*, *13*(7), 581–583. doi:10.1038/nmeth.3869

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., & Madden, T. L. (2009). BLAST+: architecture and applications. *BMC Bioinformatics*, *10*(1), 421. doi:10.1186/1471-2105-10-421

Capella-Gutiérrez, S., Silla-Martínez, J. M., & Gabaldón, T. (2009). trimAl: a tool for automated alignment trimming in large-scale phylogenetic analyses. *Bioinformatics*, *25*(15), 1972–1973. doi:10.1093/bioinformatics/btp348

Cherry, D. S. & Cairns, J. (1982). Biological monitoring part V—Preference and avoidance studies. *Water Research*, *16*(3), 263–301. doi:https://doi.org/10.1016/0043-1354(82)90189-0

Cherry, D. S., Dickson, K. L., Cairns Jr., J., & Stauffer, J. R. (1977). Preferred, Avoided, and Lethal Temperatures of Fish During Rising Temperature Conditions. *Journal of the Fisheries Research Board of Canada*, *34*(2), 239–246. doi:10.1139/f77-035

Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., & Sayers, E. W. (2015). GenBank. *Nucleic Acids Research*, *44*(D1), D67–D72. doi:10.1093/nar/gkv1276

Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., … de Hoon, M. J. L. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, *25*(11), 1422–1423. doi:10.1093/bioinformatics/btp163

Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., & Rice, P. M. (2009). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, *38*(6), 1767–1771. doi:10.1093/nar/gkp1137

Cordier, T., Forster, D., Dufresne, Y., Martins, C. I. M., Stoeck, T., & Pawlowski, J. (2018). Supervised machine learning outperforms taxonomy-based environmental DNA metabarcoding applied to biomonitoring. *Molecular Ecology Resources*, *18*(6), 1381–1391. doi:https://doi.org/10.1111/1755-0998.12926

Cornish-Bowden, A. (1985). Nomenclature for incompletely specified bases in nucleic acid sequences: rcommendations 1984. *Nucleic Acids Research*, *13*(9), 3021–3030. doi:10.1093/nar/13.9.3021

Cox, M. P., Peterson, D. A., & Biggs, P. J. (2010). SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. *BMC bioinformatics*, *11*, 485. doi:10.1186/1471-2105-11-485

Curd, E. E., Gold, Z., Kandlikar, G. S., Gomer, J., Ogden, M., O'Connell, T., ... Meyer, R. S. (2019). Anacapa Toolkit: An environmental DNA toolkit for processing multilocus metabarcode datasets. *Methods in Ecology and Evolution*, *10*(9), 1469–1475. doi:https://doi.org/10.1111/2041-210X.13214

de Souza, L. S., Godwin, J. C., Renshaw, M. A., & Larson, E. (2016). Environmental DNA (eDNA) Detection Probability Is Influenced by Seasonal Activity of Organisms. *PLOS ONE*, *11*(10), 1–15. doi:10.1371/journal.pone.0165273

Deagle, B. E., Jarman, S. N., Coissac, E., Pompanon, F., & Taberlet, P. (2014). DNA metabarcoding and the cytochrome *c* oxidase subunit I marker: not a perfect match. *Biology Letters*, *10*(9), 20140562. doi:10.1098/rsbl.2014.0562

Djemiel, C., Dequiedt, S., Karimi, B., Cottin, A., Girier, T., El Djoudi, Y., ... Terrat, S. (2020). BIOCOM-PIPE: a new user-friendly metabarcoding pipeline for the characterization of microbial diversity from 16S, 18S and 23S rRNA gene amplicons. *BMC Bioinformatics*, *21*(1), 492. doi:10.1186/s12859-020-03829-3

Djurhuus, A., Closek, C. J., Kelly, R. P., Pitz, K. J., Michisaki, R. P., Starks, H. A., ... Breitbart, M. (2020). Environmental DNA reveals seasonal shifts and potential interactions in a marine community. *Nature Communications*, *11*(1), 254. doi:10.1038/s41467-019-14105-1

Doorenspleet, K., Jansen, L., Oosterbroek, S., & Nijland, R. (2021). Accurate long-read eDNA metabarcoding of North Sea fish using Oxford Nanopore sequencing. *ARPHA Conference Abstracts*, *4*, e65550. doi:10.3897/aca.4.e65550

Dufresne, Y., Lejzerowicz, F., Perret-Gentil, L. A., Pawlowski, J., & Cordier, T. (2019). SLIM: a flexible web application for the reproducible processing of environmental DNA metabarcoding data. *BMC Bioinformatics*, *20*(1), 88. doi:10.1186/s12859-019-2663-2

Edgar, R. C. [Robert C]. (2013). UPARSE: highly accurate OTU sequences from microbial amplicon reads. *Nature methods*, *10*(10), 996–998. doi:10.1038/nmeth.2604

Edgar, R. C. [Robert C.]. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, *32*(5), 1792–1797. doi:10.1093/nar/gkh340

Edgar, R. C. [Robert C.]. (2010). Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, *26*(19), 2460–2461. doi:10.1093/bioinformatics/btq461

Edgar, R. C. [Robert C.] & Flyvbjerg, H. (2015). Error filtering, pair assembly and error correction for next-generation sequencing reads. *Bioinformatics*, *31*(21), 3476–3482. doi:10.1093/bioinformatics/btv401

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, *7*(1), 1–26.

Eichmiller, J. J., Best, S. E., & Sorensen, P. W. (2016). Effects of Temperature and Trophic State on Degradation of Environmental DNA in Lake Water. *Environmental Science & Technology*, *50*(4), 1859–1867. doi:10.1021/acs.est.5b05672

Eick, D. (2013). Habitat preferences of the burbot (*Lota lota*) from the River Elbe: an experimental approach. *Journal of Applied Ichthyology*, *29*(3), 541–548. doi:https://doi.org/10.1111/jai.12110

Elbrecht, V., Vamos, E. E., Meissner, K., Aroviita, J., & Leese, F. (2017). Assessing strengths and weaknesses of DNA metabarcoding-based macroinvertebrate identification for routine stream monitoring. *Methods in Ecology and Evolution*, *8*(10), 1265–1275. doi:https://doi.org/10.1111/2041-210X.12789

Fassler, J. & Cooper, P. (2011, July 14). *BLAST® Help [Internet]*: *BLAST Glossary* (N. C. for Biotechnology Information (US), Ed.). Bethesda (MD).

Fred Hutchinson Cancer Research Center, Computational Biology. (n.d.). TAXTASTIC. Retrieved January 9, 2021, from https://github.com/fhcrc/taxtastic

Gao, X., Lin, H., Revanna, K., & Dong, Q. (2017). A Bayesian taxonomic classification method for 16S rRNA gene sequences with improved species-level accuracy. *BMC Bioinformatics*, *18*(1), 247. doi:10.1186/s12859-017-1670-4

Goldberg, C. S., Turner, C. R., Deiner, K., Klymus, K. E., Thomsen, P. F., Murphy, M. A., ... Taberlet, P. (2016). Critical considerations for the application of environmental DNA methods to detect aquatic species. *Methods in Ecology and Evolution*, *7*(11), 1299–1307. doi:https://doi.org/10.1111/2041-210X.12595

Gordon, A., Hannon, G. et al. (2010). Fastx-toolkit. Retrieved December 20, 2020, from http://hannonlab.cshl.edu/fastx_toolkit/index.html

Hahn, C. & Lunt, D. (n.d.). metaBEAT - metaBarcoding and Environmental DNA analysis Tool. Retrieved January 3, 2021, from https://github.com/HullUni-bioinformatics/metaBEAT

Hänfling, B., Lawson Handley, L., Read, D. S., Hahn, C., Li, J., Nichols, P., ... Winfield, I. J. (2016). Environmental DNA metabarcoding of lake fish communities reflects long-term data from established survey methods. *Molecular Ecology*, *25*(13), 3101–3119. doi:https://doi.org/10.1111/mec.13660

Hansen, M. A., Oey, H., Fernandez-Valverde, S., Jung, C.-H., & Mattick, J. S. (2008). Biopieces: a bioinformatics toolset and framework. Retrieved December 20, 2020, from http://www.biopieces.org/

Harper, L., Buxton, A. S., Rees, H. C., Bruce, K., Brys, R., Halfmaerten, D., ... Hänfling, B. (2019). Prospects and challenges of environmental DNA (eDNA) monitoring in freshwater ponds. *Hydrobiologia*, *826*(1), 25–41. doi:10.1007/s10750-018-3750-5

He, B., Zhu, R., Yang, H., Lu, Q., Wang, W., Song, L., ... Lang, J. (2020). Assessing the Impact of Data Preprocessing on Analyzing Next Generation Sequencing Data. *Frontiers in Bioengineering and Biotechnology*, *8*, 817. doi:10.3389/fbioe.2020.00817

Hebert, P. D. N., Cywinska, A., Ball, S. L., & deWaard, J. R. (2003). Biological identifications through DNA barcodes. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, *270*(1512), 313–321. doi:10.1098/rspb.2002.2218

Holovachov, O., Haenel, Q., Bourlat, S. J., & Jondelius, U. (2017). Taxonomy assignment approach determines the efficiency of identification of OTUs in marine nematodes. *Royal Society Open Science*, *4*(8), 170315. doi:10.1098/rsos.170315

Huse, S. M., Welch, D. M., Morrison, H. G., & Sogin, M. L. (2010). Ironing out the wrinkles in the rare biosphere through improved OTU clustering. *Environmental Microbiology*, *12*(7), 1889–1898. doi:https://doi.org/10.1111/j.1462-2920.2010.02193.x

IUCN. (2018). *Gasterosteus aculeatus*. The IUCN Red List of Threatened Species. Retrieved June 4, 2021, from https://www.iucnredlist.org/species/8951/58295405

Kandlikar, G., Gold, Z., Cowen, M., Meyer, R., Freise, A., Kraft, N., … Curd, E. (2018). ranacapa: An R package and Shiny web app to explore environmental DNA data with exploratory statistics and interactive visualizations [version 1; peer review: 1 approved, 2 approved with reservations]. *F1000Research*, *7*(1734). doi:10.12688/f1000research.16680.1

Kasai, A., Takada, S., Yamazaki, A., Masuda, R., & Yamanaka, H. (2020). The effect of temperature on environmental DNA degradation of Japanese eel. *Fisheries Science*, *86*(3), 465–471. doi:10.1007/s12562-020-01409-1

Katoh, K. & Standley, D. M. (2013). MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. *Molecular Biology and Evolution*, *30*(4), 772–780. doi:10.1093/molbev/mst010

Kelly, R. P., Shelton, A. O., & Gallego, R. (2019). Understanding PCR Processes to Draw Meaningful Conclusions from Environmental DNA Studies. *Scientific Reports*, *9*(1), 12133. doi:10.1038/s41598-019-48546-x

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., … development team, J. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Scmidt (Eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (pp. 87–90). IOS Press.

Koch, L. F. (1957). Index of Biotal Dispersity. *Ecology*, *38*(1), 145–148.

Kottelat, M. & Freyhof, J. (2007). *Handbook of European freshwater fishes*. Publications Kottelat.

Kozlov, A. M., Zhang, J., Yilmaz, P., Glöckner, F. O., & Stamatakis, A. (2016). Phylogeny-aware identification and correction of taxonomically mislabeled sequences. *Nucleic Acids Research*, *44*(11), 5022–5033. doi:10.1093/nar/gkw396

Lamb, P. D., Hunter, E., Pinnegar, J. K., Creer, S., Davies, R. G., & Taylor, M. I. (2019). How quantitative is metabarcoding: A meta-analytical approach. *Molecular Ecology*, *28*(2), 420–430. doi:https://doi.org/10.1111/mec.14920

Langmead, B. & Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature Methods*, *9*(4), 357–359. doi:10.1038/nmeth.1923

Lawson Handley, L., Read, D. S., Winfield, I. J., Kimbell, H., Johnson, H., Li, J., … Hänfling, B. (2019). Temporal and spatial variation in distribution of fish environmental DNA in England's largest lake. *Environmental DNA*, *1*(1), 26–39. doi:https://doi.org/10.1002/edn3.5

Leuven, R., Hendriks, A., Huijbregts, M., Lenders, H., Matthews, J., & Velde, G. V. D. (2011). Differences in sensitivity of native and exotic fish species to changes in river temperature. *Current Zoology*, *57*(6), 852–862. doi:10.1093/czoolo/57.6.852

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., … 1000 Genome Project Data Processing Subgroup. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, *25*(16), 2078–2079. doi:10.1093/bioinformatics/btp352

Lladó Fernández, S., Větrovský, T., & Baldrian, P. (2019). The concept of operational taxonomic units revisited: genomes of bacteria that are regarded as closely related are often highly dissimilar. *Folia Microbiologica*, *64*(1), 19–23. doi:10.1007/s12223-018-0627-y

Locey, K. J. & Lennon, J. T. (2016). Scaling laws predict global microbial diversity. *Proceedings of the National Academy of Sciences*, *113*(21), 5970–5975. doi:10.1073/pnas.1521291113

Macher, T.-H., Beermann, A. J., & Leese, F. (2021). TaxonTableTools: A comprehensive, platform-independent graphical user interface software to explore and visualise DNA metabarcoding data. *Molecular Ecology Resources*. doi:https://doi.org/10.1111/1755-0998.13358

Magoč, T. & Salzberg, S. L. (2011). FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics*, *27*(21), 2957–2963. doi:10.1093/bioinformatics/btr507

Martin, M. (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal*, *17*(1), 10–12. doi:10.14806/ej.17.1.200

Mehner, T., Diekmann, M., Brämick, U., & Lemcke, R. (2005). Composition of fish communities in German lakes as related to lake morphology, trophic state, shore structure and human-use intensity. *Freshwater Biology*, *50*(1), 70–85. doi:https://doi.org/10.1111/j.1365-2427.2004.01294.x

Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J. 2014*(239).

Miya, M., Sato, Y., Fukunaga, T., Sado, T., Poulsen, J. Y., Sato, K., … Iwasaki, W. (2015). MiFish, a set of universal PCR primers for metabarcoding environmental DNA from fishes: detection of more than 230 subtropical marine species. *Royal Society Open Science*, *2*(7), 150088. doi:10.1098/rsos.150088

Muri, C. D., Handley, L. L., Bean, C. W., Li, J., Peirson, G., Sellers, G. S., … Hänfling, B. (2020). Read counts from environmental DNA (eDNA) metabarcoding reflect fish abundance and biomass in drained ponds. *Metabarcoding and Metagenomics*, *4*, e56959. doi:10.3897/mbmg.4.56959

Nearing, J. T., Douglas, G. M., Comeau, A. M., & Langille, M. G. I. (2018). Denoising the Denoisers: an independent evaluation of microbiome sequence error-correction approaches. *PeerJ*, *6*, e5364–e5364. 5364[PII]. doi:10.7717/peerj.5364

Normandeau, E. (n.d.). Environmental DNA metabarcoding analysis. Retrieved December 3, 2020, from https://github.com/enormandeau/barque

Nosova, A. Y., Kipen, V. N., Tsar, A. I., & Lemesh, V. A. (2020). Differentiation of Hybrid Progeny of Silver Carp (*Hypophthalmichthys molitrix Val.*) and Bighead Carp (*H. nobilis Rich.*) Based on Microsatellite Polymorphism. *Russian Journal of Genetics*, *56*(3), 317–323. doi:10.1134/S1022795420030126

Nugent, C. M. & Adamowicz, S. J. (2020). Alignment-free classification of COI DNA barcode data with the Python package Alfie. *Metabarcoding and Metagenomics*, *4*, e55815. doi:10.3897/mbmg.4.55815

O'Rourke, D. R., Bokulich, N. A., Jusino, M. A., MacManes, M. D., & Foster, J. T. (2020). A total crapshoot? Evaluating bioinformatic decisions in animal diet metabarcoding analyses. *Ecology and Evolution*, *10*(18), 9721–9739. doi:https://doi.org/10.1002/ece3.6594

Oksanen, J., Blanchet, F. G., Friendly, M., Kindt, R., Legendre, P., McGlinn, D., ... Wagner, H. (2019). *vegan: Community Ecology Package*. R package version 2.5-6.

Orlov, Y. L. & Potapov, V. N. (2004). Complexity: an internet resource for analysis of DNA sequence complexity. *Nucleic Acids Research*, *32*(suppl_2), W628–W633. doi:10.1093/nar/gkh466

Pauvert, C., Buée, M., Laval, V., Edel-Hermann, V., Fauchery, L., Gautier, A., ... Vacher, C. (2019). Bioinformatics matters: The accuracy of plant and soil fungal community data is highly dependent on the metabarcoding pipeline. *Fungal Ecology*, *41*, 23–33. doi:https://doi.org/10.1016/j.funeco.2019.03.005

Pavlopoulos, G. A., Soldatos, T. G., Barbosa-Silva, A., & Schneider, R. (2010). A reference guide for tree analysis and visualization. *BioData Mining*, *3*(1), 1. doi:10.1186/1756-0381-3-1

Puga, J. L., Krzywinski, M., & Altman, N. (2015). Bayes' theorem. *Nature Methods*, *12*(4), 277–278. doi:10.1038/nmeth.3335

R Core Team. (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Reuter, J. A., Spacek, D. V., & Snyder, M. P. (2015). High-Throughput Sequencing Technologies. *Molecular Cell*, *58*(4), 586–597. doi:https://doi.org/10.1016/j.molcel.2015.05.004

Riaz, T., Shehzad, W., Viari, A., Pompanon, F., Taberlet, P., & Coissac, E. (2011). ecoPrimers: inference of new DNA barcode markers from whole genome sequence analysis. *Nucleic Acids Research*, *39*(21), e145–e145. doi:10.1093/nar/gkr732

Risch, N. (1992). Genetic linkage: interpreting lod scores. *Science*, *255*(5046), 803–804. doi:10.1126/science.1536004

Rish, I. et al. (2001). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, *22*, pp. 41–46).

Rognes, T., Flouri, T., Nichols, B., Quince, C., & Mahé, F. (2016). VSEARCH: a versatile open source tool for metagenomics. *PeerJ*, *4*, e2584. doi:10.7717/peerj.2584

Santos, A., van Aerle, R., Barrientos, L., & Martinez-Urtaza, J. (2020). Computational methods for 16S metabarcoding studies using Nanopore sequencing data. *Computational and Structural Biotechnology Journal*, *18*, 296–305. doi:https://doi.org/10.1016/j.csbj.2020.01.005

Sato, Y., Miya, M., Fukunaga, T., Sado, T., & Iwasaki, W. (2018). MitoFish and MiFish Pipeline: A Mitochondrial Genome Database of Fish with an Analysis Pipeline for Environmental DNA Metabarcoding. *Molecular Biology and Evolution*, *35*(6), 1553–1555. doi:10.1093/molbev/msy074

Schenekar, T., Schletterer, M., Lecaudey, L. A., & Weiss, S. J. (2020). Reference databases, primer choice, and assay sensitivity for environmental metabarcoding: Lessons learnt from a re-evaluation of an eDNA fish assessment in the Volga headwaters. *River Research and Applications*, *36*(7), 1004–1013. doi:https://doi.org/10.1002/rra.3610

Schmieder, R., Lim, Y. W., Rohwer, F., & Edwards, R. (2010). TagCleaner: Identification and removal of tag sequences from genomic and metagenomic datasets. *BMC Bioinformatics*, *11*(1), 341. doi:10.1186/1471-2105-11-341

Sellers, G. S., Muri, C. D., Gómez, A., & Hänfling, B. (2018). Mu-DNA: a modular universal DNA extraction method adaptable for a wide range of sample types. *Metabarcoding and Metagenomics*, *2*, e24556. doi:10.3897/mbmg.2.24556

Sepulveda, A. J., Schabacker, J., Smith, S., Al-Chokhachy, R., Luikart, G., & Amish, S. J. (2019). Improved detection of rare, endangered and invasive trout in using a new large-volume sampling method for eDNA capture. *Environmental DNA*, *1*(3), 227–237. doi:https://doi.org/10.1002/edn3.23

SEQme. (2018). *Microbiome and Metagenome Data analysis workshop*. České Budějovice, Czech Republic.

Seymour, M. (2019). Rapid progression and future of environmental DNA research. *Communications Biology*, *2*(1), 80. doi:10.1038/s42003-019-0330-9

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379–423. doi:10.1002/j.1538-7305.1948.tb01338.x

Sigsgaard, E. E., Jensen, M. R., Winkelmann, I. E., Møller, P. R., Hansen, M. M., & Thomsen, P. F. (2020). Population-level inferences from environmental DNA—Current status and future perspectives. *Evolutionary Applications*, *13*(2), 245–262. doi:https://doi.org/10.1111/eva.12882

Stamatakis, A. (2014). RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, *30*(9), 1312–1313. doi:10.1093/bioinformatics/btu033

Strickler, K. M., Fremier, A. K., & Goldberg, C. S. (2015). Quantifying effects of UV-B, temperature, and pH on eDNA degradation in aquatic microcosms. *Biological Conservation*, *183*, 85–92. Special Issue: Environmental DNA: A powerful new tool for biological conservation. doi:https://doi.org/10.1016/j.biocon.2014.11.038

Taberlet, P., Bonin, A., Zinger, L., & Coissac, E. (2018a). DNA amplification and multiplexing. Environmental DNA. Oxford: Oxford University Press. doi:10.1093/oso/9780198767220.003.0006

Taberlet, P., Bonin, A., Zinger, L., & Coissac, E. (2018b). DNA metabarcode choice and design. Environmental DNA. Oxford: Oxford University Press. doi:10.1093/oso/9780198767220.003.0002

Taberlet, P., Bonin, A., Zinger, L., & Coissac, E. (2018c). DNA metabarcoding data analysis. Environmental DNA. Oxford: Oxford University Press. doi:10.1093/oso/9780198767220.003.0008

Taberlet, P., Bonin, A., Zinger, L., & Coissac, E. (2018d). Freshwater ecosystems. Environmental DNA. Oxford: Oxford University Press. doi:10.1093/oso/9780198767220.003.0012

Taberlet, P., Coissac, E., Hajibabaei, M., & Rieseberg, L. H. (2012). Environmental DNA. *Molecular Ecology*, *21*(8), 1789–1793. doi:https://doi.org/10.1111/j.1365-294X.2012.05542.x

Taberlet, P., Coissac, E., Pompanon, F., Brochmann, C., & Willerslev, E. (2012). Towards next-generation biodiversity assessment using DNA metabarcoding. *Molecular Ecology*, *21*(8), 2045–2050. doi:https://doi.org/10.1111/j.1365-294X.2012.05470.x

Tange, O. (2020). GNU Parallel 20201122 ('Biden'). Zenodo. doi:10.5281/zenodo.4284075

Thomsen, P. F. & Willerslev, E. (2015). Environmental DNA – An emerging tool in conservation for monitoring past and present biodiversity. *Biological Conservation*, *183*, 4–18. Special Issue: Environmental DNA: A powerful new tool for biological conservation. doi:https://doi.org/10.1016/j.biocon.2014.11.019

Valentini, A., Taberlet, P., Miaud, C., Civade, R., Herder, J., Thomsen, P. F., ... Dejean, T. (2016). Next-generation monitoring of aquatic biodiversity using environmental DNA metabarcoding. *Molecular Ecology*, *25*(4), 929–942. doi:https://doi.org/10.1111/mec.13428

van de Pol, I., Flik, G., & Gorissen, M. (2017). Comparative Physiology of Energy Metabolism: Fishing for Endocrine Signals in the Early Vertebrate Pool. *Frontiers in Endocrinology*, *8*, 36. doi:10.3389/fendo.2017.00036

Van Rossum, G. & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

Wang, Q., Garrity, G. M., Tiedje, J. M., & Cole, J. R. (2007). Naive Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy. *Applied and Environmental Microbiology*, *73*(16), 5261–5267. doi:10.1128/AEM.00062-07

Wangensteen, O. S. & Turon, X. (2017). Metabarcoding Techniques for Assessing Biodiversity of Marine Animal Forests. In S. Rossi, L. Bramanti, A. Gori, & C. Orejas (Eds.), *Marine animal forests: The ecology of benthic biodiversity hotspots* (pp. 445–473). Cham: Springer International Publishing. doi:10.1007/978-3-319-21012-4_53

Weigand, H., Beermann, A. J., Čiampor, F., Costa, F. O., Csabai, Z., Duarte, S., … Ekrem, T. (2019). DNA barcode reference libraries for the monitoring of aquatic biota in Europe: Gap-analysis and recommendations for future work. *Science of The Total Environment*, *678*, 499–524. doi:https://doi.org/10.1016/j.scitotenv.2019.04.247

Whittaker, R. H. (1972). Evolution and measurement of species diversity. *TAXON*, *21*(2-3), 213–251. doi:https://doi.org/10.2307/1218190

Willemsen, J. (1980). Fishery-aspects of eutrophication. *Hydrobiological Bulletin*, *14*(1), 12–21. doi:10.1007/BF02260268

Xiong, W. & Zhan, A. (2018). Testing clustering strategies for metabarcoding-based investigation of community–environment interactions. *Molecular Ecology Resources*, *18*(6), 1326–1338. doi:https://doi.org/10.1111/1755-0998.12922

Zafeiropoulos, H., Viet, H. Q., Vasileiadou, K., Potirakis, A., Arvanitidis, C., Topalis, P., … Pafilis, E. (2020). PEMA: a flexible Pipeline for Environmental DNA Metabarcoding Analysis of the 16S/18S ribosomal RNA, ITS, and COI marker genes. *GigaScience*, *9*(3). giaa022. doi:10.1093/gigascience/giaa022

Zielezinski, A., Vinga, S., Almeida, J., & Karlowski, W. M. (2017). Alignment-free sequence comparison: benefits, applications, and tools. *Genome Biology*, *18*(1), 186. doi:10.1186/s13059-017-1319-7

List of Tables

List of Figures

List of Source Codes

Appendices

```
#!/usr/bin/env python3
import os
import argparse
import subprocess
from Bio import SeqIO
from tqdm import tqdm
from multiprocessing import cpu_count

"""Clustering sequences and removing redundancy from the file.

Removing redundancy by clustering sequences with vsearch.
"""

def getArguments():
    """Get arguments from terminal

    This function gets arguments from terminal via argparse

    Returns
    -------------
    arguments: Namespace
        Namespace object with all arguments
    """

    num_threads = cpu_count() - 2
    if num_threads < 1:
        num_threads = 1

    parser = argparse.ArgumentParser(
        description='Removing redundancy from the gb file.')
    parser.add_argument('gb', type=argparse.FileType('r'),
                    help='genbank file .gb')
    parser.add_argument('-n', '--num_threads', nargs='?', type = int,
                        const=num_threads, default=num_threads,
                     help="Number of threads to be executed in parallel.")
```

```
36
37      return parser.parse_args()
38
39  def gb2fasta(gb):
40      """Convert genbank file to fasta.
41
42      This function converts a file in a genbank
43      format to a file in a fasta file format.
44
45      Parameters
46      -------------
47      gb: io.TextIOWrapper
48          Genbank file
49
50      Returns
51      -------------
52      fasta: str
53          Name of the new fasta file
54
55      """
56
57      base = os.path.basename(gb.name)
58      name = os.path.splitext(base)[0]
59
60      fasta = "{}.fasta".format(name)
61      SeqIO.convert(gb, "genbank", fasta, "fasta")
62
63      return fasta
64
65  def clustering(fasta, num_threads):
66      """Clustering sequences with VSEARCH
67
68      This function uses VSEARCH suite to
69      cluster the sequences using the cluster_fast
70      algorithm.
71
72
73      Parameters
74      -------------
75      fasta: str
76          Fasta file name
77      num_threads: int
78          Number of threads to be used
79      """
80
81      vsearch = "vsearch -threads {num_threads} --cluster_fast {fasta}" \
```

```
            " --strand both --uc cluster.uc --id 1 --query_cov 1"
    subprocess.call(vsearch.format(num_threads = num_threads,
                                   fasta = fasta) , shell=True)

def removing_redundancy(gb):
    """Removing redundancy from genbank file after
    clustering sequences

    This function parses the uc file generated
    after clustering the sequences to discard
    the sequences from the original file.


    Parameters
    -------------
    gb: io.TextIOWrapper
        Genbank file
    """

    with open("cluster.uc") as cluster:
        gb_all_data = SeqIO.index(gb.name, "genbank")

        base = os.path.basename(gb.name)
        name = os.path.splitext(base)[0]

        gb_c_name = "{}_c.gb".format(name)
        with open(gb_c_name, "w") as gb_c:
            for uc in tqdm(cluster, desc="Removing redundancy"):
                if uc[0] == "C":
                    row = uc.split("\t")
                    print(gb_all_data.get_raw(row[-2]).decode(),
                          file=gb_c, end="")


if __name__ == "__main__":
    args = getArguments()
    fasta = gb2fasta(args.gb)
    clustering(fasta, args.num_threads)
    print()
    removing_redundancy(args.gb)
```

Source Code A.1: VSEARCH clustering to remove redundant sequences.

```
#!/usr/bin/env python3
import os
```

```
3  import argparse
4  from Bio import SeqIO
5  from tqdm import tqdm
6
7  """
8  Filter sequences by length and exclude all the outliers.
9
10 Find all sequences with length lower than threshold, and
11 remove them from the genbank file.
12 """
13
14 def getArguments():
15     """Get arguments from terminal
16
17     This function gets arguments from terminal via argparse
18
19     Returns
20     -------------
21     arguments: Namespace
22         Namespace object with all arguments
23     """
24
25     parser = argparse.ArgumentParser(
26         description='Filter sequences by length and exclude all the outliers.')
27     parser.add_argument('gb', type=argparse.FileType('r'),
28                         help='genbank file .gb')
29     parser.add_argument('-t', '--threshold', nargs='?', const=100,
30                         default=100, type = int,
31                         help="Threshold for removing the sequences, default: 100")
32
33     return parser.parse_args()
34
35 def filter_by_length(gb, threshold):
36     """Filter sequences by length and exclude all the outliers
37
38     This function removes all sequences with sequence length
39     lower than the threshold.
40
41     Parameters
42     -------------
43     gb : io.TextIOWrapper
44         Genbank file
45
46     threshold : int
47         Threshold value
48     """
```

```
49
50      gb_all_data = SeqIO.index(gb.name, "genbank")
51      gb_new_data = []
52
53      base = os.path.basename(gb.name)
54      name = os.path.splitext(base)[0]
55
56      gb_l_name = "{}_l.gb".format(name)
57      for acession in tqdm(gb_all_data,
58          desc="Removing sequences smaller than {}".format(threshold)):
59
60          record = gb_all_data.get(acession)
61          if len(record.seq) > threshold:
62              gb_new_data.append(record)
63
64      SeqIO.write(gb_new_data, gb_l_name, "genbank")
65
66
67  if __name__ == "__main__":
68      args = getArguments()
69      filter_by_length(args.gb, args.threshold)
```

Source Code A.2: Discard sequences smaller than the threshold.

```
1   #!/usr/bin/env python3
2   import os
3   import pathlib
4   import argparse
5   import tempfile
6   import datetime
7   from tqdm import tqdm
8   from Bio import SeqIO
9   from Bio import Entrez
10  from ete3 import NCBITaxa
11  from Bio.Align.Applications import *
12  from multiprocessing import cpu_count
13
14  """
15  Sequences Alignment:
16
17  Align sequences by group of sequences
18  based on taxonomic rank chosen (example: species).
19
20  A file with primers can be provided to be included
21  in the alignment.
```

```
22
23  Multiple sequence alignment program can be chosen.
24  """
25
26  CHOICES_P = ["mafft", "clustalo", "muscle", "all"]
27
28  CHOICES_R = ["superkingdom", "kingdom", "phylum",
29               "subphylum", "superclass", "class",
30               "subclass", "order", "suborder",
31               "family", "subfamily", "tribe",
32               "genus", "species", "all"]
33
34  CHOICES_S = ["taxdump", "ncbi"]
35
36  LINEAGES = {}
37
38  PATH_MAIN = "{}_alignments/".format(datetime.datetime.now().strftime("%d%m%Y_%H%M%S")
      ↪ )
39
40  LOG = "{}Alignment.log".format(PATH_MAIN)
41
42
43  def getArguments():
44      """Get arguments from terminal
45
46      This function gets arguments from terminal via argparse
47
48      Returns
49      -------------
50      arguments: Namespace
51          Namespace object with all arguments
52      """
53
54      num_threads = cpu_count() - 2
55      if num_threads < 1:
56          num_threads = 1
57
58      parser = argparse.ArgumentParser(
59          description="Alignment of sequences from a genbank"\
60                      " file based on taxonomic rank.")
61      parser.add_argument("gb", type=argparse.FileType("r"),
62                          help="genbank file (.gb)")
63      parser.add_argument("-p", "--program", nargs="*", default=["mafft"],
64                          choices=CHOICES_P, type = lambda s : s.lower(),
65                          help="Multiple sequence alignment program, default: mafft")
66      parser.add_argument("-r", "--rank", nargs="*", default=["species"],
```

```
                    choices=CHOICES_R, type = lambda s : s.lower(),
                    help="Taxonomic classification rank to be used"\
                    " to separate the groups, default: species")
    parser.add_argument("-s", "--source", nargs="?", const="taxdump", default="taxdump
        ↪ ",
                    choices=CHOICES_S, type = lambda s : s.lower(),
                    help="Source to be used to collect"\
                    " the info about the taxonomic rank, default: taxdump")
    parser.add_argument("-pr", "--primers", type=argparse.FileType("r"),
                    help="A fasta file with primers")
    parser.add_argument("-sp", "--species_from_file", action="store_true",
                    help="Should the species name from file"\
                    " be used or be collected from NCBI/taxdump?")
    parser.add_argument('-n', '--num_threads', nargs='?', type = int,
                    const=num_threads, default=num_threads,
                    help="Number of threads to be executed in parallel.")

    return parser.parse_args()

def get_tax_lineage(taxonid, source):
    """Return taxonomy lineage information

    This function uses Biopython library to connect NCBI database
    and search for taxonomy information or ete3 to download
    taxdump file and search the information locally.

    Parameters
    -------------
    taxonid : string
        Taxonomic id of the species
    source : string
        Source to be used to collect the info about the taxonid

    Returns
    -------------
    lineage: dict
        Species lineage

    """

    if taxonid not in LINEAGES:
        if source == "taxdump":
            ncbi_taxdump = NCBITaxa()
            lineage_ids = ncbi_taxdump.get_lineage(taxonid)
            ranks = ncbi_taxdump.get_rank(lineage_ids)
            names = ncbi_taxdump.get_taxid_translator(lineage_ids)
```

```
            lineage = {ranks[i]:names[i] for i in lineage_ids}

            LINEAGES[taxonid] = lineage
            return LINEAGES[taxonid]

        while True:
            data = ""
            try:
                Entrez.email = "Your.Name.Here@example.org"
                handle = Entrez.efetch(id = taxonid, db = "taxonomy", retmode = "xml")
                data = Entrez.read(handle)
                handle.close()
            except Exception as e:
                with open(LOG, "a") as log:
                    print("Error when searching information about {}".format(taxonid),
                        file=log)

            if data:
                break

        lineage = {d["Rank"]:d["ScientificName"] for d in data[0]["LineageEx"]}
        lineage[data[0]["Rank"]] = data[0]["ScientificName"]
        LINEAGES[taxonid] = lineage


    return LINEAGES[taxonid]

def read_sequences(gb, rank, source, species_from_file):
    """Read the genbank file and parse the sequences
    for the taxonomic rank

    This function uses Biopython library to scan the genbank file
    and parse the sequences for the taxonomic rank.

    Parameters
    ----------
    gb : io.TextIOWrapper
        A genbank file
    rank: string
        Taxonomic rank
    source : string
        Source to be used to collect the info about the taxonid
    species_from_file: bool
        Indicate if species name from file should be used

    Returns
```

```
    -------------
    sequences: dictionary
        A dictionary with key representing ranks and values
        representing sequences in fasta format with species name
        as the header of the sequence

    """

    print("Rank: {}".format(rank))
    sequences = {}
    for r in tqdm(gb, desc="Reading sequences"):
        record = gb.get(r)
        for feature in record.features:
            if feature.type == "source" and \
                "taxon" in feature.qualifiers["db_xref"][0]:
                taxonid = feature.qualifiers["db_xref"][0].split(":")[1]

        lineage = get_tax_lineage(taxonid, source)

        if species_from_file:
            lineage["species"] = record.features[0].qualifiers["organism"][0]
        else:
            if "species" not in lineage:
                lineage["species"] = record.features[0].qualifiers["organism"][0]

        try:
            if lineage[rank] not in sequences:
                sequences[lineage[rank]] = [">{}_{}\n{}".format(
                                            lineage["species"].replace(" ", "_"),
                                            record.id, record.seq)]
            else:
                sequences[lineage[rank]].append(">{}_{}\n{}".format(
                                            lineage["species"].replace(" ", "_"),
                                            record.id, record.seq))
        except:
            with open(LOG, "a") as log:
                print("\nRank '{}' not found for organism '{}', taxonid '{}'".format(
                    rank, lineage["species"], taxonid), file=log)

    return sequences

def alignment(sequences, program, rank, primers, num_threads):
    """Sequences alignment using the program chosen by rank level.

    This function uses either MAFFT, Clustal omega, or Muscle
    to perform a multiple alignment for each group
```

```
    of sequences in the rank(s) chosen.

    Parameters
    -------------
    sequences : dictionary
        A dictionary with sequences
    program: string
        Program to be used to align the sequences
    rank: string
        Taxonomic rank
    primers: list
        A list of primers to be included in the alignment
    num_threads: int
        Number of threads to be used

    """

    print("Program: {}".format(program))
    path_alignments = "{}{}_alignments/{}".format(PATH_MAIN, program, rank)
    pathlib.Path(path_alignments).mkdir(parents=True, exist_ok=True)

    for seq in tqdm(sequences, desc="Alignment"):
        with tempfile.NamedTemporaryFile(mode="w") as temp:
            temp.write("\n".join(primers + sorted(sequences[seq])))
            temp.seek(0)

            path_seq = "{}/{}".format(path_alignments, seq.replace(" ", "_"))
            pathlib.Path(path_seq).mkdir(parents=True, exist_ok=True)

            try:
                if program == "clustalo":
                    cmdline = ClustalOmegaCommandline(
                                    infile=temp.name,
                                    outfile="{}/{}.aln".format(path_seq,
                                    seq.replace(" ", "_")),
                                    guidetree_out="{}/{}.tree".format(path_seq,
                                    seq.replace(" ", "_")),
                                    force=True,
                                    threads = num_threads
                                    )
                    cmdline()

                elif program == "mafft":
                    cmdline = MafftCommandline(input=temp.name, treeout=True,
                                        localpair=True, maxiterate=1000,
                                        adjustdirectionaccurately=True,
```

```
                                            thread = num_threads)
                stdout, stderr = cmdline()
                os.rename("{}.tree".format(temp.name), "{}/{}.tree".format(path_seq,
                    ↪
                                          seq.replace(" ", "_")))
                with open("{}/{}.aln".format(path_seq,
                                          seq.replace(" ", "_")), "w") as align:
                        print(stdout, file=align)

            elif program == "muscle":
                cmdline = MuscleCommandline(input=temp.name,
                                            clwout="{}/{}.clw".format(path_seq,
                                            seq.replace(" ", "_")),
                                            fastaout="{}/{}.fa".format(path_seq,
                                            seq.replace(" ", "_")),
                                            htmlout="{}/{}.html".format(path_seq,
                                            seq.replace(" ", "_")),
                                            msfout="{}/{}.msf".format(path_seq,
                                            seq.replace(" ", "_")),
                                            phyiout="{}/{}.phy".format(path_seq,
                                            seq.replace(" ", "_")),
                                            tree1="{}/{}_1.tree".format(path_seq,
                                            seq.replace(" ", "_")),
                                            tree2="{}/{}_2.tree".format(path_seq,
                                            seq.replace(" ", "_"))
                                            )

                cmdline()

        except Exception as e:
            if os.listdir(path_seq) == []:
                os.rmdir(path_seq)
            with open(LOG, "a") as log:
                print("{species}: {error}".format(
                        species=seq, error=e), file=log)
            continue


if __name__ == "__main__":
    args = getArguments()

    pathlib.Path(PATH_MAIN).mkdir(parents=True, exist_ok=True)
    gb = SeqIO.index(args.gb.name, "genbank")
    with open(LOG, "w"): pass

    primers = []
```

```
    if args.primers:
        try:
            primers = [">{}\n{}".format(r.description.replace(" ", "_"), r.seq)
                       for r in SeqIO.parse(args.primers, "fasta")]
        except:
            with open(LOG, "a") as log:
                print("Fasta primers not found!! " +
                      "Continuing without primers", file=log)

    ranks = args.rank
    if "all" in args.rank:
        ranks = CHOICES_R[:-1]

    programs = args.program
    if "all" in args.program:
        programs = CHOICES_P[:-1]

    for rank in ranks:
        sequences = read_sequences(gb, rank, args.source,
                                   args.species_from_file)
        for program in programs:
                alignment(sequences, program, rank, primers, args.num_threads)
```

Source Code A.3: Alignment of the sequences using a multiple alignment algorithm.

```
#!/usr/bin/env python3
import os
import argparse
import tempfile
import subprocess
from Bio import SeqIO
from tqdm import tqdm

"""
Trim alignment to remove large gaps in both extremities
of the sequences for building the phylogenetic tree.
The new trimmed alignment should not be used as a
reference to map reads.
"""

def getArguments():
    """Get arguments from terminal

    This function gets arguments from terminal via argparse

```

```
21      Returns
22      -------------
23      arguments: Namespace
24          Namespace object with all arguments
25      """
26
27      parser = argparse.ArgumentParser(
28          description="Trim alignment to remove large gaps in the" +
29                      " extremities of the sequences for building the tree.")
30      parser.add_argument('aln', type=argparse.FileType('r'),
31                      help='alignment file')
32
33      return parser.parse_args()
34
35  def trim_alignment(aln):
36      """Trim the alignment to remove large gaps
37
38      This function uses trimal to remove large gaps from the
39      multiple alignment sequences.
40
41      Parameters
42      -------------
43      aln: io.TextIOWrapper
44          Alignment file
45      """
46
47      base = os.path.basename(aln.name)
48      name = os.path.splitext(base)[0]
49
50      trimal = "trimal -in {} -gappyout > {}_t.aln".format(aln.name, name)
51      subprocess.call(trimal , shell=True)
52
53  def convert2Phy(aln):
54      """Convert Alignment to PHYLIP's format
55
56      This function uses biopython package to read an alignment file,
57      remove the name of the species (only accession number is kept),
58      and convert it to PHYLIP's format.
59
60      Parameters
61      -------------
62      aln: io.TextIOWrapper
63          Alignment file
64      """
65
66      base = os.path.basename(aln.name)
```

```
    name = os.path.splitext(base)[0]

    aln_t = "{}_t.aln".format(name)
    phy = "{}.phy".format(name)

    with open(aln_t) as aln_file:
        aln_f = aln_file.read().split(">")

    for i in range(len(aln_f)):
        if aln_f[i]:
            aln_s = aln_f[i].split("_")
            aln_f[i] = aln_s[-1]
            if not aln_f[i][0].isalpha():
                aln_f[i] = "_".join(aln_s[-2:])

    with tempfile.NamedTemporaryFile(mode='w') as temp:
         temp.write(">".join(aln_f))
         temp.seek(0)

         SeqIO.convert(temp.name, "fasta", phy, "phylip-relaxed")



if __name__ == "__main__":
    args = getArguments()
    trim_alignment(args.aln)
    convert2Phy(args.aln)
```

Source Code A.4: Alignment trimming to remove poorly aligned regions.

```
#!/usr/bin/env python3
import os
import csv
import pathlib
import argparse
import datetime
import subprocess
from Bio import SeqIO
from Bio import Entrez
from tqdm import tqdm
from ete3 import NCBITaxa

"""
The SATIVA Algorithm is used for taxonomically
mislabelled sequences identification and to
```

```
suggest corrections.
"""

CHOICES_S = ["ncbi", "taxit", "taxdump"]

TAX_LEVELS = ["superkingdom","phylum","class","order","family","genus","species"]
LINEAGES = {}

PATH_MAIN = "{}_sativa/".format(datetime.datetime.now().strftime("%d%m%Y_%H%M%S"))
PATH_TO_SATIVA_TAX = "{PATH_MAIN}Sativa.tax".format(PATH_MAIN=PATH_MAIN)
PATH_TO_SATIVA = "sativa/"

PATH_TO_TAXID = "{PATH_INFORMED}/TaxIDS.txt"
PATH_TO_TAXA = "{PATH_INFORMED}/Taxa.csv"
PATH_TO_DB = "{PATH_INFORMED}/ncbi_taxonomy.db"

def getArguments():
    """Get arguments from terminal

    This function gets arguments from terminal via argparse

    Returns
    -------------
    arguments: Namespace
        Namespace object with all arguments
    """

    parser = argparse.ArgumentParser(
        description="Identification of taxonomically"\
                    " mislabelled sequences")
    parser.add_argument("gb", type=argparse.FileType("r"),
                    help="genbank format file (.gb)")
    parser.add_argument("phy", type=argparse.FileType("r"),
                    help="PHYLIP multiple sequence alignment format file (.phy)")
    parser.add_argument("-s", "--source", nargs="?", const="taxdump", default="taxdump
        ↪ ",
                    choices=CHOICES_S, type = lambda s : s.lower(),
                    help="Source to be used to collect"\
                    " the info about the taxonomic rank, default: taxdump")
    parser.add_argument('-p', '--path_to_sativa', nargs='?', type = str,
                    const=PATH_TO_SATIVA, default=PATH_TO_SATIVA,
                    help="Path to sativa code.")
    parser.add_argument('-t', '--path_to_taxid_files', nargs='?', type = str,
                    const=PATH_MAIN, default=PATH_MAIN,
                    help="Path to taxit files.")

```

```
61      return parser.parse_args()
62
63  def download_and_install_sativa():
64      """Check if sativa is installed
65
66      This function checks if the path to sativa exists
67      and if sativa was installed.
68
69      """
70
71      print("Downloading sativa ...")
72      subprocess.call("git clone --recursive" \
73                      " https://github.com/"\
74                      "amkozlov/sativa.git {path_to_sativa}".format(
75                      path_to_sativa=PATH_TO_SATIVA),
76                      shell=True)
77
78      print("Installing sativa ...")
79      subprocess.call("bash {path_to_sativa}/install.sh".format(
80                      path_to_sativa=PATH_TO_SATIVA),
81                      shell=True)
82
83  def get_tax_lineage(taxonid, source, tax_rank_id={}):
84      """Return taxonomy lineage information
85
86      This function uses either Biopython library to connect
87      NCBI database and search for taxonomy information
88      or searches the information locally by using ete3 taxdump
89      file or taxit program to create sql version of it.
90
91      Parameters
92      -------------
93      taxonid : string
94          Taxonomic id of the species
95      source : string
96          Source to be used to collect the info about the taxonid
97      tax_rank_id: dict
98          Taxonomic rank and id
99
100     Returns
101     -------------
102     lineage: dict
103         Species lineage
104
105     """
106
```

```
107     if taxonid not in LINEAGES:
108         if source == "taxdump":
109             ncbi_taxdump = NCBITaxa()
110             lineage_ids = ncbi_taxdump.get_lineage(taxonid)
111             ranks = ncbi_taxdump.get_rank(lineage_ids)
112             names = ncbi_taxdump.get_taxid_translator(lineage_ids)
113             lineage = {ranks[i]:names[i] for i in lineage_ids}
114 
115             LINEAGES[taxonid] = lineage
116             return LINEAGES[taxonid]
117 
118         if source == "taxit":
119             lineage = {level:tax_rank_id[tax_rank_id[
120                         taxonid][level]]["tax_name"]
121                         for level in TAX_LEVELS}
122 
123             LINEAGES[taxonid] = lineage
124             return LINEAGES[taxonid]
125 
126 
127         while True:
128             data = ""
129             try:
130                 Entrez.email = "Your.Name.Here@example.org"
131                 handle = Entrez.efetch(id = taxonid, db = "taxonomy", retmode = "xml")
132                 data = Entrez.read(handle)
133                 handle.close()
134             except Exception as e:
135                 with open(LOG, "a") as log:
136                     print("Error when searching information about {}".format(taxonid),
137                         file=log)
138 
139             if data:
140                 break
141 
142         lineage = {d["Rank"]:d["ScientificName"] for d in data[0]["LineageEx"]}
143         lineage[data[0]["Rank"]] = data[0]["ScientificName"]
144         LINEAGES[taxonid] = lineage
145 
146 
147     return LINEAGES[taxonid]
148 
149 def parse_taxID(gb):
150     """Parse taxon ids from genbank file
151 
152     This function uses Biopython library to parse
```

```
153     taxon ids and create a file with them.
154
155     Parameters
156     -------------
157     gb: io.TextIOWrapper
158         Genbank file
159
160     """
161     tax_ids = set()
162     gb = SeqIO.index(gb.name, "genbank")
163
164     for r in tqdm(gb, desc="Reading sequences"):
165         record = gb.get(r)
166         taxonid = record.features[0].qualifiers["db_xref"][0].split(":")[1]
167         tax_ids.add(taxonid)
168
169     with open(PATH_TO_TAXID, "w") as out_taxids:
170         out_taxids.write("\n".join(tax_ids))
171
172 def taxit():
173     """Download a create taxonomic database using taxit
174
175     This function executes taxit to download taxonomy database
176     and creates a table with the taxonomic lineages.
177     """
178
179     print("Downloading database ...")
180     subprocess.call("taxit new_database {PATH_TO_DB} -p {PATH_DOWNLOAD}".format(
181                     PATH_TO_DB=PATH_TO_DB,
182                     PATH_DOWNLOAD=PATH_MAIN) , shell=True)
183
184     print("Creating tax table ...")
185     subprocess.call("taxit taxtable {PATH_TO_DB}".format(PATH_TO_DB=PATH_TO_DB) +
186                     " -f {PATH_TO_TAXID}".format(PATH_TO_TAXID=PATH_TO_TAXID) +
187                     " -o {PATH_TO_TAXA}".format(PATH_TO_TAXA=PATH_TO_TAXA),
188                     shell=True)
189
190     print("DONE!")
191
192 def parse_taxa():
193     """Parse taxonomic information from Taxa.csv
194
195     This function opens Taxa.csv file to parse
196     tax id and lineage rank.
197
198     Returns
```

```
    -------------
    tax_rank_id: dict
        Taxonomic rank and id
    """

    with open(PATH_TO_TAXA) as taxa_input:
        taxa = csv.DictReader(taxa_input)
        tax_rank_id = {row["tax_id"]:row for row in taxa}

    return tax_rank_id

def tax4Sativa(gb, source, tax_rank_id={}):
    """Generate a taxonomic file for sativa
    using NCBI, taxa.csv from taxit, or taxdump with ete3

    This function creates a file with the taxon id,
    and the tax levels for ech taxon id.

    Parameters
    -------------
    gb: io.TextIOWrapper
        Genbank file
    source : string
        Source to be used to collect the info about the taxonid
    tax_rank_id: dict
        Taxonomic rank and id
    """

    sativa_taxes = []

    gb = SeqIO.index(gb.name, "genbank")

    for r in tqdm(gb, desc="Creating Sativa.tax"):
        record = gb.get(r)
        taxonid = record.features[0].qualifiers["db_xref"][0].split(":")[1]
        lineage = get_tax_lineage(taxonid, source, tax_rank_id)

        sativa_tax = "{}\t".format(record.id)
        for level in TAX_LEVELS:
            if level not in lineage:
                sativa_tax += "unknown;"
                continue

            sativa_tax += "{};".format(lineage[level])

        sativa_taxes.append(sativa_tax[:-1])
```

```
245
246     with open("{PATH_TO_SATIVA_TAX}".format(
247                 PATH_TO_SATIVA_TAX=PATH_TO_SATIVA_TAX
248                 ), "w") as sativa_tax_output:
249         sativa_tax_output.write("\n".join(sativa_taxes))
250
251 def sativa(phy):
252     """Run sativa for identification of taxonomically
253     mislabelled sequences
254
255     This function executes SATIVA algorithm to identify
256     taxonomically mislabelled sequences.
257
258     phy: io.TextIOWrapper
259         PHYLIP multiple sequence alignment format
260     """
261
262     print("SATIVA ...")
263     subprocess.call("python {PATH_TO_SATIVA}/sativa.py".format(
264                     PATH_TO_SATIVA=PATH_TO_SATIVA) +
265                     " -s {PATH_TO_PHY} -t {PATH_TO_SATIVA_TAX}".format(
266                     PATH_TO_PHY=phy.name, PATH_TO_SATIVA_TAX=PATH_TO_SATIVA_TAX) +
267                     " -x zoo -n 12S -o {PATH_MAIN}sativa_result/".format(
268                     PATH_MAIN=PATH_MAIN) +
269                     " -T 10 -v" , shell=True)
270     print("DONE!")
271
272
273 if __name__ == "__main__":
274     args = getArguments()
275
276     pathlib.Path("{PATH_MAIN}sativa_result/".format(
277                 PATH_MAIN=PATH_MAIN)).mkdir(
278                 parents=True, exist_ok=True)
279     PATH_TO_SATIVA = args.path_to_sativa
280
281     if not os.path.isfile("{path_to_sativa}/sativa.py".format(
282                           path_to_sativa=PATH_TO_SATIVA)):
283         download_and_install_sativa()
284
285     if args.source == "ncbi":
286
287         tax4Sativa(args.gb, args.source)
288         sativa(args.phy)
289
290     elif args.source == "taxit":
```

```
291
292         PATH_TO_TAXID = "{PATH_INFORMED}/TaxIDS.txt".format(
293                         PATH_INFORMED=args.path_to_taxid_files)
294         PATH_TO_TAXA = "{PATH_INFORMED}/Taxa.csv".format(
295                         PATH_INFORMED=args.path_to_taxid_files)
296         PATH_TO_DB = "{PATH_INFORMED}/ncbi_taxonomy.db".format(
297                         PATH_INFORMED=args.path_to_taxid_files)
298
299         if not os.path.isfile(PATH_TO_TAXA):
300             parse_taxID(args.gb)
301             taxit()
302
303         tax_rank_id = parse_taxa()
304         tax4Sativa(args.gb, args.source, tax_rank_id)
305         sativa(args.phy)
306
307     elif args.source == "taxdump":
308
309         tax4Sativa(args.gb, args.source)
310         sativa(args.phy)
```

Source Code A.5: Identify taxonomically mislabelled sequences.

```
1  #!/usr/bin/env python3
2  import os
3  import argparse
4  from Bio import SeqIO
5  from tqdm import tqdm
6
7  """
8  Remove mislabelled sequences identified by Sativa algorithm.
9
10 All the sequences identified by Sativa are removed from
11 the genbank file.
12 """
13
14 def getArguments():
15     """Get arguments from terminal
16
17     This function gets arguments from terminal via argparse
18
19     Returns
20     -------------
21     arguments: Namespace
22         Namespace object with all arguments
```

```
    """

    parser = argparse.ArgumentParser(
        description="Remove mislabelled sequences.")
    parser.add_argument("gb", type=argparse.FileType("r"),
                    help="genbank file (.gb)")
    parser.add_argument("mis", type=argparse.FileType("r"),
                    help="file with mislabelled sequences" \
                    " resulted after running sativa (.mis)")

    return parser.parse_args()

def parse_mis(mis):
    """Read file.mis and parse all sequences which need
    to be removed.

    This function parses the acession numbers in file.mis
    that should be removed from the genbank file.

    Parameters
    ------------
    mis : io.TextIOWrapper
        Sativa mislabelled sequences file

    Returns
    -------------
    acession_numbers: list
        List of acession numbers
    """

    acession_numbers = {m.split("\t")[0]:m.split("\t")[4] for m in mis
                        if not m.startswith(";")}

    return acession_numbers

def remove_mislabelled(gb, acession_numbers):
    """Remove mislabelled sequences from genbank file

    This function removes from the genbank file all
    sequences found by Sativa algorithm.

    Parameters
    -------------
    gb : io.TextIOWrapper
        Genbank file

```

```
    acession_numbers : list
        List of acession numbers to be removed

    """

    gb_all_data = SeqIO.index(gb.name, "genbank")

    base = os.path.basename(gb.name)
    name = os.path.splitext(base)[0]

    gb_m_name = "{}_m.gb".format(name)
    with open(gb_m_name, "w") as gb_c:
        for acession in tqdm(gb_all_data, desc="Removing mislabelled from genbank"):
            if not acession in acession_numbers:
                print(gb_all_data.get_raw(acession).decode(),
                          file=gb_c, end="")


if __name__ == "__main__":
    args = getArguments()
    acession_numbers = parse_mis(args.mis)
    remove_mislabelled(args.gb, acession_numbers)
```

Source Code A.6: Remove from the genbank file mislabelled sequences identified by the sativa algorithm.

```
#!/usr/bin/env python3
import os
import argparse
import datetime
import subprocess
from ete3 import PhyloTree, TreeStyle

"""
Build Phylogenetic Tree using raxmlHPC-PTHREADS-SSE3

From the alignment in fasta format and tree in newick format,
a pdf with the tree and alignment (side by side) will be generated.
"""

RAXML = ("raxmlHPC-PTHREADS-SSE3 -f a -m GTRGAMMA"
        " -n {output_file} -p 765 -s {input_file}"
        " -T 10 -x 498 -N 100")

CHOICES_F = ["pdf", "svg", "png", "jpg"]

```

```
21 def getArguments():
22     """Get arguments from terminal
23 
24     This function gets arguments from terminal via argparse
25 
26     Returns
27     -------------
28     arguments: Namespace
29         Namespace object with all arguments
30     """
31 
32     parser = argparse.ArgumentParser(
33         description="Build the Phylogenetic Tree and save it in a file.")
34     parser.add_argument("aln", type=argparse.FileType("r"),
35                         help="alignment file")
36     parser.add_argument("-f", "--format", nargs="?", const="jpg", default="jpg",
37                         choices=CHOICES_F, type = lambda s : s.lower(),
38                         help="Format to save the phylogenetic tree, default: jpg")
39     parser.add_argument('-s', '--show', action='store_true',
40                         help='Show ETE tree Browser')
41 
42     return parser.parse_args()
43 
44 def raxml(aln, basename):
45     """Build a phylogenetic tree based on alignment provided
46     using raxmlHPC-PTHREADS-SSE3
47 
48     This function executes RAxML program the create a
49     phylogenetic tree using GTRGAMMA substitution model
50 
51     Parameters
52     -------------
53     aln: io.TextIOWrapper
54         Alignment file
55     basename: string
56         Basename of the original alignment file
57     """
58 
59     print("Building tree ...")
60     output_file = "{BASENAME}.raxml".format(BASENAME=basename)
61 
62     subprocess.call(RAXML.format(output_file=output_file,
63                                  input_file=aln.name),
64                                  shell=True)
65     print("RAxML DONE!")
66 
```

```
67  def build_tree(aln, tree, basename, show, output_format):
68      """Build phylogenetic tree from files
69
70      This function creates a file with the phylogenetic tree and alignment
71      from the fasta multiple alignment file and the tree in newick format.
72
73      Parameters
74      -------------
75      aln: string
76          Alignment string in fasta format
77      tree: string
78          Tree string in newick format
79      basename: string
80          Basename of the original alignment file
81      show: boolean
82          Show ETE tree browser (yes/no)
83      output_format: string
84          Format of the output
85      """
86
87      if tree[-1] != ";":
88          genetree = PhyloTree("{};".format(tree))
89      else:
90          genetree = PhyloTree(tree)
91
92      ts = TreeStyle()
93      ts.show_leaf_name = False
94
95      new_tree = "{BASENAME}_Tree.{FORMAT}".format(
96                                  BASENAME=basename,
97                                  FORMAT=output_format)
98      new_tree_aln = "{BASENAME}_Tree_aln.{FORMAT}".format(
99                                  BASENAME=basename,
100                                 FORMAT=output_format)
101
102     if show:
103         genetree.render(new_tree, tree_style=ts)
104         genetree.link_to_alignment(aln)
105         genetree.render(new_tree_aln, tree_style=ts)
106         genetree.show(tree_style=ts)
107     else:
108         genetree.render(new_tree, tree_style=ts)
109         genetree.link_to_alignment(aln)
110         genetree.render(new_tree_aln, tree_style=ts)
111
112 if __name__ == "__main__":
```

```
    args = getArguments()

    basename = os.path.splitext(os.path.basename(
                               args.aln.name))[0]

    raxml(args.aln, basename)

    tree_file = "RAxML_bestTree.{BASENAME}.raxml".format(
                                  BASENAME=basename)

    aln = args.aln.read()
    with open(tree_file) as tree_file:
        tree = tree_file.read()

    build_tree(aln, tree, basename, args.show, args.format)
```

Source Code A.7: Build phylogenetic tree from a multiple alignment file.

```

## THe script searches for barcodes in forward and reverse reads
## that are not at the start of the read, but instead it looks for the bcs
## in the first 30 bases (this can be changed by changing the variable 'search_until'.

import sys
import gzip
from Bio import SeqIO
import time

def find_bcs(readpair, sample_data, search_until):

# print("checking: \n%s\n%s" %(readpair[1],readpair[5]))


    #Try forward orientation, i.e. foward barcode in forwared read and reverse barcode in reverse read
    for sample in sample_data:

        startindex = -1;
        endindex = -1;
        forw = sample_data[sample]['bcs'][0].upper()
        reve = sample_data[sample]['bcs'][1].upper()
# print "trying: %s\t%s" %(forw,reve)
        if forw in readpair[1][:search_until]:
            startindex = readpair[1].index(forw)
```

```
# print "found forward %s -> %s" %(forw, startindex)
          if reve in readpair[5][:search_until]:
              endindex = readpair[5].index(reve)
# print "found reverse %s -> %s" %(reve,endindex)
              break

    if startindex >= 0 and endindex >= 0:
#  print "forward assigned to sample: %s\n" %sample
        readpair[1] = readpair[1][startindex+len(forw):]
        readpair[3] = readpair[3][startindex+len(forw):]
        readpair[5] = readpair[5][endindex+len(reve):]
        readpair[7] = readpair[7][endindex+len(reve):]
        sample_data[sample]['seqs']['R1'].extend(readpair[:4])
        sample_data[sample]['seqs']['R2'].extend(readpair[4:])

    #  print "\n%s\n%s" %(readpair[1],readpair[5])
        return

    else:

    #Try reverse orientation, i.e. foward barcode in reverse read and reverse barcode
        ↪ in forward read
    # print "try reverse\n%s\n%s" %(readpair[1],readpair[5])
        for sample in sample_data:
               startindex = -1;
               endindex = -1;
               #assign barcodes in opposite order
               forw = sample_data[sample]['bcs'][1].upper()
               reve = sample_data[sample]['bcs'][0].upper()
    #  print "trying: %s\t%s" %(forw,reve)
               if forw in readpair[1][:search_until]:
                      startindex = readpair[1].index(forw)
    #  print "found forward %s -> %s" %(forw, startindex)
                      if reve in readpair[5][:search_until]:
                             endindex = readpair[5].index(reve)
    #  print "found reverse %s -> %s" %(reve,endindex)
                             break

    if startindex >= 0 and endindex >= 0:
# print "reverse assigned to sample: %s\n" %sample
        readpair[1] = readpair[1][startindex+len(forw):]
        readpair[3] = readpair[3][startindex+len(forw):]
        readpair[5] = readpair[5][endindex+len(reve):]
        readpair[7] = readpair[7][endindex+len(reve):]
        sample_data[sample]['seqs']['R1'].extend(readpair[:4])
        sample_data[sample]['seqs']['R2'].extend(readpair[4:])
```

```
#  print "\n%s\n%s" %(readpair[1],readpair[5])
        return
    else:
    # print "no proper hit\n"
        return readpair
#  sample_data['invalid']['R1'].extend(readpair[:4])
#  sample_data['invalid']['R2'].extend(readpair[4:])

# print "\n%s\n%s" %(readpair[1],readpair[5])


def touch_files():

    for sample in sample_data:
        fh1 = open(target+'/'+sample+'.R1.fastq','w')
        fh2 = open(target+'/'+sample+'.R2.fastq','w')
    fh1 = open(target+'/invalid.R1.fastq','w')
    fh2 = open(target+'/invalid.R2.fastq','w')

def write_out(reads=0):

    for sample in sorted(sample_data):
        if len(sample_data[sample]['seqs']['R1']) > 0:
            fh1 = open(target+'/'+sample+'.R1.fastq','a')
            fh2 = open(target+'/'+sample+'.R2.fastq','a')
    # for seq in sample_data[sample]['seqs']:
    #  fh.write(seq+'\n')
            for i in range(len(sample_data[sample]['seqs']['R1'])):
                fh1.write(sample_data[sample]['seqs']['R1'][i]+"\n")
                fh2.write(sample_data[sample]['seqs']['R2'][i]+"\n")
            fh1.close()
            fh2.close()
            sample_data[sample]['count'] += len(sample_data[sample]['seqs']['R1'])
            sample_data[sample]['seqs']['R1'] = []
            sample_data[sample]['seqs']['R2'] = []
            if reads:
                print("%s\t%i read pairs (%.2f %%)" %(sample, sample_data[sample]['
                    ↪ count']/4, (float(sample_data[sample]['count'])/4)/reads*100))
# else:
        # print "no valid reads found for sample '%s'" %sample


    if len(invalid_recs['R1']) > 0:
        fh1 = open(target+'/invalid.R1.fastq','a')
        fh2 = open(target+'/invalid.R2.fastq','a')
        for i in range(len(invalid_recs['R1'])):
```

```
116             fh1.write(invalid_recs['R1'][i]+"\n")
117             fh2.write(invalid_recs['R2'][i]+"\n")
118         invalid_recs['count'] += len(invalid_recs['R1'])
119         invalid_recs['R1'] = []
120         invalid_recs['R2'] = []
121 
122         fh1.close()
123         fh2.close()
124         if reads:
125             print("\ninvalid\t%i read pairs (%.2f %%)\n" %(invalid_recs['count']/4, (
                    ↪ float(invalid_recs['count'])/4)/reads*100))
126 
127     if reads:
128         print("total number of read pairs processed: %i" %reads)
129 
130 def process_pairs(f1, f2, sample_data, invalid_recs, search_until):
131     """Interleaves two (open) fastq files.
132     """
133     count = 0
134     touch_files()
135     while True:
136         lines = []
137         line = f1.readline()
138         if line.strip() == "":
139             break
140         lines.append(line.strip())
141 
142         for i in range(3):
143             lines.append(f1.readline().strip())
144 
145         for i in range(4):
146             lines.append(f2.readline().strip())
147 
148         temp = find_bcs(lines, sample_data, search_until)
149         if temp:
150             invalid_recs['R1'].extend(temp[:4])
151             invalid_recs['R2'].extend(temp[4:])
152         count+=1
153         if (count % 100000) == 1:
154             print("["+time.strftime("%c")+"] - %i read pairs processed" %(count/2*2))
155             write_out(0)
156     return count
157 
158 
159 
160 
```

```
search_until = 30

if not len(sys.argv) == 5:
        print("Expecting 4 arguments\n")
        sys.exit()


print(sys.argv[1])
file1 = sys.argv[2]
file2 = sys.argv[3]
target = sys.argv[4]

print(sys.version)

fh = open(sys.argv[1],'r')
sample_data = {}

for l in fh:
    print(l)
    cols = l.strip().split("\t")
    sample = cols[1]
    bcs = cols[2].split(":")
    sample_data[sample] = {'count': 0, 'bcs':[], 'seqs':{ 'R1': [], 'R2': []}}
    sample_data[sample]['bcs'] = bcs


readcount = 0
invalid_recs = {'count':0, 'R1':[], 'R2':[]}

if file1[-2:] == "gz":
    import gzip
    with gzip.open(file1, 'rt') as f1:
        with gzip.open(file2, 'rt') as f2:
            readcount = process_pairs(f1, f2, sample_data, invalid_recs, search_until)
else:
    with open(file1) as f1:
        with open(file2) as f2:
            readcount = process_pairs(f1, f2, sample_data, invalid_recs, search_until)
f1.close()
f2.close()

write_out(readcount)
```

Source Code A.8: FASTQ demultiplexer developed by the Evolutionary and Environmental Genomics Group at the University of Hull (http://www.evohull.org/).

```
#!/usr/bin/env python3
import os
import sys
import csv
import glob
import shutil
import pathlib
import argparse
import datetime
import subprocess
from pathlib import Path

"""Demultiplex fastq files in a folder.

For each fastq file inside the folder,
collect the information from the table and
demultiplex it to new files.
"""

PATH_RESULT = "{}_Demultiplexed".format(datetime.datetime.now().strftime("%d%m%Y_%H%M
    ↪ %S"))

LOG = "{}/Demultiplex_error.log".format(PATH_RESULT)

DEMULTIPLEXING = "python demultiplex_obi_Sep_2017.py {TSV} {R1} {R2} {folder_to_save}
    ↪ "

def getArguments():
    """Get arguments from terminal

    This function gets arguments from terminal via argparse

    Returns
    -------------
    arguments: Namespace
        Namespace object with all arguments
    """

    parser = argparse.ArgumentParser(
        description="Demultiplex FASTQ files"\
                    " using the information from the table")
    parser.add_argument("folder_fastq", type=str,
                        help="A folder with FASTQ files")
    parser.add_argument("folder_tables", type=str,
                        help="A folder with CSV/TSV table files")
```

```
44
45      return parser.parse_args()
46
47  def demultiplexing(fastq_file, folder_tables):
48      """Demultiplex FASTQ file.
49
50      For each pair R1 R2, this function uses the table
51      provided to demultiplex the fastq files.
52
53      fastq_file: str
54          fastq file name
55
56      folder_tables: str
57          Path to folder with csv/tsv files
58      """
59
60      file_name = os.path.basename(fastq_file)
61      base_name = file_name.split("_")[0]
62
63      table_name = "{dirname}/{name}*".format(dirname=folder_tables,
64                                              name=base_name)
65
66      table = glob.glob(table_name)
67
68      if not table:
69          raise Exception("{table} table not found".format(
70                          table=table))
71
72      table = table[0]
73
74      path_tsv = "{dirname}/TSV/".format(dirname=PATH_RESULT)
75      Path(path_tsv).mkdir(parents=True, exist_ok=True)
76      tsv_file = "{path_tsv}{table_name}.tsv".format(
77                  path_tsv=path_tsv, table_name=base_name)
78
79      with open(tsv_file, "w+") as output_tsv:
80          with open(table) as input_table:
81              table = input_table.read()
82              sniffer = csv.Sniffer()
83              dialect = sniffer.sniff(table)
84
85              table = table.rstrip().lstrip().split("\n")
86              csv.writer(output_tsv, delimiter='\t').writerows(
87                  csv.reader(table, delimiter=dialect.delimiter))
88
89
```

```
90      path_demultiplexed = "{dirname}/{name}".format(
91                           dirname=PATH_RESULT, name=base_name)
92
93      Path(path_demultiplexed).mkdir(parents=True, exist_ok=True)
94
95      subprocess.run(DEMULTIPLEXING.format(TSV=tsv_file,
96                                           R1=fastq_file,
97                                           R2=fastq_file.replace("R1", "R2"),
98                                           folder_to_save=path_demultiplexed),
99                                           shell=True,
100                                          stderr=sys.stderr,
101                                          stdout=sys.stdout)
102
103
104 if __name__ == "__main__":
105     args = getArguments()
106
107     Path(PATH_RESULT).mkdir(parents=True, exist_ok=True)
108
109     fastq_files = []
110     fastq_files.extend(glob.glob("{folder}*R1*.fastq.gz".format(
111         folder=args.folder_fastq)))
112
113     for fastq_file in fastq_files:
114         try:
115             demultiplexing(fastq_file, args.folder_tables)
116         except Exception as e:
117             with open(LOG, "a") as log:
118                 print("{}\n".format(e), file=log)
119
120     raw_reads = "{dirname}/raw_reads".format(dirname=PATH_RESULT)
121     Path(raw_reads).mkdir(parents=True, exist_ok=True)
122
123     print("\nMOVING to raw reads folder")
124     fastq_demultiplexed = []
125     fastq_demultiplexed.extend(sorted(glob.glob("{dirname}/**/*.fastq".format(
126         dirname=PATH_RESULT), recursive=True)))
127
128     fastq_files = {}
129     for fastq_file in fastq_demultiplexed:
130         file_name = os.path.basename(fastq_file)
131         if file_name not in fastq_files:
132             fastq_files[file_name] = [fastq_file]
133         else:
134             fastq_files[file_name].append(fastq_file)
135
```

```
    mv = "mv {fastq_file} {raw_reads}"
    cat = "cat {files} > {raw_reads}/{file_name}"
    for file_name, fastq_file in fastq_files.items():
        if len(fastq_files[file_name]) == 1:
            subprocess.run(mv.format(fastq_file=fastq_files[file_name][0],
                                     raw_reads=raw_reads),
                                     shell=True, stderr=sys.stderr,
                                     stdout=sys.stdout)
        else:
            subprocess.run(cat.format(files=" ".join(fastq_files[file_name]),
                                     raw_reads=raw_reads,
                                     file_name=file_name),
                                     shell=True, stderr=sys.stderr,
                                     stdout=sys.stdout)


    print("REMOVING invalid files and temporary folders")
    rm = "rm {raw_reads}/invalid*"
    subprocess.run(rm.format(raw_reads=raw_reads), shell=True,
                          stderr=sys.stderr, stdout=sys.stdout)

    folders = glob.glob("{PATH_RESULT}/*".format(PATH_RESULT=PATH_RESULT))
    for folder in folders:
        path = pathlib.PurePath(folder)
        if path.name != "raw_reads" \
                and path.name != "TSV":
            shutil.rmtree(folder)

    print("ZIPPING files")
    gzip = "gzip {raw_reads}/*.fastq"
    subprocess.run(gzip.format(raw_reads=raw_reads), shell=True,
                          stderr=sys.stderr, stdout=sys.stdout)
```

Source Code A.9: Demultiplexer automatization for all files inside a folder.

```
#!/usr/bin/env python3
import os
import glob
import argparse
import datetime
import pandas as pd
from Bio.Seq import Seq
from pathlib import Path
from multiprocessing.dummy import Pool
from subprocess import run, check_output
```

```
11 from multiprocessing import Process, cpu_count
12
13 """Remove adapters from fasta(q) files
14
15 From a list of files inside a folder,
16 it removes adapters from the 5', 3'
17 or both sides.
18 """
19
20 DATETIME = datetime.datetime.now().strftime("%d%m%Y_%H%M%S")
21 PATH_RESULT = "{DATETIME}_Remove_Adapter".format(DATETIME=DATETIME)
22 CUTADAPT = "cutadapt {SIDE_ADAPTER}-o {OUTPUT} {INPUT}"
23
24 CHOICES_S = ["a", "b", "g"]
25
26 COMMANDS = {}
27
28 def getArguments():
29     """Get arguments from terminal
30
31     This function gets arguments from terminal via argparse
32
33     Returns
34     -------------
35     arguments: Namespace
36         Namespace object with all arguments
37     """
38
39     num_threads = cpu_count() - 2
40     if num_threads < 1:
41         num_threads = 1
42
43     parser = argparse.ArgumentParser(
44         description="Remove adapters from fasta(q) files")
45     parser.add_argument("folder_fastaq", type=str,
46                 help="A folder with fasta(q) files")
47     parser.add_argument("folder_demultiplex", type=str,
48                 help="A folder with tables with" \
49                 " information about demultiplex adapters. "\
50                 "Table files must have an identical name as fasta(q)")
51     parser.add_argument("-n", "--num_threads", nargs="?", type = int,
52                         const=num_threads, default=num_threads,
53                  help="Number of threads to be executed in parallel.")
54     parser.add_argument("-s", "--side", nargs="?", const="a",
55                  default="a", choices=CHOICES_S,
56                  type = lambda s : s.lower(),
```

```
                    help="Side of the sequence to cut the adapter"\
                    ", default: a (3' side) ")

    return parser.parse_args()

def parse_demultiplex_files(demultiplex_files):
    """Parse demultiplex files to get adapters.

    For each demultiplex table, parse the id of
    the file and the adapter in normal and reverse
    direction

    demultiplex_files: list
        List with tables demultiplex info

    Returns
    -------------
    demultiplex: dataframe
        pandas dataframe with demultiplex adapters info
    """

    demultiplex = pd.DataFrame()

    for demultiplex_file in demultiplex_files:
        demultiplex = demultiplex.append(pd.read_csv("{DEMULTIPLEX_FILE}".format(
                                    DEMULTIPLEX_FILE = demultiplex_file),
                                    sep = None, header = None,
                                    engine = 'python')
                                    )

    return demultiplex

def remove_adapters(fastaq_files, demultiplex, side, num_threads):
    """Remove adapters out of the fasta(q) files.

    From a list of fasta(q) files, this function
    removes adapters of the 5' end, 3' end
    or both sides.

    fastaq_files: list
        List of fasta(q) files
    demultiplex: dataframe
        Pandas dataframe with demultiplex info
    side: str
        Side of the sequence to remove adapters
    num_threads: int
```

```
        Number of threads for the multithreading
    """

    for fastaq_file in fastaq_files:
        file_name = Path(fastaq_file).stem
        base_name = file_name.split(".")[0]

        adapters = set(demultiplex.loc[demultiplex.iloc[:,1] == base_name,
                           2].values.flatten().tolist())

        side_adapter = ""
        for adapter in adapters:
            adapter = adapter.split(":")[0]

            if side == "a":
                side_adapter += "-{SIDE} {ADAPTER} ".format(
                    SIDE = side,
                    ADAPTER = str(Seq(adapter).reverse_complement()))

                continue

            side_adapter += "-{SIDE} {ADAPTER} ".format(
                    SIDE = side,
                    ADAPTER = adapter)

        COMMANDS["{BASE_NAME} R1".format(
                BASE_NAME = base_name)] = CUTADAPT.format(
                SIDE_ADAPTER = side_adapter,
                OUTPUT = "{PATH_RESULT}/Sequences/{BASE_NAME}"\
                    "_L001_R1_001.fastq.gz".format(
                    PATH_RESULT = PATH_RESULT,
                    BASE_NAME = base_name),
                INPUT = fastaq_file)

        COMMANDS["{BASE_NAME} R2".format(
                BASE_NAME = base_name)] = CUTADAPT.format(
                SIDE_ADAPTER = side_adapter,
                OUTPUT = "{PATH_RESULT}/Sequences/{BASE_NAME}"\
                    "_L001_R2_001.fastq.gz".format(
                    PATH_RESULT = PATH_RESULT,
                    BASE_NAME = base_name),
                INPUT = fastaq_file.replace("R1", "R2"))

    pool = Pool(num_threads)
    for returncode in pool.imap(execute_command, COMMANDS):
        if returncode:
```

```
            print("command failed: {}".format(returncode))


def execute_command(base_name):
    """Execute command

    This function executes command on terminal.

    base_name: str
        base_name of the fastq file
    """

    print(COMMANDS[base_name])
    with open("{PATH_RESULT}/Logs/{BASE_NAME}.log".format(
                PATH_RESULT = PATH_RESULT,
                BASE_NAME = base_name), "a") as log_file:
        run(COMMANDS[base_name], shell=True, stderr=log_file, stdout=log_file)

if __name__ == "__main__":
    args = getArguments()

    # Reading fastq files
    fastaq_files = sorted(glob.glob("{folder}/*R1*.fast*".format(
        folder=args.folder_fastaq)))
    demultiplex_files = sorted(glob.glob("{folder}/*.tsv".format(
        folder=args.folder_demultiplex)))
    side = args.side
    num_threads = args.num_threads

    # Parsing demultiplex files info
    demultiplex = parse_demultiplex_files(demultiplex_files)

    # Creating folders
    Path(PATH_RESULT).mkdir(parents=True, exist_ok=True)
    Path("{PATH_RESULT}/Logs".format(PATH_RESULT = PATH_RESULT)).mkdir(
            parents=True, exist_ok=True)
    Path("{PATH_RESULT}/Sequences".format(PATH_RESULT = PATH_RESULT)).mkdir(
            parents=True, exist_ok=True)

    # Removing adapters
    remove_adapters(fastaq_files, demultiplex, side, num_threads)
```

Source Code A.10: Remove adapter from the 3' end of the read fragment.

```
#!/usr/bin/env python3
```

```
import os
import csv
import pathlib
import argparse
import datetime
import subprocess
from Bio import SeqIO
from tqdm import tqdm
from Bio import Entrez
from ete3 import NCBITaxa

"""
Convert genbank format to fasta format to be used
in the pipelines execution.

Additionally to the conversion,
a taxonomy table with accession number +
superkingdom,phylum,class,order,family,genus,species
is created when converting for Anacapa pipeline
and taxid table is created when converting
for SEQme pipeline.

"""

CHOICES_S = ["ncbi", "taxit", "taxdump"]

CHOICES_R = ["superkingdom", "phylum", "class",
           "order", "family", "genus",
           "species", "all"]

CHOICES_P = ["anacapa", "barque", "metabeat",
           "mifish", "seqme", "none", "all"]

PATH_MAIN = "{}_genbank2Fasta/".format(
              datetime.datetime.now().strftime("%d%m%Y_%H%M%S"))

LOG = "{}genbank2Fasta_error.log".format(PATH_MAIN)

PIPELINES_OUTPUT_FASTA = {"none": ">{species_}_{id}\n{seq}",
                      "anacapa": ">{id}\n{seq}",
                      "barque": ">{phylum}_{species_}\n{seq}",
                      "metabeat": ">{id}|{taxonid}|{species}\n{seq}",
                      "mifish": ">gb|{id}|{species_}\n{seq}",
                      "seqme": (">{id}\t{superkingdom};{kingdom};"
                                "{phylum};{class};{order};"
                                "{family};{genus};{species}\n{seq}")}
```

```
48
49  PIPELINES_OUTPUT_TAX = {"anacapa": ("{id}\t{superkingdom};{phylum};"
50                                      "{class};{order};{family};"
51                                      "{genus};{species}")}
52
53  LINEAGES = {}
54
55  TAXONOMIC_RANK = {"superkingdom": 0, "kingdom": 1, "phylum": 2,
56                    "class": 3, "order": 4, "family": 5,
57                    "genus": 6, "species": 7}
58  TAXONOMIC_HIERARCHY = {"kingdom": "superkingdom", "phylum": "kingdom",
59                    "class": "phylum", "order": "class", "family": "order",
60                    "genus": "family", "species": "genus"}
61
62  PATH_TO_TAXID = "{PATH_INFORMED}/TaxIDS.txt"
63  PATH_TO_TAXA = "{PATH_INFORMED}/Taxa.csv"
64  PATH_TO_DB = "{PATH_INFORMED}/ncbi_taxonomy.db"
65
66
67  def getArguments():
68      """Get arguments from terminal
69
70      This function gets arguments from terminal via argparse
71
72      Returns
73      -------------
74      arguments: Namespace
75          Namespace object with all arguments
76      """
77
78      parser = argparse.ArgumentParser(
79          description="Conversion from genbank to fasta format "\
80                      "to be used in the execution of the pipeline(s).")
81      parser.add_argument("gb", type=argparse.FileType("r"),
82                          help="genbank file (.gb)")
83      parser.add_argument('-sp', '--species_from_file', action='store_true',
84                          help="Should it use species"\
85                          " from file or download it from NCBI?")
86      parser.add_argument("-p", "--pipeline", nargs="*", default=["none"],
87                          choices=CHOICES_P, type = lambda s : s.lower(),
88                          help="Pipeline convertion format, default: none")
89      parser.add_argument("-r", "--rank", nargs="*", default=["superkingdom"],
90                          choices=CHOICES_R, type = lambda s : s.lower(),
91                          help="Taxonomic classification rank to be used"\
92                          " to separate the groups, default: superkingdom")
93      parser.add_argument("-s", "--source", nargs="?", const="taxdump",
```

```
                    default="taxdump", choices=CHOICES_S,
                    type = lambda s : s.lower(),
                    help="Source to be used to collect"\
                    " the info about the taxonomic rank, default: taxdump")
    parser.add_argument('-t', '--path_to_taxid_files', nargs='?', type = str,
                    const=PATH_MAIN, default=PATH_MAIN,
                    help="Path to taxit files.")

    return parser.parse_args()

def parse_taxID(gb):
    """Parse taxon ids from genbank file

    This function uses Biopython library to parse
    taxon ids and create a file with them.

    Parameters
    ------------
    gb: io.TextIOWrapper
        Genbank file

    """

    tax_ids = set()

    for r in tqdm(gb, desc="Reading sequences"):
        record = gb.get(r)
        taxonid = record.features[0].qualifiers["db_xref"][0].split(":")[1]
        tax_ids.add(taxonid)

    with open(PATH_TO_TAXID, "w") as out_taxids:
        out_taxids.write("\n".join(tax_ids))

def taxit():
    """Download a create taxonomic database using taxit

    This function executes taxit to download taxonomy database
    and creates a table with the taxonomic lineages.
    """

    print("Downloading taxit database ...")
    subprocess.call("taxit new_database {PATH_TO_DB} -p {PATH_DOWNLOAD}".format(
                    PATH_TO_DB=PATH_TO_DB,
                    PATH_DOWNLOAD=PATH_MAIN) , shell=True)

    print("Creating tax table ...")
```

```
140     subprocess.call("taxit taxtable {PATH_TO_DB}".format(PATH_TO_DB=PATH_TO_DB) +
141                     " -f {PATH_TO_TAXID}".format(PATH_TO_TAXID=PATH_TO_TAXID) +
142                     " -o {PATH_TO_TAXA}".format(PATH_TO_TAXA=PATH_TO_TAXA),
143                     shell=True)
144
145 def parse_taxa():
146     """Parse taxonomic information from Taxa.csv
147
148     This function opens Taxa.csv file to parse
149     tax id and lineage rank.
150
151     Returns
152     -------------
153     tax_rank_id: dict
154         Taxonomic rank and id
155     """
156
157     with open(PATH_TO_TAXA) as taxa_input:
158         taxa = csv.DictReader(taxa_input)
159         tax_rank_id = {row["tax_id"]:row for row in taxa}
160
161     return tax_rank_id
162
163 def get_tax_lineage(taxonid, source, tax_rank_id={}):
164     """Return taxonomy lineage information
165
166     This function uses either Biopython library to connect
167     NCBI database and search for taxonomy information
168     or searches the information locally by using ete3 taxdump
169     file or taxit program to create sql version of it.
170
171     Parameters
172     -------------
173     taxonid : string
174         Taxonomic id of the species
175     source : string
176         Source to be used to collect the info about the taxonid
177     tax_rank_id: dict
178         Taxonomic rank and id
179
180     Returns
181     -------------
182     lineage: dict
183         Species lineage
184
185     """
```

```
186
187     if taxonid not in LINEAGES:
188         if source == "taxdump":
189             ncbi_taxdump = NCBITaxa()
190             lineage_ids = ncbi_taxdump.get_lineage(taxonid)
191             ranks = ncbi_taxdump.get_rank(lineage_ids)
192             names = ncbi_taxdump.get_taxid_translator(lineage_ids)
193             lineage = {ranks[i]:names[i] for i in lineage_ids}
194
195             LINEAGES[taxonid] = lineage
196             return LINEAGES[taxonid]
197
198         if source == "taxit":
199             lineage = {level:tax_rank_id[tax_rank_id[
200                         taxonid][level]]["tax_name"]
201                         for level in CHOICES_R[:-1]}
202
203             LINEAGES[taxonid] = lineage
204             return LINEAGES[taxonid]
205
206
207         while True:
208             data = ""
209             try:
210                 Entrez.email = "Your.Name.Here@example.org"
211                 handle = Entrez.efetch(id = taxonid, db = "taxonomy", retmode = "xml")
212                 data = Entrez.read(handle)
213                 handle.close()
214             except Exception as e:
215                 with open(LOG, "a") as log:
216                     print("Error when searching information about {}".format(taxonid),
217                         file=log)
218
219             if data:
220                 break
221
222         lineage = {d["Rank"]:d["ScientificName"] for d in data[0]["LineageEx"]}
223         lineage[data[0]["Rank"]] = data[0]["ScientificName"]
224         LINEAGES[taxonid] = lineage
225
226
227     return LINEAGES[taxonid]
228
229 def read_sequences(gb, pipeline, rank, source, species_from_file, tax_rank_id={}):
230     """Read the genbank file and parse the sequences
231     based on the taxonomic rank
```

```
232
233     This function uses Biopython library to scan the genbank file
234     and parse the sequences based on the taxonomic rank.
235
236     Parameters
237     -------------
238     gb : io.TextIOWrapper
239         A genbank file
240     pipeline: string
241         The pipeline format of the fasta format
242     rank: string
243         Taxonomic rank
244     source : string
245         Source to be used to collect the info about the taxonid
246     species_from_file: bool
247         Indicate if species name from file should be used
248     tax_rank_id: dict
249         Taxonomic rank and id
250
251     Returns
252     -------------
253     sequences: dictionary
254         A dictionary with key representing ranks and values
255         representing sequences in fasta format with species name
256         as the header of the sequence
257     tax_tables: dictionary
258         A dictionary with key representing ranks and values
259         representing taxonomic tables if the pipeline
260         is either anacapa or seqme
261
262     """
263
264     sequences = {}
265     tax_tables = {}
266     sequence_info = {}
267     taxid_table = {}
268
269     for r in tqdm(gb, desc="Reading sequences"):
270         record = gb.get(r)
271         tax = []
272
273         sequence_info["id"] = record.id
274         sequence_info["seq"] = record.seq
275
276         for feature in record.features:
277             if feature.type == "source" and \
```

```
278                 "taxon" in feature.qualifiers["db_xref"][0]:
279                 taxonid = feature.qualifiers["db_xref"][0].split(":")[1]
280                 sequence_info["taxonid"] = taxonid
281 
282         lineage = get_tax_lineage(taxonid, source, tax_rank_id)
283 
284         if species_from_file:
285             lineage["species"] = record.features[0].qualifiers[
286                                     "organism"][0].lstrip().rstrip()
287         else:
288             if "species" not in lineage:
289                 lineage["species"] = record.features[0].qualifiers[
290                                     "organism"][0].lstrip().rstrip()
291 
292         lineage["species_"] = lineage["species"].replace(" ", "_")
293         sequence_info.update(lineage)
294 
295         if pipeline in "anacapa":
296                 tax = PIPELINES_OUTPUT_TAX[pipeline].format(**sequence_info)
297         elif pipeline == "seqme":
298 
299             for lin in lineage:
300                 if lin not in TAXONOMIC_RANK:
301                     continue
302 
303                 if lineage[rank] not in taxid_table:
304                     taxid_table[lineage[rank]] = [
305                            {"Eukaryota":"0*Eukaryota*-1*0*superkingdom"},
306                            {"Eukaryota": 0}, 1
307                        ]
308                     tax.append(taxid_table[lineage[rank]][0]["Eukaryota"])
309 
310                 if lineage[lin] not in taxid_table[lineage[rank]][0]:
311                     taxid_table[lineage[rank]][1][
312                            lineage[lin]] = taxid_table[lineage[rank]][2]
313                     taxid_table[lineage[rank]][0][
314                            lineage[lin]] = "{number}*{tax}*{b_tax}"\
315                                    "*{taxid}*{lineage}".format(
316                                    number=taxid_table[lineage[rank]][2],
317                                    tax=lineage[lin],
318                                    b_tax=taxid_table[lineage[rank]][1][lineage[
319                                            TAXONOMIC_HIERARCHY[lin]]],
320                                    taxid=TAXONOMIC_RANK[lin],
321                                    lineage=lin)
322 
323                     taxid_table[lineage[rank]][2] += 1
```

```
                    tax.append(taxid_table[lineage[rank]][0][lineage[lin]])

            tax = "\n".join(tax)

        try:
            sequence = PIPELINES_OUTPUT_FASTA[pipeline].format(**sequence_info)

            if lineage[rank] not in sequences:
                sequences[lineage[rank]] = [sequence]
                if tax:
                    tax_tables[lineage[rank]] = [tax]
            else:
                sequences[lineage[rank]].append(sequence)
                if tax:
                    tax_tables[lineage[rank]].append(tax)

        except:
            with open(LOG, "a") as log:
                print("\nRank '{}' not found for organism '{}', taxonid '{}'".format(
                    rank, lineage["species"], taxonid), file=log)

    return sequences, tax_tables

def save_fasta(sequences, pipeline, rank):
    """Save sequences to file

    This function saves each group of sequence
    in the dictionary to fasta file format
    based on rank grouping.

    sequences : dictionary
        A dictionary with sequences
    pipeline: string
        The pipeline format of the fasta format
    rank: string
        Taxonomic rank

    """

    path_fasta = "{}{}/{}".format(PATH_MAIN, pipeline, rank)
    pathlib.Path(path_fasta).mkdir(parents=True, exist_ok=True)

    for seq in tqdm(sequences, desc="Saving FASTA"):
        with open("{PATH_FASTA}/{FASTA_NAME}.fasta".format(
                                    PATH_FASTA=path_fasta,
                                    FASTA_NAME=seq.replace(" ", "_")
```

```
                                ), "w") as fasta_file:
            fasta_file.write("\n".join(sequences[seq]))

def save_tax_tables(tax_tables, pipeline, rank):
    """Save tax tables to file

    This function saves each group of tax table
    in the dictionary to a text file
    based on rank grouping. Tax tables are
    only created if the pipeline variable
    is equal to anacapa or seqme.

    tax_tables : dictionary
        A dictionary with tax tables
    pipeline: string
        The pipeline format of the fasta format
    rank: string
        Taxonomic rank

    """

    path_tax_tables = "{}{}/{}".format(PATH_MAIN, pipeline, rank)
    pathlib.Path(path_tax_tables).mkdir(parents=True, exist_ok=True)

    for tax_table in tqdm(tax_tables, desc="Saving tax table"):
        with open("{PATH_TAX_TABLE}/{TAX_TABLE_NAME}.txt".format(
                                PATH_TAX_TABLE=path_tax_tables,
                                TAX_TABLE_NAME=tax_table.replace(
                                                " ", "_")
                                ), "w") as tax_table_file:
            tax_table_file.write("\n".join(tax_tables[tax_table]))


if __name__ == "__main__":
    args = getArguments()

    pathlib.Path(PATH_MAIN).mkdir(parents=True, exist_ok=True)
    gb = SeqIO.index(args.gb.name, "genbank")
    with open(LOG, "w"): pass

    tax_rank_id = {}
    if args.source == "taxit":

        PATH_TO_TAXID = "{PATH_INFORMED}/TaxIDS.txt".format(
                        PATH_INFORMED=args.path_to_taxid_files)
        PATH_TO_TAXA = "{PATH_INFORMED}/Taxa.csv".format(
```

```
                        PATH_INFORMED=args.path_to_taxid_files)
        PATH_TO_DB = "{PATH_INFORMED}/ncbi_taxonomy.db".format(
                        PATH_INFORMED=args.path_to_taxid_files)

        if not os.path.isfile(PATH_TO_TAXA):
            parse_taxID(gb)
            taxit()

        tax_rank_id = parse_taxa()

    ranks = args.rank
    if "all" in args.rank:
        ranks = CHOICES_R[:-1]

    pipelines = args.pipeline
    if "all" in args.pipeline:
        pipelines = CHOICES_P[:-1]

    for pipeline in pipelines:
        print("Pipeline: {}".format(pipeline))
        for rank in ranks:
            print("Rank: {}".format(rank))

            sequences, tax_tables = read_sequences(gb, pipeline,
                                                   rank, args.source,
                                                   args.species_from_file,
                                                   tax_rank_id)

            save_fasta(sequences, pipeline, rank)
            if tax_tables:
                save_tax_tables(tax_tables, pipeline, rank)
```

Source Code A.11: Convert genbank format to FASTA format and create a taxonomic table if pipeline is either Anacapa or SEQme.

```
#!/bin/bash

# Modify the following parameter values according to your experiment
# Do not modify the parameter names or remove parameters
# Do not add spaces around the equal (=) sign
# It is a good idea to try to run Barque with different parameters

# Global parameters
NCPUS=10 # Number of CPUs to use. A lot of the steps are parallelized (int, 1+)
PRIMER_FILE="02_info/primers.csv" # File with PCR primers information
```

```
11
12 # Skip data preparation and rerun only from vsearchp
13 SKIP_DATA_PREP=0 # 1 to skip data preparation steps, 0 to run full pipeline (
       ↪ recommended)
14
15 # Filtering with Trimmomatic
16 CROP_LENGTH=126 # Cut reads to this length after filtering. Just under amplicon
       ↪ length
17
18 # Merging reads with flash
19 MIN_OVERLAP=15 # Minimum number of overlapping nucleotides to merge reads (int, 1+)
20 MAX_OVERLAP=126 # Maximum number of overlapping nucleotides to merge reads (int, 1+)
21
22 # Extracting barcodes
23 MAX_PRIMER_DIFF=2 # Maximum number of differences allowed between primer and sequence
       ↪  (int, 0+)
24
25 # Running or skipping chimera detection
26 SKIP_CHIMERA_DETECTION=0 # 0 to search for chimeras (RECOMMENDED), 1 to skip chimera
       ↪ detection
27                          # or use already created chimera cleaned files
28
29 # vsearch
30 MAX_ACCEPTS=20 # Accept at most this number of sequences before stoping search (int,
       ↪ 1+)
31 MAX_REJECTS=20 # Reject at most this number of sequences before stoping search (int,
       ↪ 1+)
32 QUERY_COV=0.85 # At least that proportion of the sequence must match the database (
       ↪ float, 0-1)
33
34 # Filters
35 MIN_HIT_LENGTH=90 # Minimum vsearch hit length to keep in results (int, 1+)
36 MIN_HITS_SAMPLE=1 # Minimum number of hits a species must have in at least one sample
37                          # to keep in results (int, 1+)
38 # Non-annotated reads
39 NUM_NON_ANNOTATED_SEQ=1000 # Number of unique most-frequent non-annotated reads to
       ↪ keep (int, 1+)
40
41 # OTUs
42 SKIP_OTUS=1 # 1 to skip OTU creation, 0 to use it
43 MIN_SIZE_FOR_OTU=20 # Only unique reads with at least this coverage will be used for
       ↪ OTUs
```

Source Code A.12: Barque configuration file modified according to the project data.

```
import os
import glob
import datetime
```

```
initial_time = datetime.datetime.now()

os.chdir("/home/working/12S/")
os.getcwd()
```

```
!mkdir trimming
```

```
cd trimming
```

```
files = glob.glob("../../raw_reads/*.fastq.gz")
files = set([f.split("/")[-1].split("_")[0] for f in files])
files = sorted(files)

with open("Sample_accessions.tsv", "w") as sample_accessions:
    sample_accessions.write("SampleID\n" + "\n".join(files))
```

```
%%bash

for a in $(cat Sample_accessions.tsv | grep "SampleID" -v)
do
    R1=$(ls -1 ../../raw_reads/$a* | grep "_L001_R1_001.fastq")
    R2=$(ls -1 ../../raw_reads/$a* | grep "_L001_R2_001.fastq")

    echo -e "$a\tfastq\t$R1\t$R2\t18\t18"
done > Querymap.txt
```

```
%%bash

metaBEAT_global.py \
-Q Querymap.txt \
--trim_qual 20 \
--trim_minlength 90 \
--merge \
--product_length 106 \
--read_crop 110 \
--forward_only \
--length_filter 106 \
--length_deviation 0.2 \
-m 12S -n 10 -v \
-@ your_email@gmail.com &> log_trim
```

```
cd ../
```

```
!mkdir chimera_detection
```

```
cd chimera_detection
```

```
%%bash

#Write REFmap
for file in $(ls -1 ../../supplementary_data/reference_DBs/* | grep "
    ↪ reference_database.gb$")
do
     echo -e "$file\tgb"
done >> REFmap.txt
```

```
%%bash

metaBEAT_global.py \
-R REFmap.txt \
-f \
-@ your_email@gmail.com
```

```
%%bash


for a in $(cut -f 1 ../trimming/Querymap.txt)
do
    if [ -s ../trimming/$a/$a\_trimmed.fasta ]
    then
        echo -e "\n### Detecting chimeras in $a ###\n"
        mkdir $a
        cd $a
        vsearch --uchime_ref ../../trimming/$a/$a\_trimmed.fasta --db ../refs.fasta \
        --nonchimeras $a-nonchimeras.fasta --chimeras $a-chimeras.fasta &> log
        cd ..

    else
        echo -e "$a is empty"
    fi
done
```

```
cd ..
```

```
!mkdir non-chimeras
```

```
cd non-chimeras/
```

```
%%bash

#Write REFmap
for file in $(ls -1 ../../supplementary_data/reference_DBs/* | grep "
    ↪ reference_database.gb$")
```

```
do
      echo -e "$file\tgb"
done >> REFmap.txt
```

```
%%bash

#Querymap
for a in $(ls -l ../chimera_detection/ | grep "^d" | perl -ne 'chomp; @a=split(" ");
    ↪ print "$a[-1]\n"')
do
   if [ "$a" != "GLOBAL" ]
   then
      echo -e "$a-nc\tfasta\t../chimera_detection/$a/$a-nonchimeras.fasta"
   fi
done > Querymap.txt
```

```
%%bash

metaBEAT_global.py \
-Q Querymap.txt \
-R REFmap.txt \
--blast --min_ident 1 --min_ali_length 0.85 \
--cluster --clust_match 1 --clust_cov 3 \
-m 12S -n 10 \
-E -v \
-@ your_email@gmail.com \
-o metaBEAT_1.0 &> log_assignment
```

```
final_time = datetime.datetime.now()
total_time = final_time - initial_time

with open("Time.txt", "w") as total_time_file:
    total_time_file.write("Total time to run the pipeline: {TOTAL_TIME}\n".format(
                              TOTAL_TIME=total_time))
```

Source Code A.13: metaBEAT workflow jupyter notebook.

```
#!/usr/bin/env python3
import os
import sys
import glob
import argparse
import datetime
import numpy as np
import pandas as pd
from tqdm import tqdm
```

```
10 from pathlib import Path
11 from functools import partial
12 from multiprocessing.dummy import Pool
13 from multiprocessing import Process, cpu_count
14 from subprocess import run, check_output, DEVNULL, STDOUT
15
16 """SeqME pipeline for metabarcoding detection.
17
18 For each sample fastq pair of files:
19
20     .Joining paired-ends (fastq-join)
21     .Quality filtering (fastx_toolkit)
22     .Removing too short and too long sequences (Biopieces)
23     .Clustering (USEARCH v10.0.240)
24     .Creating an OTU table (USEARCH v10.0.240)
25     .Alpha diversity & normalization (USEARCH v10.0.240)
26     .Identifying OTUs by classifier (RDPTools)
27
28 """
29
30 PATH_CLASSIFIER = "Classifier/rRNAClassifier.properties"
31
32 DATETIME = datetime.datetime.now().strftime("%d%m%Y_%H%M%S")
33
34 PATH_RESULT = "{DATETIME}_SeqME".format(DATETIME=DATETIME)
35 PATH_FASTQ_JOINED = "{PATH_RESULT}/Fastq_Joined"
36 PATH_QUALITY_FILTERED = "{PATH_RESULT}/Fastq_Quality_Filtered"
37 PATH_FASTA = "{PATH_RESULT}/Fasta"
38 PATH_FASTA_REMOVED_SHORT_LONG_SEQ = "{PATH_RESULT}/Fasta_Removed_Short_Long_Seq"
39 PATH_UNIQUES = "{PATH_RESULT}/Fasta_Uniques"
40 PATH_OTUS_FASTA = "{PATH_RESULT}/Fasta_OTUs"
41 PATH_OTUS_TABLE = "{PATH_RESULT}/Table_OTUs"
42 PATH_OTUS_TABLE_NORMALIZE = "{PATH_RESULT}/Table_OTUs_Normalized"
43 PATH_ALPHA_DIVERSITY = "{PATH_RESULT}/Table_Alpha_Diversity"
44 PATH_OTU_IDENTIFIED = "{PATH_RESULT}/Table_OTUs_identified"
45
46 FASTQ_JOIN = "fastq-join -v ' ' -p 15 -m 15 {R1} {R2} -o {FASTQ_OUTPUT}"
47 FASTQ_QUALITY_FILTER = "fastq_quality_filter -i {FASTQ_INPUT}"\
48                        " -Q33 -q 20 -p 50 -o {FASTQ_OUTPUT}"
49 FASTQ_TO_FASTA = "fastq_to_fasta -i {FASTQ_INPUT} -o {FASTA_OUTPUT}"
50 FASTA_REMOVE_SHORT_LONG_SEQ = "read_fasta -i {FASTA_INPUT}" \
51                               " | grab -e 'SEQ_LEN >= 90'"\
52                               " | grab -e 'SEQ_LEN <= 150'"\
53                               " | write_fasta -x -o {FASTA_OUTPUT}"
54 FASTA_CLUSTER_UNIQUES = "usearch -fastx_uniques {FASTA_INPUT}"\
55           " -fastaout {FASTA_OUTPUT} -uc {UC_OUTPUT}"\
```

```
           " -sizeout -relabel Uniq"
FASTA_CLUSTER_OTU = "usearch -cluster_otus {FASTA_INPUT}"\
                    " -otus {FASTA_OUTPUT} -relabel Otu"
FASTA_CREATE_OTU_TABLE = "usearch -otutab {FASTA_INPUT}"\
                         " -otus {FASTA_OTU_INPUT} -otutabout"\
                         " {OTU_OUTPUT} -mapout {MAP_OUTPUT}"
OTU_NORMALIZE = "usearch -otutab_rare {OTU_INPUT} -sample_size"\
                " 5000 -output {OTU_OUTPUT}"
ALPHA_DIVERSITY = "usearch -alpha_div {OTU_INPUT}"\
                  " -output {ALPHA_OUTPUT}"
OTU_IDENTIFY = "classifier classify -t {CLASSIFIER}"\
               " -c 1 -w 150 -o {TABLE_OUTPUT}"\
               " -h {HIER_OUTPUT} {FASTA_INPUT}"

GUNZIP = "gunzip {folder}/*.gz"

COUNT_READS = "echo $(cat {fastq}|wc -l)/4|bc"

THRESHOLDS = np.round(np.arange(0.1, 1.01, 0.01), 2)

CHOICES_R = ["superkingdom", "kingdom", "phylum",
             "class", "order", "family", "genus",
             "species"]


def getArguments():
    """Get arguments from terminal

    This function gets arguments from terminal via argparse

    Returns
    -------------
    arguments: Namespace
        Namespace object with all arguments
    """

    num_threads = cpu_count() - 2
    if num_threads < 1:
        num_threads = 1

    parser = argparse.ArgumentParser(
        description="SeqME pipeline for Metabarcoding")
    parser.add_argument("folder_fastq", type=str,
                        help="A folder with fastq files")
    parser.add_argument("-t", "--threshold", nargs="?", const=1, default=1,
                        choices=THRESHOLDS, type = float,
```

```
                    help="Specify the minimum threshold"\
                    " to the taxonomy rank be kept, default: 1")
    parser.add_argument("-o", "--only_joining", nargs="?", const="",
                        default="", type = str,
                    help="Inform folder - Only the final joining of"\
                    " the results is done")
    parser.add_argument("-n", "--num_threads", nargs="?", type = int,
                        const=num_threads, default=num_threads,
                    help="Number of threads to be executed in parallel")
    parser.add_argument("-no", "--normalized", action="store_true",
                    help="Use normalized data")
    parser.add_argument("-r", "--rank", nargs="?", const="species", default="species",
                    choices=CHOICES_R, type = lambda s : s.lower(),
                    help="Lowest taxonomic classification rank"\
                    " to be in the result, default: species")

    return parser.parse_args()

def join_paired_ends(fastq_file, base_name):
    """Join paired-end Illumina data.

    From a pair forward (R1) and reverse (R2),
    this function creates the command line
    to merge the pair of files into a single
    sequence using fastq-join.

    fastq_file: str
        fastq file
    base_name: str
        fastq file base name

    Returns
    -------------
    command: tuple
        A tuple with name, command and log
    """

    # Output file name
    output_file_name = "{}_\%.fastq".format(base_name)

    # Folder to be saved
    fastq_output = "{PATH_FASTQ_JOINED}/{FASTQ_OUTPUT}".format(
                    PATH_FASTQ_JOINED=PATH_FASTQ_JOINED,
                    FASTQ_OUTPUT= output_file_name
                    )

```

```
148     return ("Joining paired ends: {file_name}".format(file_name=base_name),
149             "{PATH_FASTQ_JOINED}/{base_name}.log".format(
150                 PATH_FASTQ_JOINED=PATH_FASTQ_JOINED,
151                 base_name=base_name),
152             FASTQ_JOIN.format(R1=fastq_file,
153                               R2=fastq_file.replace("R1", "R2"),
154                               FASTQ_OUTPUT=fastq_output)
155         )
156
157 def quality_filtering(base_name):
158     """Quality filtering of fastq file.
159
160     From a fastq file, this function creates the
161     command line to remove low quality nucleotides.
162
163     base_name: str
164         fastq file base name
165
166     Returns
167     -------------
168     command: tuple
169         A tuple with name, command and log
170     """
171
172     # Output and joined file names
173     output_file_name = "{}.fastq".format(base_name)
174     joined_file_name = "{}_join.fastq".format(base_name)
175
176     # Joined file path
177     joined_file_path = "{PATH_FASTQ_JOINED}/{FASTQ_INPUT}".format(
178                   PATH_FASTQ_JOINED=PATH_FASTQ_JOINED,
179                   FASTQ_INPUT= joined_file_name
180                   )
181
182     # Folder to be saved
183     fastq_output = "{PATH_QUALITY_FILTERED}/{FASTQ_OUTPUT}".format(
184                   PATH_QUALITY_FILTERED=PATH_QUALITY_FILTERED,
185                   FASTQ_OUTPUT= output_file_name
186                   )
187
188     return ("Quality filtering: {file_name}".format(file_name=base_name),
189             "{PATH_QUALITY_FILTERED}/{base_name}.log".format(
190                 PATH_QUALITY_FILTERED=PATH_QUALITY_FILTERED,
191                 base_name=base_name),
192             FASTQ_QUALITY_FILTER.format(FASTQ_INPUT=joined_file_path,
193                               FASTQ_OUTPUT=fastq_output)
```

```
194         )
195 
196 def convert_fastq_to_fasta(base_name):
197     """Convert fastq to fasta.
198 
199     From a fastq file, this function converts fastq
200     to fasta format.
201 
202     base_name: str
203         fastq file base name
204 
205     Returns
206     -------------
207     command: tuple
208         A tuple with name, command and log
209     """
210 
211     # Output and input file names
212     input_file_name = "{}.fastq".format(base_name)
213     output_file_name = "{}.fasta".format(base_name)
214 
215     fastq_input = "{PATH_QUALITY_FILTERED}/{FASTQ_INPUT}".format(
216                   PATH_QUALITY_FILTERED=PATH_QUALITY_FILTERED,
217                   FASTQ_INPUT= input_file_name
218                   )
219     fasta_output = "{PATH_FASTA}/{FASTA_OUTPUT}".format(
220                   PATH_FASTA=PATH_FASTA,
221                   FASTA_OUTPUT= output_file_name
222                   )
223 
224     return ("Converting fastq to fasta: {file_name}".format(file_name=base_name),
225             "{PATH_FASTA}/{base_name}.log".format(
226                 PATH_FASTA=PATH_FASTA,
227                 base_name=base_name),
228             FASTQ_TO_FASTA.format(FASTQ_INPUT=fastq_input,
229                                   FASTA_OUTPUT=fasta_output)
230         )
231 
232 def remove_short_long_seq(base_name):
233     """Remove too short and too long sequences.
234 
235     From a fasta file, this function removes
236     too short and too long sequences.
237 
238     base_name: str
239         fastq file base name
```

```
240
241     Returns
242     -------------
243     command: tuple
244         A tuple with name, command and log
245     """
246
247     # Output and input file names
248     fasta_file_name = "{}.fasta".format(base_name)
249
250     fasta_input = "{PATH_FASTA}/{FASTA_INPUT}".format(
251                 PATH_FASTA=PATH_FASTA,
252                 FASTA_INPUT= fasta_file_name
253                     )
254     fasta_output = "{PATH_FASTA_REMOVED_SHORT_LONG_SEQ}/{FASTA_OUTPUT}".format(
255                 PATH_FASTA_REMOVED_SHORT_LONG_SEQ=PATH_FASTA_REMOVED_SHORT_LONG_SEQ,
256                 FASTA_OUTPUT= fasta_file_name
257                     )
258
259     return ("Removing short and long seq: {file_name}".format(file_name=base_name),
260             "{PATH_FASTA_REMOVED_SHORT_LONG_SEQ}/{base_name}.log".format(
261                 PATH_FASTA_REMOVED_SHORT_LONG_SEQ=PATH_FASTA_REMOVED_SHORT_LONG_SEQ,
262                 base_name=base_name),
263             FASTA_REMOVE_SHORT_LONG_SEQ.format(FASTA_INPUT=fasta_input,
264                                                FASTA_OUTPUT=fasta_output)
265         )
266
267 def cluster_uniques(base_name):
268     """Cluster uniques sequences (dereplication).
269
270     From a fasta file, this function finds a set
271     of unique sequences in the file.
272
273     base_name: str
274         fastq file base name
275
276     Returns
277     -------------
278     command: tuple
279         A tuple with name, command and log
280     """
281
282     # Output and input file names
283     fasta_file_name = "{}.fasta".format(base_name)
284     uc_file_name = "{}.uc".format(base_name)
285
```

```
286     fasta_input = "{PATH_FASTA_REMOVED_SHORT_LONG_SEQ}/{FASTA_INPUT}".format(
287                 PATH_FASTA_REMOVED_SHORT_LONG_SEQ=PATH_FASTA_REMOVED_SHORT_LONG_SEQ,
288                 FASTA_INPUT= fasta_file_name
289                     )
290     fasta_output = "{PATH_UNIQUES}/{FASTA_OUTPUT}".format(
291                 PATH_UNIQUES=PATH_UNIQUES,
292                 FASTA_OUTPUT= fasta_file_name
293                     )
294     uc_output = "{PATH_UNIQUES}/{UC_OUTPUT}".format(
295                 PATH_UNIQUES=PATH_UNIQUES,
296                 UC_OUTPUT= uc_file_name
297                     )
298 
299     return ("Clustering uniques (Dereplication): {file_name}".format(file_name=
        ↪ base_name),
300             "{PATH_UNIQUES}/{base_name}.log".format(
301                 PATH_UNIQUES=PATH_UNIQUES,
302                 base_name=base_name),
303             FASTA_CLUSTER_UNIQUES.format(FASTA_INPUT=fasta_input,
304                                          FASTA_OUTPUT=fasta_output,
305                                          UC_OUTPUT=uc_output)
306         )
307 
308 def cluster_otus(base_name):
309     """Cluster OTUs.
310 
311     From a fasta file, this function does a OTU
312     clustering. Chimeras are also filtered
313     during this step.
314 
315     base_name: str
316         fastq file base name
317 
318     Returns
319     -------------
320     command: tuple
321         A tuple with name, command and log
322     """
323 
324     # Output and input file names
325     fasta_file_name = "{}.fasta".format(base_name)
326 
327     fasta_input = "{PATH_UNIQUES}/{FASTA_INPUT}".format(
328                 PATH_UNIQUES=PATH_UNIQUES,
329                 FASTA_INPUT= fasta_file_name
330                     )
```

```
    fasta_output = "{PATH_OTUS_FASTA}/{FASTA_OUTPUT}".format(
                PATH_OTUS_FASTA=PATH_OTUS_FASTA,
                FASTA_OUTPUT= fasta_file_name
                    )

    return ("Clustering OTUs: {file_name}".format(file_name=base_name),
            "{PATH_OTUS_FASTA}/{base_name}.log".format(
                PATH_OTUS_FASTA=PATH_OTUS_FASTA,
                base_name=base_name),
            FASTA_CLUSTER_OTU.format(FASTA_INPUT=fasta_input,
                                     FASTA_OUTPUT=fasta_output)
        )

def create_otu_tables(base_name):
    """Create OTU tables.

    From a fasta file, this function creates
    an OTU table with the identification of which
    OTU the sequence belongs to, and number of sequences
    for each OTU.

    base_name: str
        fastq file base name

    Returns
    -------------
    command: tuple
        A tuple with name, command and log
    """

    # Output and input file names
    fasta_file_name = "{}.fasta".format(base_name)
    otu_file_name = "{}.otu".format(base_name)
    map_file_name = "{}.map".format(base_name)

    fasta_input = "{PATH_FASTA_REMOVED_SHORT_LONG_SEQ}/{FASTA_INPUT}".format(
                PATH_FASTA_REMOVED_SHORT_LONG_SEQ=PATH_FASTA_REMOVED_SHORT_LONG_SEQ,
                FASTA_INPUT= fasta_file_name
                    )
    fasta_otu_input = "{PATH_OTUS_FASTA}/{FASTA_OTU_INPUT}".format(
                PATH_OTUS_FASTA=PATH_OTUS_FASTA,
                FASTA_OTU_INPUT= fasta_file_name
                    )
    otu_output = "{PATH_OTUS_TABLE}/{OTU_OUTPUT}".format(
                PATH_OTUS_TABLE=PATH_OTUS_TABLE,
                OTU_OUTPUT= otu_file_name
```

```
                )
    map_output = "{PATH_OTUS_TABLE}/{MAP_OUTPUT}".format(
                PATH_OTUS_TABLE=PATH_OTUS_TABLE,
                MAP_OUTPUT= map_file_name
                )

    return ("Creating OTU tables: {file_name}".format(file_name=base_name),
            "{PATH_OTUS_TABLE}/{base_name}.log".format(
                PATH_OTUS_TABLE=PATH_OTUS_TABLE,
                base_name=base_name),
            FASTA_CREATE_OTU_TABLE.format(FASTA_INPUT=fasta_input,
                                          FASTA_OTU_INPUT=fasta_otu_input,
                                          OTU_OUTPUT=otu_output,
                                          MAP_OUTPUT=map_output)
        )

def normalize_otu_tables(base_name):
    """Normalize OTU tables.

    From a fasta file, this function
    normalizes all samples.

    base_name: str
        fastq file base name

    Returns
    -------------
    command: tuple
        A tuple with name, command and log
    """

    # Output and input file names
    otu_file_name = "{}.otu".format(base_name)

    otu_input = "{PATH_OTUS_TABLE}/{OTU_INPUT}".format(
                PATH_OTUS_TABLE=PATH_OTUS_TABLE,
                OTU_INPUT= otu_file_name
                )
    otu_output = "{PATH_OTUS_TABLE_NORMALIZE}/{OTU_OUTPUT}".format(
                PATH_OTUS_TABLE_NORMALIZE=PATH_OTUS_TABLE_NORMALIZE,
                OTU_OUTPUT= otu_file_name
                )

    return ("Normalizing OTU tables: {file_name}".format(file_name=base_name),
            "{PATH_OTUS_TABLE_NORMALIZE}/{base_name}.log".format(
                PATH_OTUS_TABLE_NORMALIZE=PATH_OTUS_TABLE_NORMALIZE,
```

```
423                 base_name=base_name),
424             OTU_NORMALIZE.format(OTU_INPUT=otu_input,
425                                  OTU_OUTPUT=otu_output)
426         )
427
428 def calculate_alpha_diversity(base_name):
429     """Calculate alpha diversity.
430
431     From a fasta file, this function calculates
432     the alpha diversity.
433
434     base_name: str
435         fastq file base name
436
437     Returns
438     -------------
439     command: tuple
440         A tuple with name, command and log
441     """
442
443     # Output and input file names
444     otu_file_name = "{}.otu".format(base_name)
445     alpha_file_name = "{}.alpha".format(base_name)
446
447     otu_input = "{PATH_OTUS_TABLE_NORMALIZE}/{OTU_INPUT}".format(
448                 PATH_OTUS_TABLE_NORMALIZE=PATH_OTUS_TABLE_NORMALIZE,
449                 OTU_INPUT= otu_file_name
450                     )
451     alpha_output = "{PATH_ALPHA_DIVERSITY}/{ALPHA_OUTPUT}".format(
452                 PATH_ALPHA_DIVERSITY=PATH_ALPHA_DIVERSITY,
453                 ALPHA_OUTPUT= alpha_file_name
454                     )
455
456     return ("Calculating alpha diversity: {file_name}".format(file_name=base_name),
457             "{PATH_ALPHA_DIVERSITY}/{base_name}.log".format(
458                 PATH_ALPHA_DIVERSITY=PATH_ALPHA_DIVERSITY,
459                 base_name=base_name),
460             ALPHA_DIVERSITY.format(OTU_INPUT=otu_input,
461                                    ALPHA_OUTPUT=alpha_output)
462         )
463
464 def identify_otu_by_classifier(base_name):
465     """Identify OTU by classifier.
466
467     From a fasta file, this function identifies
468     taxonomy rank for each OTU using a Naive Bayes
```

```
469     classifier.
470 
471     base_name: str
472         fastq file base name
473 
474     Returns
475     -------------
476     command: tuple
477         A tuple with name, command and log
478     """
479 
480     # Output and input file names
481     fasta_file_name = "{}.fasta".format(base_name)
482     hier_file_name = "{}.hier".format(base_name)
483     table_file_name = "{}.csv".format(base_name)
484 
485     fasta_otu_input = "{PATH_OTUS_FASTA}/{FASTA_OTU_INPUT}".format(
486                 PATH_OTUS_FASTA=PATH_OTUS_FASTA,
487                 FASTA_OTU_INPUT= fasta_file_name
488                     )
489     hier_output = "{PATH_OTU_IDENTIFIED}/{HIER_OUTPUT}".format(
490                 PATH_OTU_IDENTIFIED=PATH_OTU_IDENTIFIED,
491                 HIER_OUTPUT= hier_file_name
492                     )
493     table_output = "{PATH_OTU_IDENTIFIED}/{TABLE_OUTPUT}".format(
494                 PATH_OTU_IDENTIFIED=PATH_OTU_IDENTIFIED,
495                 TABLE_OUTPUT= table_file_name
496                     )
497 
498     return ("Identifying OTU by classifier: {file_name}".format(file_name=base_name),
499             "{PATH_OTU_IDENTIFIED}/{base_name}.log".format(
500                 PATH_OTU_IDENTIFIED=PATH_OTU_IDENTIFIED,
501                 base_name=base_name),
502             OTU_IDENTIFY.format(CLASSIFIER=PATH_CLASSIFIER,
503                                 TABLE_OUTPUT=table_output,
504                                 HIER_OUTPUT=hier_output,
505                                 FASTA_INPUT=fasta_otu_input)
506         )
507 
508 def join_results(base_names, threshold, normalized, rank_to_stop):
509     """Join the results in one unique csv file.
510 
511     From the list of fastq files, this function joins
512     them and create a unique file with the taxonomic rank
513     and the number of reads.
514 
```

```
    base_names: list
        List with base names of fastq files
    threshold: float
        Minimum threshold to the taxonomy rank be kept
    normalized: bool
        Use normalized data
    rank_to_stop: str
        Taxonomic rank to stop searching
    """

    # Initialize dataframe
    df = pd.DataFrame()

    otus_identified = sorted(glob.glob(
        "{PATH_OTU_IDENTIFIED}/*.csv".format(
        PATH_OTU_IDENTIFIED=PATH_OTU_IDENTIFIED)))

    for otu_identified in tqdm(otus_identified, desc="Joining results in a unique file
        ↪ "):
        try:
            file_name = os.path.basename(otu_identified)
            base_name = os.path.splitext(file_name)[0]

            with open("{PATH_OTU_IDENTIFIED}/{TABLE}.csv".format(
                    PATH_OTU_IDENTIFIED=PATH_OTU_IDENTIFIED,
                    TABLE=base_name
                        )) as table_file:
                table = table_file.readlines()

            if not normalized:
                otu_path = "{PATH_OTUS_TABLE}/{OTU}.otu".format(
                        PATH_OTUS_TABLE=PATH_OTUS_TABLE,
                        OTU=base_name
                            )
            else:
                otu_path = "{PATH_OTUS_TABLE_NORMALIZE}/{OTU}.otu".format(
                        PATH_OTUS_TABLE_NORMALIZE=PATH_OTUS_TABLE_NORMALIZE,
                        OTU=base_name
                            )

            with open(otu_path) as otu_file:
                df = df.append(pd.read_table(otu_file, index_col=0,
                                             names=[base_name], skiprows=1))

            for line in table:
                line = line.split("\t")
```

```
                otu_id = line[0]

                for i in range(len(line)-1, 1, -3):
                    value = float(line[i])
                    rank = line[i-1]
                    tax = line[i-2]

                    if value >= threshold:
                        df.rename(index={otu_id:tax}, inplace=True)
                        break

                    if rank == rank_to_stop.lower():
                        break

                if otu_id in df.index:
                    df.drop(otu_id, inplace=True)

        except Exception as e:
            with open("{PATH_RESULT}/{DATETIME}_join_results.log".format(
                PATH_RESULT=PATH_RESULT,
                DATETIME=DATETIME),
                "a") as log:
                print(e, file=log)

    df.index.name = "TAX"
    df = df.groupby(df.index).sum()
    df.sort_index(axis=1, inplace=True)
    df.sort_index().to_csv("{PATH_RESULT}/{DATETIME}_SeqME.tsv".format(
            PATH_RESULT=PATH_RESULT,
            DATETIME=DATETIME),
                sep="\t")

def initialize_paths():
    """Initialize Paths

    This function initilizes paths to
    the results of each step of the pipeline.
    """

    global PATH_FASTQ_JOINED, PATH_QUALITY_FILTERED, PATH_FASTA, \
            PATH_FASTA_REMOVED_SHORT_LONG_SEQ, PATH_UNIQUES, \
            PATH_OTUS_FASTA, PATH_OTUS_TABLE, PATH_OTUS_TABLE_NORMALIZE, \
            PATH_ALPHA_DIVERSITY, PATH_OTU_IDENTIFIED

    PATH_FASTQ_JOINED = PATH_FASTQ_JOINED.format(PATH_RESULT=PATH_RESULT)
    PATH_QUALITY_FILTERED = PATH_QUALITY_FILTERED.format(PATH_RESULT=PATH_RESULT)
```

```
606     PATH_FASTA = PATH_FASTA.format(PATH_RESULT=PATH_RESULT)
607     PATH_FASTA_REMOVED_SHORT_LONG_SEQ = PATH_FASTA_REMOVED_SHORT_LONG_SEQ.format(
608                                         PATH_RESULT=PATH_RESULT)
609     PATH_UNIQUES = PATH_UNIQUES.format(PATH_RESULT=PATH_RESULT)
610     PATH_OTUS_FASTA = PATH_OTUS_FASTA.format(PATH_RESULT=PATH_RESULT)
611     PATH_OTUS_TABLE = PATH_OTUS_TABLE.format(PATH_RESULT=PATH_RESULT)
612     PATH_OTUS_TABLE_NORMALIZE = PATH_OTUS_TABLE_NORMALIZE.format(
613                                 PATH_RESULT=PATH_RESULT)
614     PATH_ALPHA_DIVERSITY = PATH_ALPHA_DIVERSITY.format(
615                            PATH_RESULT=PATH_RESULT)
616     PATH_OTU_IDENTIFIED = PATH_OTU_IDENTIFIED.format(PATH_RESULT=PATH_RESULT)
617 
618 def create_folders():
619     """Create folders
620 
621     This function create folders to
622     the results of each step of the pipeline
623     """
624 
625     Path(PATH_RESULT).mkdir(parents=True, exist_ok=True)
626     Path(PATH_FASTQ_JOINED).mkdir(parents=True, exist_ok=True)
627     Path(PATH_QUALITY_FILTERED).mkdir(parents=True, exist_ok=True)
628     Path(PATH_FASTA).mkdir(parents=True, exist_ok=True)
629     Path(PATH_FASTA_REMOVED_SHORT_LONG_SEQ).mkdir(parents=True, exist_ok=True)
630     Path(PATH_UNIQUES).mkdir(parents=True, exist_ok=True)
631     Path(PATH_OTUS_FASTA).mkdir(parents=True, exist_ok=True)
632     Path(PATH_OTUS_TABLE).mkdir(parents=True, exist_ok=True)
633     Path(PATH_OTUS_TABLE_NORMALIZE).mkdir(parents=True, exist_ok=True)
634     Path(PATH_ALPHA_DIVERSITY).mkdir(parents=True, exist_ok=True)
635     Path(PATH_OTU_IDENTIFIED).mkdir(parents=True, exist_ok=True)
636 
637 def execute_command(base_name):
638     """Execute command
639 
640     This function executes command from terminal.
641 
642     base_name: str
643         Base name of the fastq file
644     """
645 
646     for command in commands[base_name]:
647         print(command[0])
648         with open(command[1], "a") as log_file:
649             run(command[2], shell=True, stderr=log_file, stdout=log_file)
650 
651 
```

```
652 if __name__ == "__main__":
653     args = getArguments()
654 
655     initial_time = datetime.datetime.now()
656 
657     # Unzipping compressed files
658     run(GUNZIP.format(folder=args.folder_fastq),
659         shell=True, stdout=DEVNULL, stderr=STDOUT)
660 
661     # Reading fastq files
662     fastq_files = sorted(glob.glob("{folder}/*R1*.fastq".format(
663         folder=args.folder_fastq)))
664     base_names = []
665     commands = {}
666     reads = {}
667 
668     if args.only_joining:
669         PATH_RESULT = args.only_joining
670         initialize_paths()
671     else:
672         initialize_paths()
673 
674     for fastq_file in tqdm(fastq_files, desc="Reading fastq files"):
675         file_name = os.path.basename(fastq_file)
676         base_name = file_name.split("_")[0].split(".R1")[0]
677         base_names.append(base_name)
678 
679         output = check_output(COUNT_READS.format(
680                               fastq=fastq_file), shell=True)
681         reads[base_name] = int(output)
682 
683         commands[base_name] = [join_paired_ends(fastq_file, base_name),
684                               quality_filtering(base_name),
685                               convert_fastq_to_fasta(base_name),
686                               remove_short_long_seq(base_name),
687                               cluster_uniques(base_name),
688                               cluster_otus(base_name),
689                               create_otu_tables(base_name),
690                               normalize_otu_tables(base_name),
691                               calculate_alpha_diversity(base_name),
692                               identify_otu_by_classifier(base_name)]
693 
694     if args.only_joining:
695         if Path(args.only_joining).exists():
696             join_results(base_names, args.threshold, args.normalized, args.rank)
697         else:
```

```
            print("Folder '{FOLDER}' does not exist".format(
                FOLDER=args.only_joining))

        raise SystemExit(0)

    # Creating folders
    create_folders()

    # Running commands
    num_threads = args.num_threads

    pool = Pool(num_threads)
    for returncode in pool.imap(execute_command, commands):
        if returncode:
            print("command failed: {}".format(returncode))


    join_results(base_names, args.threshold, args.normalized, args.rank)

    final_time = datetime.datetime.now()
    total_time = final_time - initial_time
    print("\nTotal time to run the pipeline: {TOTAL_TIME}\n".format(
            TOTAL_TIME=total_time))
```

Source Code A.14: SEQme pipeline workflow.

```
#!/usr/bin/env python3
import os
import sys
import pathlib
import argparse
import datetime
import pandas as pd
from tqdm import tqdm
from pathlib import Path
from subprocess import check_output
from collections import defaultdict

"""Count reads from fasta or fastq inside the folder and
create a csv file.

For each fasta or fastq inside the folder informed count
the number of reads and create a table where the row name
is informed and column names are parsed from the files.
"""
```

```
20
21  DATETIME = datetime.datetime.now().strftime("%d%m%Y_%H%M%S")
22
23  PATH_RESULT = "{DATETIME}_Count_Reads".format(DATETIME=DATETIME)
24
25  FASTA_COUNT_READS = '{Z}grep -c ">" {FASTA}'
26  FASTQ_COUNT_READS = "echo $({Z}cat {FASTQ}|wc -l)/4|bc"
27
28  CHOICES_F = ['fastq', 'fasta']
29
30  def getArguments():
31      """Get arguments from terminal
32
33      This function gets arguments from terminal via argparse
34
35      Returns
36      -------------
37      arguments: Namespace
38          Namespace object with all arguments
39      """
40
41      parser = argparse.ArgumentParser(
42          description="Count reads from FASTA or FASTQ files")
43      parser.add_argument("folder_fastaq", type=str,
44                          help="A folder with FASTA or FASTQ files")
45      parser.add_argument("extension", type = lambda s : s.lower(),
46                          choices=CHOICES_F, help="Files extension")
47      parser.add_argument("row_name", type=str,
48                          help="Row name of the new table created")
49      parser.add_argument("pattern", type=str,
50                          help="A pattern to identify files to be parsed")
51      parser.add_argument("-seqme", action='store_true',
52                          help="Are the files from SeqME pipeline?")
53
54      return parser.parse_args()
55
56  def count_reads(fastaq_files, row_name, pattern, extension, seqme):
57      """Count the reads from FASTA/Q files and create a table
58
59      From the list of FASTA or FASTQ files, this function
60      counts the number of reads and create a csv table with
61      the row name informed and column names are parsed
62      from files.
63
64      fastaq_files: list
65          List with FASTA or FASTQ files
```

```
    row_name: str
        Row name
    pattern: str
        Pattern for the identification of files
    extension: str
        Extension of files
    seqme: bool
        Boolean if files are from SeqME
    """

    # Initialize dataframe
    df = pd.DataFrame()
    seqme_count = defaultdict(int)

    for fastaq_file in tqdm(fastaq_files, desc="Counting reads"):
        file_name = os.path.basename(fastaq_file)
        base_name = file_name.replace(pattern, "")

        if seqme:
            with open(fastaq_file) as fastaq:
                fasta = list(filter(None, fastaq.read().split(">")))

                for sequence in fasta:
                    base_name = sequence.split("|")[0]
                    seqme_count[base_name] = seqme_count[base_name] + 1

                continue

        z = ""
        if pathlib.Path(fastaq_file).suffix == ".gz":
            z = "z"

        try:
            if extension == "fasta":
                output = check_output(FASTA_COUNT_READS.format(
                                      Z=z,
                                      FASTA=fastaq_file),
                                      shell=True)
            elif extension == "fastq":
                output = check_output(FASTQ_COUNT_READS.format(
                                      Z=z,
                                      FASTQ=fastaq_file),
                                      shell=True)
        except:
            output = 0

```

```
        df.loc[row_name, base_name] = int(output)

    if seqme:
        df = pd.DataFrame(seqme_count, index=[row_name,])

    df.sort_index(axis=1).to_csv(
            "{PATH_RESULT}/{DATETIME}_Count_Reads.tsv".format(
            PATH_RESULT=PATH_RESULT,
            DATETIME=DATETIME),
                sep="\t")

if __name__ == "__main__":
    args = getArguments()

    Path(PATH_RESULT).mkdir(parents=True, exist_ok=True)

    fastaq_files = Path(args.folder_fastaq).rglob(
                        "*{PATTERN}*".format(PATTERN=args.pattern))

    count_reads(fastaq_files, args.row_name, args.pattern, args.extension, args.seqme)
```

Source Code A.15: Count the number of sequence reads for each FASTA/FASTQ file in a folder based on pattern provided.

```
library(rChoiceDialogs)
library(data.table)

# Threshold of 0.1%
# It needs to be divided by 100 in R
threshold = 0.001

# Get table file names
tables.path = list.files(rchoose.dir(caption = "Choose tables directory"), pattern =
    ↪ "*.tsv", full.names = TRUE ,recursive = TRUE)

for (file.name in tables.path) {

  # Parse base name and dir name
  base.name = basename(file.name)
  dir.name = dirname(file.name)

  # Read file
  dat = read.csv(file.name, sep = '\t', header=T, row.names = 1, check.names=F)
  # Remove last column "taxomomy" if you did not remove it#
  # dat = dat[, !colnames(dat) %in% "taxomomy",]
```

```
21
22    # Transpose data
23    dat = as.data.frame(t(dat))
24
25    # Create 'datt': a proportion reads data frame#
26    datt = dat / dat$Total
27    datt[is.na(datt)] <- 0
28
29    # Apply threshold to proportions of datt to dat#
30    dat[datt < threshold] = 0
31
32    # Calculate assigned
33    dat$Assigned = rowSums(dat[, !colnames(dat) %in% c("Unassigned", "Assigned", "Total
          ↪ ")])
34
35    # Calculate unassigned
36    dat$Unassigned = dat$Total - dat$Assigned
37
38    # Transpose back
39    dat = as.data.frame(t(dat))
40
41    # Remove species that sum zero
42    dat = dat[rowSums(dat) > 0,]
43
44    # Export dataframe to CSV
45    file.result.name <- paste(dir.name, '/Filtered_', base.name, sep = "")
46    fwrite(x=dat, file=file.result.name, sep = "\t", row.names = T)
47  }
```

Source Code A.16: Remove assignments where the number of reads assigned falls below a threshold of 0.1 % of the total of reads assigned for the sample.

```
1  library(docstring)
2  library(data.table)
3  library(tidyverse)
4  library(collections)
5  library(taxize)
6  library(zeallot)
7
8  auto.increment = function(variable){
9    #' "Auto" increment 1 to the variable
10   #'
11   #' @description This function adds 1 to the variable.
12   #'
13   #' @param variable numeric.
```

```
  #' @usage auto.increment(variable)
  #' @return It returns the variable after adding 1 to it.
  #' @details The input must be a numeric.
  #' @examples
  #' auto.increment(variable)

  # Get name of the variable
  name = deparse(substitute(variable))

  # Get value of the variable
  value = get(name)

  # Sum 1 to the value
  value = value + 1

  # Assign new value to a global variable with the same name
  # Auto-increment
  assign(name, value, envir = .GlobalEnv)

  # Return new value
  return(value)
}

modify.column.names <- function(column.names){
  #' Modify column names
  #'
  #' @description This function modifies the column names.
  #'
  #' It adds the name of the pipeline at the end, and it formats the
  #' reservoir and season names.
  #'
  #' @param column.names list
  #' @usage modify.column.names(column.names)
  #' @return It returns the new name.
  #' @details The input must be a list.
  #' @examples
  #' modify.column.name(column.names)

  # Create a dictionary for reservoirs
  reservoirs <- dict()
  reservoirs$set("K", "Klíčava")
  reservoirs$set("R", "Římov")
  reservoirs$set("Z", "Žlutice")

  # Create a dictionary for seasons
  seasons <- dict()
```

```
  seasons$set("S", "Summer")
  seasons$set("W", "Autumn")

  new.column.names <- lapply(column.names, function(column.name) {
    # Get reservoir
    reservoir = reservoirs$get(toupper(substring(column.name, 1, 1)))

    # Get season
    season = seasons$get(toupper(substring(column.name, 2, 2)))

    # Get extra info
    extra.info = substring(column.name, 3)

    # Join all info in a vector
    info = c(reservoir, season, extra.info, table.name)

    # Remove empty element in the vector
    info = info[info != ""]

    # Join reservoir, season, extra info, and pipeline name.
    new.name = paste(info, collapse = " ")

    # Return the new name
    return(new.name)
  })

  # Return new column names
  return(new.column.names)
}

create.tables.all.pipelines.detailed <- function(){
  #' Create Table All Pipelines Detailed
  #'
  #' @description This function creates a detailed version of the
  #' table using all pipelines.
  #'
  #' @usage create.tables.all.pipelines.detailed()
  #' @return It does not return anything.
  #' @examples
  #' create.tables.all.pipelines.detailed()

  # Remove TAX column and rows Total, Assigned and Unassigned
  table = all.pipelines.reduced.no.controls
  table$TAX = NULL
  table = table[!(row.names(table) %in% c("Total", "Assigned", "Unassigned")), ]
  table = as.data.frame(t(table))
```

```
106
107   # Transform the name of the rows as the first column
108   table = tibble::rownames_to_column(table, "Reservoir_Season_Pipeline")
109   rownames(table) = table$Reservoir_Season_Pipeline
110   table = table %>% separate(Reservoir_Season_Pipeline, c("Reservoir", "Season", "
          ↪ Pipeline"), " ")
111
112   # Assign table name
113   assign("all.pipelines.detailed", table, envir = .GlobalEnv)
114
115   # Write tsv file
116   file = paste(TABLES.DETAILED, "/All_Pipelines_detailed.csv", sep = "")
117   fwrite(x=table, file=file, sep = "\t", row.names = T)
118
119 }
120
121 create.tables.ranacapa.metadata <- function(table, table.name, reduced){
122   #' Create Tables Ranacapa Metadata
123   #'
124   #' @description This function creates a metadata table
125   #' to be used with Ranacapa statistical analyses.
126   #'
127   #' @param table dataframe.
128   #' @param table.name name for the new table.
129   #' @param reduced True if it is reduced or False if not.
130   #' @usage create.tables.ranacapa.metadata(table, table.name, reduced)
131   #' @return It does not return anything.
132   #' @examples
133   #' create.tables.ranacapa.metadata(ranacapa, "metadata", T)
134
135   # Create new dataframes
136   columns <- c("Sample", "Sample_or_Control", "Reservoir", "Season", "Pipeline")
137   ###
138   metadata <- data.frame(matrix(ncol = length(columns), nrow = 0))
139   colnames(metadata) <- columns
140
141   # Get the name of the columns
142   columns = colnames(table)[colnames(table) != "sum.taxonomy"]
143
144   # For each column name
145   # Get reservoir, season, pipeline and sample_or_control
146   for (column in columns) {
147
148     column.info = str_split(column, "\\.", n = Inf, simplify = FALSE)
149
150     # Extract info
```

```
    reservoir = column.info[[1]][1]
    season = column.info[[1]][2]
    info = column.info[[1]][3]
    pipeline = column.info[[1]][4]

    # Check if it is sample or control
    sample_or_control = "control"
    if (str_detect(info, ".*[0-9].*")) {
      sample_or_control = "sample"
    }

    if(reduced){
      pipeline = column.info[[1]][3]
      sample_or_control = "sample"
    }

    # Add row to the dataframe
    metadata = rbind(metadata, list(Sample = column,
                                    Sample_or_Control = sample_or_control,
                                    Reservoir = reservoir,
                                    Season = season,
                                    Pipeline = pipeline),
                     stringsAsFactors = FALSE)

  }

  # Write tsv file
  file = paste(TABLES.RANACAPA, "/", table.name, ".txt", sep = "")
  fwrite(x=metadata, file=file, sep = "\t", row.names = F)
}

create.tables.ranacapa <- function(table, table.name, reduced){
  #' Create Tables Ranacapa
  #'
  #' @description This function creates a table to be used with
  #' Ranacapa statistical analyses.
  #'
  #' @param table dataframe.
  #' @param table.name name for the new table.
  #' @param reduced True if it is reduced or False if not.
  #' @usage create.tables.ranacapa(table, table.name, reduced)
  #' @return It does not return anything.
  #' @examples
  #' create.tables.ranacapa(all.pipelines, "my_new_table", T)

  # Create initial dataframes
```

```
  ranacapa = table[!(row.names(table) %in% c("Total", "Assigned", "Unassigned")), ]

  # Get row names
  tax = rownames(ranacapa)

  # Change column name
  colnames(ranacapa)[which(names(ranacapa) == "TAX")] <- "sum.taxonomy"

  # Get taxonomic information
  tax.info = classification(tax, db='gbif', rows = 1)

  # Change tax from only species by phylum to species
  for (species in ranacapa$sum.taxonomy) {
    info = tax.info[[species]]
    phylum.to.species = paste(info[info$rank == "phylum",]$name,
                              info[info$rank == "class",]$name,
                              info[info$rank == "order",]$name,
                              info[info$rank == "family",]$name,
                              info[info$rank == "genus",]$name,
                              info[info$rank == "species",]$name,
                              sep = ";")

    ranacapa$sum.taxonomy[ranacapa$sum.taxonomy == species] = phylum.to.species
  }

  # Create metadata
  file.name = paste(table.name, "_metadata", sep = "")
  create.tables.ranacapa.metadata(ranacapa, file.name, reduced)

  # Write tsv file
  file = paste(TABLES.RANACAPA, "/", table.name, ".txt", sep = "")
  fwrite(x=ranacapa, file=file, sep = "\t", row.names = F)

}

create.table.joined.reduced <- function(tables){
  #' Create Table Joined Reduced for all Pipelines
  #'
  #' @description This function creates a table joining all pipelines
  #' in a unique table for each tuple reservoir and season. It also
  #' removes controls.
  #'
  #' @param tables vector with name each table variable.
  #' @usage create.table.joined.reduced(tables)
  #' @return It does not return anything.
  #' @details The input must be a vector with the names
```

```
243   #' of the tables.
244   #' @examples
245   #' create.table.joined.reduced(tables)
246
247   # Create new dataframe
248   columns <- c("TAX")
249   all.pipelines.reduced.no.controls <- data.frame(matrix(ncol = length(columns), nrow
          ↪  = 0))
250   colnames(all.pipelines.reduced.no.controls) <- columns
251
252   # For each table
253   for (table.name in tables) {
254
255     # Assign table name
256     assign("table.name", table.name, envir = .GlobalEnv)
257
258     # Get table
259     table = get(table.name)
260
261     # Remove control Mayland zebra
262     table = table[!(row.names(table) %in% "Maylandia zebra"),]
263
264     # Collect control names
265     control.names = colnames(table[,! grepl("\\d", colnames(table))])
266
267     # Remove control columns
268     table = table[,!(colnames(table) %in% control.names)]
269
270     # Create vector with the combination of the reservoir and the season
271     reservoirs.seasons = c("KS", "KW", "RS", "RW", "ZS", "ZW")
272
273     for (reservoir.season in reservoirs.seasons){
274
275       # Sum up columns matching reservoir.season variable pattern
276       row.sums = rowSums(table[ , grepl(reservoir.season, colnames(table))])
277
278       # Remove columns matching reservoir.season variable pattern
279       table = table[ , ! grepl(reservoir.season, colnames(table))]
280
281       # Add column where the name is the pattern from reservoir.season variable
282       table[, reservoir.season] = row.sums
283     }
284
285     # Rename table columns
286     table = table %>% dplyr::rename_all(modify.column.names)
287
```

```
    # Transform the name of the rows as the first column
    table = tibble::rownames_to_column(table, "TAX")

    # Merge table with the dataframe
    all.pipelines.reduced.no.controls = merge(all.pipelines.reduced.no.controls,
                                              table, by="TAX", all=TRUE)
  }

  # Set NA as zero
  all.pipelines.reduced.no.controls[is.na(all.pipelines.reduced.no.controls)] <- 0

  # Set indexes as tax
  rownames(all.pipelines.reduced.no.controls) <- all.pipelines.reduced.no.controls$
      ↪ TAX

  # Get row names
  tax = rownames(all.pipelines.reduced.no.controls)

  # Put Total, Assigned and Unassigned to the end
  tax = tax[which(!tax %in% c("Total", "Assigned", "Unassigned"))]
  tax = c(tax, c("Total", "Assigned", "Unassigned"))
  all.pipelines.reduced.no.controls = all.pipelines.reduced.no.controls[tax,]

  # Sort columns and keep TAX at the beginning
  columns = sort(colnames(all.pipelines.reduced.no.controls))
  columns = columns[which(!columns %in% "TAX")]
  columns = c("TAX", columns)
  all.pipelines.reduced.no.controls = all.pipelines.reduced.no.controls[ , columns]

  # Assign to a global variable
  assign("all.pipelines.reduced.no.controls", all.pipelines.reduced.no.controls,
         envir = .GlobalEnv)

  # Write tsv file
  fwrite(x=all.pipelines.reduced.no.controls, file=paste(TABLES.JOINED.REDUCED,
                                                         "/all_pipelines_reduced_no_
                                                             ↪ control.tsv",
                                                         sep = ""),
         sep = "\t",
         row.names = F)

}

create.table.joined <- function(tables) {
  #' Create Table Joined for all Pipelines
  #'
```

```r
#' @description This function creates a table joining all pipelines
#' in a unique table.
#'
#' @param tables vector with name each table variable.
#' @usage create.table.joined(tables)
#' @return It does not return anything.
#' @details The input must be a vector with the names
#' of the tables.
#' @examples
#' create.table.joined(tables)

# Create new dataframe
columns <- c("TAX")
all.pipelines <- data.frame(matrix(ncol = length(columns), nrow = 0))
colnames(all.pipelines) <- columns

# For each table
for (table.name in tables) {

  # Assign table name
  assign("table.name", table.name, envir = .GlobalEnv)

  # Get table
  table = get(table.name)

  # Rename table columns
  table = table %>% dplyr::rename_all(modify.column.names)

  # Transform the name of the rows as the first column
  table = tibble::rownames_to_column(table, "TAX")

  # Merge table with the dataframe
  all.pipelines = merge(all.pipelines, table, by="TAX", all=TRUE)
}

# Set NA as zero
all.pipelines[is.na(all.pipelines)] <- 0

# Set indexes as tax
rownames(all.pipelines) <- all.pipelines$TAX

# Get row names
tax = rownames(all.pipelines)

# Put Total, Assigned and Unassigned to the end
tax = tax[which(!tax %in% c("Total", "Assigned", "Unassigned"))]
```

```
  tax = c(tax, c("Total", "Assigned", "Unassigned"))
  all.pipelines = all.pipelines[tax,]

  # Sort columns and keep TAX at the beginning
  columns = sort(colnames(all.pipelines))
  columns = columns[which(!columns %in% "TAX")]
  columns = c("TAX", columns)
  all.pipelines = all.pipelines[ , columns]

  # Assign to a global variable
  assign("all.pipelines", all.pipelines, envir = .GlobalEnv)

  # Write tsv file
  fwrite(x=all.pipelines, file=paste(TABLES.JOINED,
                                     "/all_pipelines.tsv",
                                     sep = ""),
         sep = "\t",
         row.names = F)

}

create.table.cumulative.reads <- function(tables){
  #' Create Table Cumulative Reads
  #'
  #' @description This function creates a table showing the
  #' cumulative summary by reads.
  #'
  #' It counts the number of reads for each tuple reservoir and season.
  #'
  #' @param tables vector with name each table variable.
  #' @usage create.table.cumulative.reads(tables)
  #' @return It does not return anything.
  #' @details The input must be a dataframe table.
  #' @examples
  #' create.table.cumulative.reads(tables)

  # Create new dataframe
  columns <- c("Reservoir", "Season", "Assigned_Reads", "Pipeline")
  cumulative.reads <- data.frame(matrix(ncol = length(columns), nrow = 0))
  colnames(cumulative.reads) <- columns

  # For each table
  for (table.name in tables) {

    # Get table
    table = get(table.name)
```

```
424
425     # Count the number of reads detected for each pipeline
426     total = rowSums(table[,! colnames(table) %in% c("Reservoir", "Season")])
427
428     # Add row to the dataframe
429     cumulative.reads = rbind(cumulative.reads, list(Reservoir = table[,"Reservoir"],
430                                                     Season = table[,"Season"],
431                                                     Assigned_Reads = total,
432                                                     Pipeline = rep(table.name,
433                                                                    times = 6,
434                                                                    length.out = NA,
435                                                                    each = 1)
436     ),
437     stringsAsFactors = FALSE)
438
439   }
440
441   # Assign to a global variable
442   assign("cumulative.reads", cumulative.reads, envir = .GlobalEnv)
443
444   # Reset indexes (row names)
445   rownames(cumulative.reads) <- NULL
446
447   # Write tsv file
448   fwrite(x=cumulative.reads, file=paste(TABLES.CUMULATIVE.READS,
449                                         "/","Cumulative_reads",
450                                         ".tsv", sep = ""),
451          sep = "\t",
452          row.names = F)
453
454 }
455
456 create.table.cumulative.species <- function(tables){
457   #' Create Table Cumulative Species
458   #'
459   #' @description This function creates a table
460   #' showing the cumulative summary by species.
461   #'
462   #' It counts the number of species for each tuple reservoir and season.
463   #'
464   #' @param tables vector with name each table variable.
465   #' @usage create.table.cumulative.species(tables)
466   #' @return It does not return anything.
467   #' @details The input must be a dataframe table.
468   #' @examples
469   #' create.table.cumulative.species(tables)
```

```r

# Create new dataframe
columns <- c("Reservoir", "Season", "N_Species", "Pipeline")
cumulative.species <- data.frame(matrix(ncol = length(columns), nrow = 0))
colnames(cumulative.species) <- columns

# For each table
for (table.name in tables) {

  # Get table
  table = get(table.name)

  # Convert to logical
  logical = table[,! colnames(table) %in% c("Reservoir", "Season")] %>% mutate_all(
      ↪ as.logical)

  # Count the number of species detected for each pipeline
  total = rowSums(logical %>% mutate_all(as.numeric))

  # Add row to the dataframe
  cumulative.species = rbind(cumulative.species, list(Reservoir = table[,"Reservoir"
      ↪ ],
                                                      Season = table[,"Season"],
                                                      N_Species = total,
                                                      Pipeline = rep(table.name,
                                                                     times = 6,
                                                                     length.out = NA,
                                                                     each = 1)
  ),
  stringsAsFactors = FALSE)

}

# Assign to a global variable
assign("cumulative.species", cumulative.species, envir = .GlobalEnv)

# Reset indexes (row names)
rownames(cumulative.species) <- NULL

# Write tsv file
fwrite(x=cumulative.species, file=paste(TABLES.CUMULATIVE.SPECIES,
                                        "/","Cumulative_species",
                                        ".tsv", sep = ""),
       sep = "\t",
       row.names = F)

```

```r
}

create.table.detailed <- function(table.name){
  #' Create Tables detailed
  #'
  #' @description This function creates a detailed version
  #' of the original one.
  #'
  #' It creates a table where reservoir, season and species are the columns.
  #'
  #' @param table.name dataframe Where the indexes are species and
  #' columns are reservoirs.
  #' @usage create.table.detailed(table.name)
  #' @return It does not return anything.
  #' @details The input must be a dataframe table.
  #' @examples
  #' create.table.detailed(table.name)

  # Get table without Unassigned, Assigned and Total
  table = get(table.name)[! row.names(get(table.name)) %in% c("Unassigned", "Assigned
      ↪ ", "Total"),]

  # Create vector with the combination of the reservoir and the season
  reservoirs.seasons = c("KS", "KW", "RS", "RW", "ZS", "ZW")

  for (reservoir.season in reservoirs.seasons){

    # Sum up columns matching reservoir.season variable pattern
    row.sums = rowSums(table[ , grepl(reservoir.season, colnames(table))])

    # Remove columns matching reservoir.season variable pattern
    table = table[ , ! grepl(reservoir.season, colnames(table))]

    # Add column where the name is the pattern from reservoir.season variable
    table[, reservoir.season] = row.sums
  }

  # Transpose Dataframe
  table = as.data.frame(t(table))

  # Reorder dataframe by columns
  table = table[,order(colnames(table))]

  # Create new column season
  seasons = c("Summer", "Autumn")
  table = cbind(Season = seasons, table)
```

```
559
560   # Create new column Reservoir
561   reservoirs = c("Klíčava", "Klíčava", "Římov", "Římov", "Žlutice", "Žlutice")
562   table = cbind(Reservoir = reservoirs, table)
563
564   # Assign to a global variable
565   assign(table.name, table, envir = .GlobalEnv)
566
567   # Write tsv file
568   fwrite(x=table, file=paste(TABLES.DETAILED, "/",table.name, ".tsv", sep = ""),
569          sep = "\t", row.names = F)
570
571 }
572
573 keep.only.controls <- function(table){
574   #' Keep only Controls
575   #'
576   #' @description This function removes no controls from table
577   #' and keep only controls.
578   #'
579   #' It removes not control and keep D, E, F, N, P, and de.
580   #'
581   #' @param table dataframe where the indexes are species and
582   #' columns are reservoirs.
583   #' @usage keep.only.controls(table)
584   #' @return It does not return anything.
585   #' @details The input must be a dataframe table.
586   #' @examples
587   #' keep.only.controls(table)
588
589   # Collect control names
590   control.names = colnames(get(table)[, grepl("\\d", colnames(get(table)))])
591
592   # Remove control columns
593   assign(table, get(table)[,!(colnames(get(table)) %in% control.names)], envir =
      ↪ .GlobalEnv)
594
595   # Write tsv file
596   fwrite(x=get(table), file=paste(TABLES.ONLY.CONTROLS, "/",table, ".tsv", sep = ""),
597          sep = "\t", row.names = T)
598 }
599
600 create.table.positive.control <- function(){
601   #' Create Table All Pipelines only positive control
602   #'
603   #' @description This function creates a table with only
```

```
  #' positive control using all pipelines.
  #'
  #' @usage create.table.positive.control()
  #' @return It does not return anything.
  #' @examples
  #' create.table.positive.control()

  # Remove TAX column and rows Total, Assigned and Unassigned
  table = all.pipelines
  table$TAX = NULL
  table = table[!(row.names(table) %in% c("Assigned", "Unassigned")), ]
  table = as.data.frame(t(table))

  # Keep only Mayland Zebra
  table = table[table["Maylandia zebra"] > 0, c("Total", "Maylandia zebra")]

  # Correction of row names
  rownames(table) = gsub(" P ", " ", rownames(table))

  # Calculate percentage
  table["Percentage_of_the_total"] = round(table["Maylandia zebra"] / table["Total"] *
      ↪  100, digits = 2)

  # Transform the name of the rows as the first column
  table = tibble::rownames_to_column(table, "Reservoir_Season_Pipeline")
  rownames(table) = table$Reservoir_Season_Pipeline
  table = table %>% separate(Reservoir_Season_Pipeline, c("Reservoir", "Season", "
      ↪ Pipeline"), " ")

  # Assign table name
  assign("all.pipelines.only.positive.control", table, envir = .GlobalEnv)

  # Write tsv file
  file = paste(TABLES.ONLY.CONTROLS, "/All_Pipelines_only_positive_control.csv", sep
      ↪ = "")
  fwrite(x=table, file=file, sep = "\t", row.names = T)

}

remove.controls <- function(table){
  #' Remove Controls
  #'
  #' @description This function removes controls from table.
  #'
  #' It removes index Mayland zebra and columns D, E, F, N, P, and de.
  #'
```

```
  #' @param table dataframe Where the indexes are species and
  #' columns are reservoirs.
  #' @usage remove.controls(table)
  #' @return It does not return anything.
  #' @details The input must be a dataframe table.
  #' @examples
  #' remove.controls(table)

  # Remove Maylandia zebra
  assign(table, get(table)[!(row.names(get(table)) %in% "Maylandia zebra"),], envir =
      ↪  .GlobalEnv)

  # Collect control names
  control.names = colnames(get(table)[,! grepl("\\d", colnames(get(table)))])

  # Remove control columns
  assign(table, get(table)[,!(colnames(get(table)) %in% control.names)], envir =
      ↪ .GlobalEnv)

  # Write tsv file
  fwrite(x=get(table), file=paste(TABLES.CONTROLS, "/",table, ".tsv", sep = ""),
         sep = "\t", row.names = T)
}

read.tables <- function(tables.path){
  #' Read Tables
  #'
  #' @description This function reads all tables from the vector
  #'
  #' From a vector of table paths, It reads each tsv file
  #' and creates a dataframe for each one.
  #'
  #' The name of each variable is the name of each file without
  #' extension.
  #'
  #' @param tables.path vector. A vector with paths to tables.
  #' @usage read.tables(tables.path)
  #' @return Return a vector with the names of the dataframes
  #' for each table.
  #' @details The input must be a vector with the paths to the tables
  #' in tsv format.
  #' @examples
  #' read.tables(tables.path)
  #' read.tables("PATH_TO_TABLE")

  # Initialize tables variable
```

```
  tables = c()

  # For each file
  for (file.name in tables.path) {

    # Parse base name and dir name
    base.name = basename(file.name)
    dir.name = dirname(file.name)

    # File name without extension
    variable.name = tools::file_path_sans_ext(base.name)

    # Read table and assign it to the variable.name
    assign(variable.name, read.csv(file.name, sep = '\t', header=T, row.names = 1,
        ↪ check.names=F), envir = .GlobalEnv)

    # Add new element to the list of tables
    tables = c(tables, variable.name)
  }

  return(tables)
}
```

Source Code A.17: Functions to create tables.

```
source("Create_Tables_Utils.R")
library(rChoiceDialogs)

# Number of the folder
folder.number = 3

# Folder variables
TABLES.JOINED = paste(auto.increment(folder.number), "_Joined", sep = "")
TABLES.JOINED.REDUCED = paste(auto.increment(folder.number), "_Joined_Reduced", sep =
    ↪  "")
#TABLES.RANACAPA = paste(auto.increment(folder.number), "_Ranacapa", sep = "")
TABLES.CONTROLS = paste(auto.increment(folder.number), "_No_Controls", sep = "")
TABLES.ONLY.CONTROLS = paste(auto.increment(folder.number), "_Only_Controls", sep = "
    ↪ ")
TABLES.DETAILED = paste(auto.increment(folder.number), "_Detailed", sep = "")
TABLES.CUMULATIVE.SPECIES = paste(auto.increment(folder.number), "_Cumulative_Species
    ↪ ", sep = "")
TABLES.CUMULATIVE.READS = paste(auto.increment(folder.number), "_Cumulative_Reads",
    ↪ sep = "")

```

```r
# Set working directory to source file location
if(Sys.getenv("RSTUDIO") == "1"){
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
}else{
  setwd(utils::getSrcDirectory()[1])
}

# Read files
# Get table file names
tables.path = list.files(rchoose.dir(caption = "Tables directory"), pattern = "*.tsv",
    ↪  full.names = TRUE ,recursive = TRUE)

# Read tables
tables = read.tables(tables.path)

# Create folder to save the new file
dir.create(TABLES.JOINED, showWarnings = FALSE)
# Create table joined
create.table.joined(tables)

# Create folder to save the new files after removing controls
dir.create(TABLES.ONLY.CONTROLS, showWarnings = FALSE)
# Keep controls
# for (table in tables) {
# keep.only.controls(table)
# }
# Keep only positive control
create.table.positive.control()

# Create folder to save the new file
dir.create(TABLES.JOINED.REDUCED, showWarnings = FALSE)
# Create table joined reduced
create.table.joined.reduced(tables)

# Before the execution of ranacapa code below, modify the line:
# new.name = paste(info, collapse = " ")
# by
# new.name = paste(info, collapse = ".")

# Create folder to save the new file
#dir.create(TABLES.RANACAPA, showWarnings = FALSE)
# Create tables ranacapa
#create.tables.ranacapa(all.pipelines.reduced.no.controls, "ranacapa", T)

# Create folder to save the new files after removing controls
dir.create(TABLES.CONTROLS, showWarnings = FALSE)
```

```
# Remove controls
for (table in tables) {
  remove.controls(table)
}

# Create folder to save the new files
dir.create(TABLES.DETAILED, showWarnings = FALSE)
# Create detailed version of the tables
for (table.name in tables) {
  create.table.detailed(table.name)
}

# Create folder to save the new file
dir.create(TABLES.CUMULATIVE.SPECIES, showWarnings = FALSE)
# Create tables cumulative species
create.table.cumulative.species(tables)

# Create folder to save the new file
dir.create(TABLES.CUMULATIVE.READS, showWarnings = FALSE)
# Create tables cumulative reads
create.table.cumulative.reads(tables)

# Create table all pipelines detailed
create.tables.all.pipelines.detailed()

# Save R object
save.image("Tables.RData")
```

Source Code A.18: Create tables. All tables for each pipeline are joined in a unique file, the control samples are removed and new tables are created, a table with only control sample is created, a detailed version of the joined table is also created, and cumulative tables are created.

```
library(tidyverse)
library(ggplot2)
library(colorblindr)
library(tikzDevice)
library(xtable)
library(reshape2)
library(scales)
library(ggpubr)
library(gridExtra)

# Load R object
load("Tables.RData")

```

```r
# Ignore scientific notation
options(scipen=10000)

# Read table
# dat = read.csv("Tables/All_Pipelines_detailed.csv", sep = '\t', header=T, row.names = 1, check.names=F)
dat = all.pipelines.detailed

# Species order
species_order = c("Lampetra planeri", "Acipenser-sp.", "Anguilla anguilla", "Phoxinus phoxinus", "Rutilus rutilus", "Chondrostoma nasus", "Squalius cephalus", "Alburnus alburnus", "Blicca+Vimba", "Abramis brama", "L.idus+leuciscus", "Aspius+Scardinius", "Pseudorasbora parva", "Gobio gobio", "Rhodeus amarus", "Tinca tinca", "Hypophthalmichthys molitrix", "Hypophthalmichthys nobilis", "Ctenopharyngodon idella", "Barbus barbus", "Cyprinus carpio", "Carassius auratus", "Carassius carassius", "Barbatula barbatula", "Esox lucius", "Gasterosteus aculeatus", "Sander+Perca", "Gymnocephalus cernua", "Lepomis gibbosus", "Thymallus thymallus", "Oncorhynchus mykiss", "Salvelinus fontinalis", "Salmo trutta", "Coregonus-sp.", "Cottus gobio", "Cottus poecilopus", "Silurus glanis")
species_order = rev(species_order)

#### Count number of species/reads for each pipeline, reservoir, and season.

# Number of reads

# Average of all pipelines
ignored = dat[ , !(colnames(dat) %in% c("Season", "Reservoir"))]
ignored = aggregate(. ~ Pipeline, data=ignored, FUN=sum)
ignored$Total = rowSums(ignored[, !(colnames(ignored) %in% c("Pipeline"))])
sum(ignored$Total) / length(unique(dat$Pipeline))


# Number of reads

# Pipeline
ignored = dat[ , !(colnames(dat) %in% c("Season", "Reservoir"))]
ignored = aggregate(. ~ Pipeline, data=ignored, FUN=sum)
ignored$Total = rowSums(ignored[, !(colnames(ignored) %in% c("Pipeline"))])
ignored[c("Pipeline", "Total")]

# Export as latex table
print(xtable(ignored[c("Pipeline", "Total")]), booktabs=TRUE, file = "Number_of_reads_by_Pipeline.tex")

# Reservoir
```

```r
ignored = dat[ , !(colnames(dat) %in% c("Pipeline", "Season"))]
ignored = aggregate(. ~ Reservoir, data=ignored, FUN=sum)
ignored$Total = rowSums(ignored[, !(colnames(ignored) %in% c("Reservoir"))])
# Use the code below to calculate the average for Reservoir or Season
ignored$Total = ignored$Total / length(unique(dat$Pipeline))
ignored[c("Reservoir", "Total")]

# Season
ignored = dat[ , !(colnames(dat) %in% c("Pipeline", "Reservoir"))]
ignored = aggregate(. ~ Season, data=ignored, FUN=sum)
ignored$Total = rowSums(ignored[, !(colnames(ignored) %in% c("Season"))])
# Use the code below to calculate the average for Reservoir and Season
ignored$Total = ignored$Total / length(unique(dat$Pipeline))
ignored[c("Season", "Total")]

# Number of reads for pipeline, reservoir, and season together
total = dat
total$total = rowSums(total[, !(colnames(total) %in% c("Pipeline", "Reservoir", "
    ↪ Season"))])
total = total[c("Pipeline", "Reservoir", "Season", "total")]
total
total[total$total == min(total$total),]
total[total$total == max(total$total),]

# Number of species

# Total
dim(dat[! colnames(dat) %in% c("Pipeline", "Reservoir", "Season")])

# Pipeline
ignored = dat[ , !(colnames(dat) %in% c("Reservoir", "Season"))]
ignored = aggregate(. ~ Pipeline, data=ignored, FUN=sum)
ignored$Total = rowSums(ignored[, !(colnames(ignored) %in% c("Pipeline"))] != 0)
ignored[c("Pipeline", "Total")]

# Export as latex table
#print(xtable(ignored[c("Pipeline", "Total")]), booktabs=TRUE, file = "Number_of_
    ↪ Species.tex")

# Reservoir
ignored = dat[ , !(colnames(dat) %in% c("Pipeline", "Season"))]
ignored = aggregate(. ~ Reservoir, data=ignored, FUN=sum)
ignored$Total = rowSums(ignored[, !(colnames(ignored) %in% c("Reservoir"))] != 0)
ignored[c("Reservoir", "Total")]

# Season
```

```
ignored = dat[ , !(colnames(dat) %in% c("Pipeline", "Reservoir"))]
ignored = aggregate(. ~ Season, data=ignored, FUN=sum)
ignored$Total = rowSums(ignored[, !(colnames(ignored) %in% c("Season"))] != 0)
ignored[c("Season", "Total")]

# Number of species for pipeline, reservoir, and season together
total = dat
total$total = rowSums(total[, !(colnames(total) %in% c("Pipeline", "Reservoir", "
    ↪ Season"))] != 0)
total = total[c("Pipeline", "Reservoir", "Season", "total")]
total
total[total$total == min(total$total),]
total[total$total == max(total$total),]

### Average number of reads per species

number_of_reads = colSums(dat[ , !(colnames(dat) %in% c("Reservoir", "Season", "
    ↪ Pipeline"))])
# Where 1 means to apply FUN to each row of df, 2 would mean to apply FUN to columns.
# min_of_reads = apply(dat[ , !(colnames(dat) %in% c("Reservoir", "Season", "Pipeline
    ↪ "))], 2, FUN=min)
st_dev = sapply(dat[ , !(colnames(dat) %in% c("Reservoir", "Season", "Pipeline"))],
    ↪ sd)
number_of_reads = round(number_of_reads / 5)
number_of_reads = data.frame(number_of_reads, st_dev)
number_of_reads$Species = rownames(number_of_reads)
number_of_reads[order(number_of_reads$number_of_reads),]

maximum = max(number_of_reads$number_of_reads)

# Create plots
p = ggplot(number_of_reads, aes(x = factor(Species, level = species_order), y =
    ↪ number_of_reads)) +
  geom_errorbar(aes(ymin = number_of_reads-st_dev, ymax = number_of_reads+st_dev),
      ↪ alpha = 0.75) +
  geom_point(aes(fill = number_of_reads), shape = 21, size = 2.5) +
  xlab(NULL) +
  ylab(NULL) +
  theme(axis.text.x = element_text(angle = 0, size = 8, color = "black",
                                   vjust = 1, hjust = 0.5),
        axis.text.y = element_text(size = 8, color = "black", face = "italic"),
        axis.title = element_text(size = 10, face = "plain"),
        plot.margin = margin(10, 10, 10, 15),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
```

```
132         panel.grid.major = element_line(colour = "lightgray", size = 0.3),
133         panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
134         panel.border = element_rect(colour = "black", fill = NA, size = 0.5),
135         legend.position = "none"
136   ) + coord_flip() +
137   scale_fill_gradientn(limits = c(1, maximum),
138                        expand = c(0, 0),
139                        colors = rev(c("darkred", "red", "orange", "yellow", "green", "
                               ↪ lightgreen", "lightblue", "darkblue")),
140                        name = NULL)
141
142 plog = ggplot(number_of_reads, aes(x = factor(Species, level = species_order), y =
        ↪ number_of_reads)) +
143   geom_errorbar(aes(ymin = number_of_reads-st_dev, ymax = number_of_reads+st_dev),
        ↪ alpha = 0.75) +
144   geom_point(aes(fill = number_of_reads), shape = 21, size = 2.5) +
145   xlab(NULL) +
146   ylab(NULL) +
147   theme(axis.text.x = element_text(angle = 0, size = 8, color = "black",
148                                    vjust = 1, hjust = 0.5),
149         axis.text.y = element_blank(),
150         axis.title = element_text(size = 10, face = "plain"),
151         plot.margin = margin(10, 10, 10, 0),
152         plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
153                                   margin = margin(10, 0, 10, 0)),
154         panel.background = element_rect(fill = 'white'),
155         panel.grid.major = element_line(colour = "lightgray", size = 0.3),
156         panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
157         panel.border = element_rect(colour = "black", fill = NA, size = 0.5),
158         legend.position = "none"
159   ) + scale_y_continuous(trans='log2') + coord_flip() +
160   scale_fill_gradientn(limits = c(1, maximum),
161                        expand = c(0, 0),
162                        colors=rev(c("darkred", "red", "orange", "yellow", "green", "
                               ↪ lightgreen", "lightblue", "darkblue")),
163                        name = NULL)
164
165 # Add label on the right side of the plots
166 p = grid.arrange(p, top = text_grob("No Transformation", hjust = -0.25, vjust = 1.5,
        ↪ just = "centre", size = 8, face = "plain"))
167 plog = grid.arrange(plog, top = text_grob("Logarithm 2", hjust = 0.5, vjust = 1.5,
        ↪ just = "centre", size = 8, face = "plain"))
168
169 # Join plots
170 figure = ggarrange(p, plog, ncol = 2, nrow = 1, widths=c(1.65,1))
171 #figure = annotate_figure(figure, bottom = text_grob("Number of reads", hjust = 0,
```

```
    ↪ size = 10, face = "plain"))

# Export ggplot to Latex
# factor(Species, level = species_order) to change the order based on phylogenetic
    ↪ tree
tikz(file = "Number_of_reads_average_by_species.tex", width = 6, height = 5.5)
figure
dev.off()


### Number of reads per species considering Pipeline
ignored = dat[ , !(colnames(dat) %in% c("Reservoir", "Season"))]
ignored = aggregate(. ~ Pipeline, data=ignored, FUN=sum)

# For each pipeline
rownames(ignored) = ignored$Pipeline
pipelines = unique(ignored$Pipeline)
#ignored$Pipeline = NULL
for (pipeline in pipelines) {
  print(sort(ignored[pipeline, !(colnames(ignored) %in% "Pipeline")]))
  print("#################")
}


# Export ggplot to Latex

# Convert to format of GGplot
number_of_reads = melt(ignored, id.vars='Pipeline')

# Calculate breaks
maximum = max(number_of_reads$value)
breaks = c(maximum)
while (tail(breaks, n=1) != 0) {
  breaks = c(breaks, round(tail(breaks, n=1) / 4))
}
breaks = breaks[-length(breaks)]

# Convert zero to NA
number_of_reads[number_of_reads == 0] = NA

tikz(file = "Number_of_reads_by_pipelines_and_species.tex", width = 6.25, height = 5
    ↪ .5)
ggplot(number_of_reads, aes(y=Pipeline, x=factor(variable, level = species_order),
    ↪ fill=value)) +
  geom_tile(color = "black", size = 0.5) +
  xlab(NULL) +
```

```
  ylab(NULL) +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 8, color = "black", face = "italic"),
        axis.title = element_text(size = 10, face = "plain"),
        axis.ticks.x = element_blank(),
        plot.margin = margin(10, 10, 10, 20),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'lightgray'),
        panel.grid.major = element_line(colour = "white", size = 0.3),
        panel.grid.minor = element_line(colour = "white", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5),
        legend.key.size = unit(1, "cm"),
        legend.position = "right",
        legend.text = element_text(angle = 0, size = 6, color = "black",
                                   face = "plain", vjust = 1, hjust = 1),
        strip.text = element_text(size = 8, face = "plain", color = "black"),
        strip.background = element_blank()
  ) +
  scale_x_discrete(expand = c(0, 0)) +
  scale_y_discrete(expand = c(0, 0)) +
  scale_fill_gradientn(limits=c(1,maximum),
                       breaks = breaks,
                       expand = c(0,0),
                       colors=rev(c("darkred", "red", "orange", "yellow", "green", "
                           ↪ lightgreen", "lightblue", "darkblue")),
                       na.value = 'white',
                       name = NULL,
                       trans = pseudo_log_trans(base = 2)) +
  coord_flip() + facet_grid(~Pipeline, scales='free')
dev.off()
# Because trans = is applying log2, so breaks values are 2**number (exponential of 2)


### Number of reads by species considering Reservoir
ignored = dat[ , !(colnames(dat) %in% c("Pipeline", "Season"))]
ignored = aggregate(. ~ Reservoir, data=ignored, FUN=sum)

# For each reservoir
rownames(ignored) = ignored$Reservoir
reservoirs = unique(ignored$Reservoir)
#ignored$Reservoir = NULL
for (reservoir in reservoirs) {
  print(sort(round(ignored[reservoir, !(colnames(ignored) %in% "Reservoir")] / length
      ↪ (unique(dat$Pipeline)))))
  print("#################")
```

```
}


# Export ggplot to Latex

# Convert to format of GGplot
number_of_reads = melt(ignored, id.vars='Reservoir')
number_of_reads$value = round(number_of_reads$value / 5)

# Calculate breaks
maximum = max(number_of_reads$value)
breaks = c(maximum)
while (tail(breaks, n=1) != 0) {
  breaks = c(breaks, round(tail(breaks, n=1) / 4))
}
breaks = breaks[-length(breaks)]

# Convert zero to NA
number_of_reads[number_of_reads == 0] = NA

tikz(file = "Number_of_reads_by_reservoirs_and_species.tex", width = 6.25, height = 5
    ↪ .5)
ggplot(number_of_reads, aes(y=Reservoir, x=factor(variable, level = species_order),
    ↪ fill=value)) +
  geom_tile(color = "black", size = 0.5) +
  xlab(NULL) +
  ylab(NULL) +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 8, color = "black", face = "italic"),
        axis.title = element_text(size = 10, face = "plain"),
        axis.ticks.x = element_blank(),
        plot.margin = margin(10, 10, 10, 20),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'lightgray'),
        panel.grid.major = element_line(colour = "white", size = 0.3),
        panel.grid.minor = element_line(colour = "white", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5),
        legend.key.size = unit(1, "cm"),
        legend.position = "right",
        legend.text = element_text(angle = 0, size = 6, color = "black",
                                   face = "plain", vjust = 1, hjust = 1),
        strip.text = element_text(size = 8, face = "plain", color = "black"),
        strip.background = element_blank()
  ) +
  scale_x_discrete(expand = c(0, 0)) +
```

```
  scale_y_discrete(expand = c(0, 0)) +
  scale_fill_gradientn(limits=c(1,maximum),
                       breaks = breaks,
                       expand = c(0,0),
                       colors=rev(c("darkred", "red", "orange", "yellow", "green", "
                           ↪ lightgreen", "lightblue", "darkblue")),
                       na.value = 'white',
                       name = NULL,
                       trans = pseudo_log_trans(base = 2)) +
  coord_flip() + facet_grid(~Reservoir, scales='free')
dev.off()
# Because trans = is applying log2, so breaks values are 2**number (exponential of 2)


### Number of reads per species considering Season
ignored = dat[ , !(colnames(dat) %in% c("Reservoir", "Pipeline"))]
ignored = aggregate(. ~ Season, data=ignored, FUN=sum)

# For each Season
rownames(ignored) = ignored$Season
seasons = unique(ignored$Season)
#ignored$Season = NULL
for (season in seasons) {
  print(sort(round(ignored[season, !(colnames(ignored) %in% "Season")] / length(
      ↪ unique(dat$Pipeline)))))
  print("#################")
}


# Export ggplot to Latex

# Convert to format of GGplot
number_of_reads = melt(ignored, id.vars='Season')
number_of_reads$value = round(number_of_reads$value / 5)

# Calculate breaks
maximum = max(number_of_reads$value)
breaks = c(maximum)
while (tail(breaks, n=1) != 0) {
  breaks = c(breaks, round(tail(breaks, n=1) / 4))
}
breaks = breaks[-length(breaks)]

# Convert zero to NA
number_of_reads[number_of_reads == 0] = NA

```

```
tikz(file = "Number_of_reads_by_seasons_and_species.tex", width = 6.25, height = 5.5)
ggplot(number_of_reads, aes(y=Season, x=factor(variable, level = species_order), fill
    ↪ =value)) +
  geom_tile(color = "black", size = 0.5) +
  xlab(NULL) +
  ylab(NULL) +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 8, color = "black", face = "italic"),
        axis.title = element_text(size = 10, face = "plain"),
        axis.ticks.x = element_blank(),
        plot.margin = margin(10, 10, 10, 20),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'lightgray'),
        panel.grid.major = element_line(colour = "white", size = 0.3),
        panel.grid.minor = element_line(colour = "white", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5),
        legend.key.size = unit(1, "cm"),
        legend.position = "right",
        legend.text = element_text(angle = 0, size = 6, color = "black",
                                   face = "plain", vjust = 1, hjust = 1),
        strip.text = element_text(size = 8, face = "plain", color = "black"),
        strip.background = element_blank()
  ) +
  scale_x_discrete(expand = c(0, 0)) +
  scale_y_discrete(expand = c(0, 0)) +
  scale_fill_gradientn(limits=c(1,maximum),
                       breaks = breaks,
                       expand = c(0,0),
                       colors=rev(c("darkred", "red", "orange", "yellow", "green", "
                           ↪ lightgreen", "lightblue", "darkblue")),
                       na.value = 'white',
                       name = NULL,
                       trans = pseudo_log_trans(base = 2)) +
  coord_flip() + facet_grid(~Season, scales='free')
dev.off()
# Because trans = is applying log2, so breaks values are 2**number (exponential of 2)


### Number of reads by species considering pipeline, reservoir, Season
df = dat

# Remove columns and create a new column with row names
df$Pipeline = NULL
df$Reservoir = NULL
df$Season = NULL
```

```
df$Pipeline_Reservoir_Season = rownames(df)

# Export ggplot to Latex

# Convert to format of GGplot
number_of_reads = melt(df, id.vars='Pipeline_Reservoir_Season')

# Sort and check in the file created the largest and smallest value
write.csv(number_of_reads[order(number_of_reads$value),],
          file = "LS.csv", row.names = FALSE)

# Calculate breaks
maximum = max(number_of_reads$value)
breaks = c(maximum)
while (tail(breaks, n=1) != 0) {
  breaks = c(breaks, round(tail(breaks, n=1) / 4))
}
breaks = breaks[-length(breaks)]

# Convert zero to NA
number_of_reads[number_of_reads == 0] = NA
number_of_reads = number_of_reads %>% separate(Pipeline_Reservoir_Season, c("Pipeline
    ↪ ", "Reservoir", "Season"), " ")

tikz(file = "Number_of_reads_by_PRS_and_species.tex", width = 6, height = 9.5)
ggplot(number_of_reads, aes(y=Season, x=factor(variable, level = species_order), fill
    ↪ =value)) +
  geom_tile(color = "black", size = 0.5) +
  xlab(NULL) +
  ylab(NULL) +
  theme(axis.text.x = element_text(angle = 45, size = 8, color = "black",
                                   vjust = 1, hjust = 1),
        axis.text.y = element_text(size = 8, color = "black", face = "italic"),
        axis.title = element_text(size = 10, face = "plain"),
        plot.margin = margin(10, 10, 10, 20),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'lightgray'),
        panel.grid.major = element_line(colour = "white", size = 0.3),
        panel.grid.minor = element_line(colour = "white", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5),
        legend.key.size = unit(1, "cm"),
        legend.position = "right",
        legend.text = element_text(angle = 0, size = 6, color = "black",
                                   face = "plain", vjust = 1, hjust = 1),
        strip.text = element_text(size = 8, face = "plain", color = "black")
```

```
  ) +
  scale_x_discrete(expand = c(0, 0)) +
  scale_y_discrete(expand = c(0, 0)) +
  scale_fill_gradientn(limits=c(1,maximum),
                       breaks = breaks,
                       expand = c(0,0),
                       colors=rev(c("darkred", "red", "orange", "yellow", "green", "
                           ↪ lightgreen", "lightblue", "darkblue")),
                       na.value = 'white',
                       name = NULL,
                       trans = pseudo_log_trans(base = 2)) +
  coord_flip() + facet_grid(Reservoir~Pipeline)
dev.off()
# Because trans = is applying log2, so breaks values are 2**number (exponential of 2)


### Species detected only once or in all pipelines

# Get data and ignore Reservoir and Season columns
ignored = dat[ , !(colnames(dat) %in% c("Reservoir", "Season"))]
# Sum up values by Pipeline
ignored = aggregate(. ~ Pipeline, data=ignored, FUN=sum)
# Make rownames the values in Pipeline column
rownames(ignored) = ignored$Pipeline
# Check if values are different from zero
ignored = ignored[, !(colnames(ignored) %in% c("Pipeline"))] != 0

# keep only columns summing up 1
names_only_once = colnames(ignored[,colSums(ignored) == 1])
# keep only columns summing up 5 (all pipelines)
names_in_all = colnames(ignored[,colSums(ignored) == 5])

# Show which pipeline detected species found just once
rowSums(ignored[,names_only_once])


### Species detected only once or in all reservoirs

# Get data and ignore Pipeline and Season columns
ignored = dat[ , !(colnames(dat) %in% c("Pipeline", "Season"))]
# Sum up values by Reservoir
ignored = aggregate(. ~ Reservoir, data=ignored, FUN=sum)
# Make rownames the values in Reservoir column
rownames(ignored) = ignored$Reservoir
# Check if values are different from zero
ignored = ignored[, !(colnames(ignored) %in% c("Reservoir"))] != 0
```

```r

# keep only columns summing up 1
names_only_once = colnames(ignored[,colSums(ignored) == 1])
# keep only columns summing up 3 (all reservoirs)
names_in_all = colnames(ignored[,colSums(ignored) == 3])

# Show which reservoir detected species found just once
rowSums(ignored[,names_only_once])


### Species detected only once or in all seasons

# Get data and ignore Pipeline and Reservoir columns
ignored = dat[ , !(colnames(dat) %in% c("Pipeline", "Reservoir"))]
# Sum up values by Season
ignored = aggregate(. ~ Season, data=ignored, FUN=sum)
# Make rownames the values in Season column
rownames(ignored) = ignored$Season
# Check if values are different from zero
ignored = ignored[, !(colnames(ignored) %in% c("Season"))] != 0

# keep only columns summing up 1
names_only_once = colnames(ignored[,colSums(ignored) == 1])
# keep only columns summing up 2 (all seasons)
names_in_all = colnames(ignored[,colSums(ignored) == 2])

# Show which season detected species found just once
rowSums(ignored[,names_only_once])


### Species detected only once or in pipelines, reservoirs and seasons together

df = dat

# Remove columns
df$Pipeline = NULL
df$Reservoir = NULL
df$Season = NULL

df = df != 0

# keep only columns summing up 1
names_only_once = colnames(df[,colSums(df) == 1])
# keep only columns summing up 30 (all combination)
names_in_all = colnames(df[,colSums(df) == 30])

```

```
# Show which pipeline,reservoir and season detected species found just once
rowSums(df[,names_only_once])
```

Source Code A.19: Calculate the number of reads and species, and create charts for pipelines, reservoirs, and season.

```
library(vegan)
library(tidyr)
library(dplyr)
library(ggplot2)
library(colorblindr)
library(tidyverse)
library(tikzDevice)
library(xtable)
library(reshape2)
library(scales)
library(dataMaid)
library(nortest)
library(xtable)


# Set working directory to source file location
if(Sys.getenv("RSTUDIO") == "1"){
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
}else{
  setwd(utils::getSrcDirectory()[1])
}

# Load data
load("Tables.RData")

##### Statistical analyses using all.pipelines.detailed #####

### ALPHA ###

# Remove columns Reservoir, Season and Pipeline
statistic = all.pipelines.detailed[,! colnames(all.pipelines.detailed) %in% c("
    ↪ Reservoir", "Season", "Pipeline")]

# Alpha diversity: observed species
all.pipelines.detailed$Richness = rowSums(statistic > 0)

# Create column for Reservoir, Season and Pipeline together
all.pipelines.detailed = unite(all.pipelines.detailed, Reservoir_Season_Pipeline, c(
    ↪ Reservoir, Season, Pipeline), remove=F, sep=" ")
```

```
38
39 # Create column for Reservoir and Season together
40 all.pipelines.detailed = unite(all.pipelines.detailed, Reservoir_Season, c(Reservoir,
       ↪  Season), remove=F, sep=" ")
41
42 # Create column for Reservoir and Pipeline together
43 all.pipelines.detailed = unite(all.pipelines.detailed, Reservoir_Pipeline, c(
       ↪ Reservoir, Pipeline), remove=F, sep=" ")
44
45 # Create column for Season and Pipeline together
46 all.pipelines.detailed = unite(all.pipelines.detailed, Season_Pipeline, c(Season,
       ↪ Pipeline), remove=F, sep=" ")
47
48 # Create column with unique name
49 all.pipelines.detailed$All = "All"
50
51 ### Observed species ###
52
53 # Export ggplot to Latex
54 tikz(file = "Alpha_diversity_richness_PRS.tex", width = 6, height = 3)
55 # Plot Reservoir_Season_Pipeline
56 ggplot(all.pipelines.detailed, aes(x = Pipeline, y = Richness)) +
57   geom_point(aes(fill = Pipeline), shape = 21, size = 3) +
58   xlab(NULL) +
59   ylab("Species richness") +
60   theme(axis.text.x = element_blank(),
61         axis.ticks.x = element_blank(),
62         axis.text.y = element_text(size = 6, color = "black"),
63         axis.title = element_text(size = 10, face = "plain"),
64         legend.text = element_text(size = 6, color = "black"),
65         legend.title = element_text(size = 8, face = "plain"),
66         legend.margin = margin(t = 0, unit='cm'),
67         legend.key = element_rect(fill = NA, color = NA),
68         strip.text.x = element_text(size = 6, face = "plain", color = "black"),
69         strip.background = element_blank(),
70         #plot.margin = margin(10, 10, 10, 50),
71         plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
72                                   margin = margin(10, 0, 10, 0)),
73         panel.background = element_rect(fill = 'white'),
74         panel.grid.major = element_line(colour = "lightgray", size = 0.3),
75         panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
76         panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
77   ) + facet_grid(~Reservoir_Season, switch = "x") +
78   scale_fill_discrete(name = "Pipelines")
79 dev.off()
80
```

```
# Plot All
p = ggplot(all.pipelines.detailed, aes(y = Richness)) +
  stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(
      ↪ color = All)) +
  geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth
      ↪ = T, lwd = 1, aes(color = All, fill = All)) +
  xlab(NULL) +
  ylab("Species richness") +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
        axis.ticks.x = element_blank(),
        legend.text = element_text(size = 6, color = "black"),
        legend.title = element_text(size = 8, face = "plain"),
        legend.margin = margin(t = 0, unit='cm'),
        legend.key = element_rect(fill = NA, color = NA),
        strip.text.x = element_text(size = 8, face = "plain", color = "black"),
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'lightgray'),
        panel.grid.major = element_line(colour = "white", size = 0.3),
        panel.grid.minor = element_line(colour = "white", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + facet_grid(~All, switch="x")

# Calculate median and quartiles
ggplot_build(p)$data
min(all.pipelines.detailed$Richness)
max(all.pipelines.detailed$Richness)
median(all.pipelines.detailed$Richness)
quartiles(all.pipelines.detailed$Richness, maxDecimals = 0)
all.pipelines.detailed[all.pipelines.detailed$Richness == min(all.pipelines.detailed$
    ↪ Richness),]["Richness"]
all.pipelines.detailed[all.pipelines.detailed$Richness == max(all.pipelines.detailed$
    ↪ Richness),]["Richness"]

## Pipelines ##

# Plot Pipeline
p = ggplot(all.pipelines.detailed, aes(y = Richness)) +
  stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(
      ↪ color = Pipeline)) +
  geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth
      ↪ = T, lwd = 1, aes(color = Pipeline, fill = Pipeline)) +
```

```
  xlab(NULL) +
  ylab("Species richness") +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
        axis.ticks.x = element_blank(),
        legend.text = element_text(size = 6, color = "black"),
        legend.title = element_text(size = 8, face = "plain"),
        legend.margin = margin(t = 0, unit='cm'),
        legend.key = element_rect(fill = NA, color = NA),
        strip.text.x = element_text(size = 8, face = "plain", color = "black"),
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
        panel.grid.major = element_line(colour = "lightgray", size = 0.3),
        panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + facet_grid(~Pipeline, switch="x")

# Export ggplot to Latex
tikz(file = "Alpha_diversity_richness_pipelines.tex", width = 6, height = 3)
p
dev.off()

# Get min, max, median, and quartiles for each pipeline
ggplot_build(p)$data
# Get pipelines
pipelines = unique(all.pipelines.detailed$Pipeline)
for (pipeline in pipelines) {
  print(pipeline)
  # Extract pipeline
  pipeline = all.pipelines.detailed[all.pipelines.detailed$Pipeline == pipeline,]
  # Get min, max, median, and quartiles for each pipeline
  print(min(pipeline$Richness))
  print(max(pipeline$Richness))
  print(median(pipeline$Richness))
  print(quartiles(pipeline$Richness))
  print(pipeline[pipeline$Richness == min(pipeline$Richness),]["Richness"])
  print(pipeline[pipeline$Richness == max(pipeline$Richness),]["Richness"])
  print("------------------------------------------------------------")
  print("")
}

## Statistical analyses ##
```

```
167
168 # fit linear models
169 mod.richness = aov(Richness~Pipeline, data=all.pipelines.detailed)
170 # ANOVA
171 anova.test = anova(mod.richness)
172 anova.test
173 # Tukey
174 tukey.test = TukeyHSD(mod.richness)
175 tukey.test
176
177 # Check if p-value < 0.05
178 tukey.test = as.data.frame(tukey.test[["Pipeline"]])
179 tukey.test[tukey.test$`p adj` < 0.05,]
180
181 # Export as latex tables
182 print(xtable(anova.test, digits = c(0, 0, 2, 3, 4, 4)), booktabs=TRUE, file = "Alpha_
    ↪ Diversity_Richness_ANOVA_Pipelines.tex")
183 print(xtable(tukey.test$Pipeline, digits = c(0, 4, 4, 4, 4)), booktabs=TRUE, file = "
    ↪ Alpha_Diversity_Richness_Tukey_Pipelines.tex")
184
185 ## Reservoirs ##
186
187 # Plot Reservoir
188 p = ggplot(all.pipelines.detailed, aes(y = Richness)) +
189   stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(
      ↪ color = Reservoir)) +
190   geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth
      ↪ = T, lwd = 1, aes(color = Reservoir, fill = Reservoir)) +
191   xlab(NULL) +
192   ylab("Species richness") +
193   theme(axis.text.x = element_blank(),
194         axis.text.y = element_text(size = 6, color = "black"),
195         axis.title = element_text(size = 10, face = "plain"),
196         axis.ticks.x = element_blank(),
197         legend.text = element_text(size = 6, color = "black"),
198         legend.title = element_text(size = 8, face = "plain"),
199         legend.margin = margin(t = 0, unit='cm'),
200         legend.key = element_rect(fill = NA, color = NA),
201         strip.text.x = element_text(size = 8, face = "plain", color = "black"),
202         strip.background = element_blank(),
203         #plot.margin = margin(10, 10, 10, 50),
204         plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
205                                   margin = margin(10, 0, 10, 0)),
206         panel.background = element_rect(fill = 'white'),
207         panel.grid.major = element_line(colour = "lightgray", size = 0.3),
208         panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
```

```
209         panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
210   ) + facet_grid(~Reservoir, switch="x")
211 
212 # Export ggplot to Latex
213 tikz(file = "Alpha_diversity_richness_reservoirs.tex", width = 6, height = 3)
214 p
215 dev.off()
216 
217 # Get min, max, median, and quartiles for each pipeline
218 ggplot_build(p)$data
219 # Get reservoirs
220 reservoirs = unique(all.pipelines.detailed$Reservoir)
221 for (reservoir in reservoirs) {
222   print(reservoir)
223   # Extract reservoir
224   reservoir = all.pipelines.detailed[all.pipelines.detailed$Reservoir == reservoir,]
225   # Get min, max, median, and quartiles for each reservoir
226   print(min(reservoir$Richness))
227   print(max(reservoir$Richness))
228   print(median(reservoir$Richness))
229   print(quartiles(reservoir$Richness))
230   print(reservoir[reservoir$Richness == min(reservoir$Richness),]["Richness"])
231   print(reservoir[reservoir$Richness == max(reservoir$Richness),]["Richness"])
232   print("------------------------------------------------------------")
233   print("")
234 }
235 
236 ## Statistical analyses ##
237 
238 # fit linear models
239 mod.richness = aov(Richness~Reservoir, data=all.pipelines.detailed)
240 # ANOVA
241 anova.test = anova(mod.richness)
242 anova.test
243 # Tukey
244 tukey.test = TukeyHSD(mod.richness)
245 tukey.test
246 
247 # Check if p-value < 0.05
248 tukey.test = as.data.frame(tukey.test[["Reservoir"]])
249 tukey.test[tukey.test$`p adj` < 0.05,]
250 
251 # Export as latex tables
252 print(xtable(anova.test, digits = c(0, 0, 2, 3, 4, 4)), booktabs=TRUE, file = "Alpha_
    ↪ Diversity_Richness_ANOVA_Reservoirs.tex")
253 print(xtable(tukey.test$Reservoir, digits = c(0, 1, 4, 4, 4)), booktabs=TRUE, file =
```

```
    ↪ "Alpha_Diversity_Richness_Tukey_Reservoirs.tex")

## Seasons ##

# Plot Pipeline
p = ggplot(all.pipelines.detailed, aes(y = Richness)) +
  stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(
      ↪ color = Season)) +
  geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth
      ↪ = T, lwd = 1, aes(color = Season, fill = Season)) +
  xlab(NULL) +
  ylab("Species richness") +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
        axis.ticks.x = element_blank(),
        legend.text = element_text(size = 6, color = "black"),
        legend.title = element_text(size = 8, face = "plain"),
        legend.margin = margin(t = 0, unit='cm'),
        legend.key = element_rect(fill = NA, color = NA),
        strip.text.x = element_text(size = 8, face = "plain", color = "black"),
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
        panel.grid.major = element_line(colour = "lightgray", size = 0.3),
        panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + facet_grid(~Season, switch="x")

# Export ggplot to Latex
tikz(file = "Alpha_diversity_richness_seasons.tex", width = 6, height = 3)
p
dev.off()

# Get min, max, median, and quartiles for each pipeline
ggplot_build(p)$data
# Get seasons
seasons = unique(all.pipelines.detailed$Season)
for (season in seasons) {
  print(season)
  # Extract season
  season = all.pipelines.detailed[all.pipelines.detailed$Season == season,]
  # Get min, max, median, and quartiles for each season
  print(min(season$Richness))
```

```
  print(max(season$Richness))
  print(median(season$Richness))
  print(quartiles(season$Richness))
  print(season[season$Richness == min(season$Richness),]["Richness"])
  print(season[season$Richness == max(season$Richness),]["Richness"])
  print("------------------------------------------------------------")
  print("")
}

# Fences
quartiles = quartiles(all.pipelines.detailed[all.pipelines.detailed$Season == "Summer
    ↪ ",]$Richness, maxDecimals = 0)
upperq = round(quartiles$value[["75%"]])
lowerq = round(quartiles$value[["25%"]])
iqr = upperq - lowerq
upper.fence = upperq + (1.5 * iqr)
lower.fence = lowerq - (1.5 * iqr)

## Statistical analyses ##

# fit linear models
mod.richness = aov(Richness~Season, data=all.pipelines.detailed)
# ANOVA
anova.test = anova(mod.richness)
anova.test
# T-test
t.test(Richness~Season, data=all.pipelines.detailed, var.equal = TRUE)
# Tukey
tukey.test = TukeyHSD(mod.richness)
tukey.test

# Check if p-value < 0.05
tukey.test = as.data.frame(tukey.test[["Season"]])
tukey.test[tukey.test$`p adj` < 0.05,]

# Export as latex tables
print(xtable(anova.test, digits = c(0, 0, 2, 3, 4, 4)), booktabs=TRUE, file = "Alpha_
    ↪ Diversity_Richness_ANOVA_seasons.tex")
```

Source Code A.20: Calculate alpha diversity species richness and create charts for pipelines, reservoirs, and season.

```
library(vegan)
library(tidyr)
library(dplyr)
```

```r
library(ggplot2)
library(colorblindr)
library(tidyverse)
library(tikzDevice)
library(xtable)
library(reshape2)
library(scales)
library(dataMaid)
library(nortest)


# Set working directory to source file location
if(Sys.getenv("RSTUDIO") == "1"){
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
}else{
  setwd(utils::getSrcDirectory()[1])
}

# Load data
load("Tables.RData")

##### Statistical analyses using all.pipelines.detailed #####

### ALPHA ###

# Remove columns Reservoir, Season and Pipeline
statistic = all.pipelines.detailed[,! colnames(all.pipelines.detailed) %in% c("
    ↪ Reservoir", "Season", "Pipeline")]

# Alpha diversity: shannon index
all.pipelines.detailed$Shannon = diversity(statistic)

# Create column for Reservoir, Season and Pipeline together
all.pipelines.detailed = unite(all.pipelines.detailed, Reservoir_Season_Pipeline, c(
    ↪ Reservoir, Season, Pipeline), remove=F, sep=" ")

# Create column for Reservoir and Season together
all.pipelines.detailed = unite(all.pipelines.detailed, Reservoir_Season, c(Reservoir,
    ↪  Season), remove=F, sep=" ")

# Create column for Reservoir and Pipeline together
all.pipelines.detailed = unite(all.pipelines.detailed, Reservoir_Pipeline, c(
    ↪ Reservoir, Pipeline), remove=F, sep=" ")

# Create column for Season and Pipeline together
all.pipelines.detailed = unite(all.pipelines.detailed, Season_Pipeline, c(Season,
```

```
    ↪ Pipeline), remove=F, sep=" ")

# Create column with unique name
all.pipelines.detailed$All = "All"

### Shannon index ###

# Export ggplot to Latex
tikz(file = "Alpha_diversity_shannon_PRS.tex", width = 6, height = 3)
# Plot Reservoir_Season_Pipeline
ggplot(all.pipelines.detailed, aes(x = Pipeline, y = Shannon)) +
  geom_point(aes(fill = Pipeline), shape = 21, size = 3) +
  xlab(NULL) +
  ylab("Shannon index") +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
        legend.text = element_text(size = 6, color = "black"),
        legend.title = element_text(size = 8, face = "plain"),
        legend.margin = margin(t = 0, unit='cm'),
        legend.key = element_rect(fill = NA, color = NA),
        strip.text.x = element_text(size = 6, face = "plain", color = "black"),
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
        panel.grid.major = element_line(colour = "lightgray", size = 0.3),
        panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + facet_grid(~Reservoir_Season, switch = "x") +
  scale_fill_discrete(name = "Pipelines")
dev.off()

# Plot All
p = ggplot(all.pipelines.detailed, aes(y = Shannon)) +
  stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(
      ↪ color = All)) +
  geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth
      ↪ = T, lwd = 1, aes(color = All, fill = All)) +
  xlab(NULL) +
  ylab("Shannon index") +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
```

```
89        axis.ticks.x = element_blank(),
90        legend.text = element_text(size = 6, color = "black"),
91        legend.title = element_text(size = 8, face = "plain"),
92        legend.margin = margin(t = 0, unit='cm'),
93        legend.key = element_rect(fill = NA, color = NA),
94        strip.text.x = element_text(size = 8, face = "plain", color = "black"),
95        strip.background = element_blank(),
96        #plot.margin = margin(10, 10, 10, 50),
97        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
98                                  margin = margin(10, 0, 10, 0)),
99        panel.background = element_rect(fill = 'lightgray'),
100       panel.grid.major = element_line(colour = "white", size = 0.3),
101       panel.grid.minor = element_line(colour = "white", size = 0.1),
102       panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
103   ) + facet_grid(~All, switch="x")
104
105 # Calculate median and quartiles
106 ggplot_build(p)$data
107 min(all.pipelines.detailed$Shannon)
108 max(all.pipelines.detailed$Shannon)
109 median(all.pipelines.detailed$Shannon)
110 quartiles(all.pipelines.detailed$Shannon)
111 all.pipelines.detailed[all.pipelines.detailed$Shannon == min(all.pipelines.detailed$
        ↪ Shannon),]["Shannon"]
112 all.pipelines.detailed[all.pipelines.detailed$Shannon == max(all.pipelines.detailed$
        ↪ Shannon),]["Shannon"]
113
114 ## Pipelines ##
115
116 # Plot Pipeline
117 p = ggplot(all.pipelines.detailed, aes(y = Shannon)) +
118   stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(
        ↪ color = Pipeline)) +
119   geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth
        ↪ = T, lwd = 1, aes(color = Pipeline, fill = Pipeline)) +
120   xlab(NULL) +
121   ylab("Shannon index") +
122   theme(axis.text.x = element_blank(),
123         axis.text.y = element_text(size = 6, color = "black"),
124         axis.title = element_text(size = 10, face = "plain"),
125         axis.ticks.x = element_blank(),
126         legend.text = element_text(size = 6, color = "black"),
127         legend.title = element_text(size = 8, face = "plain"),
128         legend.margin = margin(t = 0, unit='cm'),
129         legend.key = element_rect(fill = NA, color = NA),
130         strip.text.x = element_text(size = 8, face = "plain", color = "black"),
```

```
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
        panel.grid.major = element_line(colour = "lightgray", size = 0.3),
        panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + facet_grid(~Pipeline, switch="x")

# Export ggplot to Latex
tikz(file = "Alpha_diversity_shannon_pipelines.tex", width = 6, height = 3)
p
dev.off()

# Get min, max, median, and quartiles for each pipeline
ggplot_build(p)$data
# Get pipelines
pipelines = unique(all.pipelines.detailed$Pipeline)
for (pipeline in pipelines) {
  print(pipeline)
  # Extract pipeline
  pipeline = all.pipelines.detailed[all.pipelines.detailed$Pipeline == pipeline,]
  # Get min, max, median, and quartiles for each pipeline
  print(round(min(pipeline$Shannon), digits = 3))
  print(round(max(pipeline$Shannon), digits = 3))
  print(median(pipeline$Shannon))
  print(quartiles(pipeline$Shannon))
  print(pipeline[pipeline$Shannon == min(pipeline$Shannon),]["Shannon"])
  print(pipeline[pipeline$Shannon == max(pipeline$Shannon),]["Shannon"])
  print("------------------------------------------------------------")
  print("")
}

## Statistical analyses ##

# fit linear models
mod.Shannon = aov(Shannon~Pipeline, data=all.pipelines.detailed)
# ANOVA
anova.test = anova(mod.Shannon)
anova.test
# Tukey
tukey.test = TukeyHSD(mod.Shannon)
tukey.test

# Check if p-value < 0.05
```

```r
tukey.test = as.data.frame(tukey.test[["Pipeline"]])
tukey.test[tukey.test$`p adj` < 0.05,]

# Export as latex tables
print(xtable(anova.test, digits = c(0, 0, 4, 4, 4, 4)), booktabs=TRUE, file = "Alpha_Diversity_Shannon_ANOVA_Pipelines.tex")
print(xtable(tukey.test$Pipeline, digits = c(0, 4, 4, 4, 4)), booktabs=TRUE, file = "Alpha_Diversity_Shannon_Tukey_Pipelines.tex")

## Reservoirs ##

# Plot Reservoir
p = ggplot(all.pipelines.detailed, aes(y = Shannon)) +
  stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(color = Reservoir)) +
  geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth = T, lwd = 1, aes(color = Reservoir, fill = Reservoir)) +
  xlab(NULL) +
  ylab("Shannon index") +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
        axis.ticks.x = element_blank(),
        legend.text = element_text(size = 6, color = "black"),
        legend.title = element_text(size = 8, face = "plain"),
        legend.margin = margin(t = 0, unit='cm'),
        legend.key = element_rect(fill = NA, color = NA),
        strip.text.x = element_text(size = 8, face = "plain", color = "black"),
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
        panel.grid.major = element_line(colour = "lightgray", size = 0.3),
        panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + facet_grid(~Reservoir, switch="x")

# Export ggplot to Latex
tikz(file = "Alpha_diversity_shannon_reservoirs.tex", width = 6, height = 3)
p
dev.off()

# Get min, max, median, and quartiles for each reservoir
ggplot_build(p)$data
# Get reservoirs
```

```
reservoirs = unique(all.pipelines.detailed$Reservoir)
for (reservoir in reservoirs) {
  print(reservoir)
  # Extract reservoir
  reservoir = all.pipelines.detailed[all.pipelines.detailed$Reservoir == reservoir,]
  # Get min, max, median, and quartiles for each reservoir
  print(round(min(reservoir$Shannon), digits = 3))
  print(round(max(reservoir$Shannon), digits = 3))
  print(median(reservoir$Shannon))
  print(quartiles(reservoir$Shannon))
  print(reservoir[reservoir$Shannon == min(reservoir$Shannon),]["Shannon"])
  print(reservoir[reservoir$Shannon == max(reservoir$Shannon),]["Shannon"])
  print("-----------------------------------------------------------")
  print("")
}

## Statistical analyses ##

# fit linear models
mod.Shannon = aov(Shannon~Reservoir, data=all.pipelines.detailed)
# ANOVA
anova.test = anova(mod.Shannon)
anova.test
# Tukey
tukey.test = TukeyHSD(mod.Shannon)
tukey.test

# Check if p-value < 0.05
tukey.test = as.data.frame(tukey.test[["Reservoir"]])
tukey.test[tukey.test$`p adj` < 0.05,]

# Export as latex tables
print(xtable(anova.test, digits = c(0, 0, 4, 4, 4, 4)), booktabs=TRUE, file = "Alpha_
    ↪ Diversity_Shannon_ANOVA_Reservoirs.tex")

## Seasons ##

# Plot Season
p = ggplot(all.pipelines.detailed, aes(y = Shannon)) +
  stat_boxplot(geom = "errorbar", lwd = 1, position = "dodge", show.legend = F, aes(
      ↪ color = Season)) +
  geom_boxplot(coef = 1.5, show.legend = F, alpha = 0.5, outlier.alpha = 1, varwidth
      ↪ = T, lwd = 1, aes(color = Season, fill = Season)) +
  xlab(NULL) +
  ylab("Shannon index") +
  theme(axis.text.x = element_blank(),
```

```
262         axis.text.y = element_text(size = 6, color = "black"),
263         axis.title = element_text(size = 10, face = "plain"),
264         axis.ticks.x = element_blank(),
265         legend.text = element_text(size = 6, color = "black"),
266         legend.title = element_text(size = 8, face = "plain"),
267         legend.margin = margin(t = 0, unit='cm'),
268         legend.key = element_rect(fill = NA, color = NA),
269         strip.text.x = element_text(size = 8, face = "plain", color = "black"),
270         strip.background = element_blank(),
271         #plot.margin = margin(10, 10, 10, 50),
272         plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
273                                   margin = margin(10, 0, 10, 0)),
274         panel.background = element_rect(fill = 'white'),
275         panel.grid.major = element_line(colour = "lightgray", size = 0.3),
276         panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
277         panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
278   ) + facet_grid(~Season, switch="x")
279 
280 # Export ggplot to Latex
281 tikz(file = "Alpha_diversity_shannon_seasons.tex", width = 6, height = 3)
282 p
283 dev.off()
284 
285 # Get min, max, median, and quartiles for each season
286 ggplot_build(p)$data
287 # Get seasons
288 seasons = unique(all.pipelines.detailed$Season)
289 for (season in seasons) {
290   print(season)
291   # Extract season
292   season = all.pipelines.detailed[all.pipelines.detailed$Season == season,]
293   # Get min, max, median, and quartiles for each season
294   print(round(min(season$Shannon), digits = 3))
295   print(round(max(season$Shannon), digits = 3))
296   print(median(season$Shannon))
297   print(quartiles(season$Shannon))
298   print(season[season$Shannon == min(season$Shannon),]["Shannon"])
299   print(season[season$Shannon == max(season$Shannon),]["Shannon"])
300   print("------------------------------------------------------------")
301   print("")
302 }
303 
304 # Fences
305 quartiles = quartiles(all.pipelines.detailed[all.pipelines.detailed$Season == "Summer
    ↪ ",]$Shannon)
306 upperq = round(quartiles$value[["75%"]])
```

```
lowerq = round(quartiles$value[["25%"]])
iqr = upperq - lowerq
upper.fence = upperq + (1.5 * iqr)
lower.fence = lowerq - (1.5 * iqr)

## Statistical analyses ##

# fit linear models
mod.Shannon = aov(Shannon~Season, data=all.pipelines.detailed)
# ANOVA
anova.test = anova(mod.Shannon)
anova.test
# T-test
t.test(Shannon~Season, data=all.pipelines.detailed, var.equal = TRUE)
# Tukey
tukey.test = TukeyHSD(mod.Shannon)
tukey.test

# Check if p-value < 0.05
tukey.test = as.data.frame(tukey.test[["Season"]])
tukey.test[tukey.test$`p adj` < 0.05,]

# Export as latex tables
print(xtable(anova.test, digits = c(0, 0, 4, 4, 4, 4)), booktabs=TRUE, file = "Alpha_
    ↪ Diversity_Shannon_ANOVA_Seasons.tex")
```

Source Code A.21: Calculate alpha diversity shannon index and create charts for pipelines, reservoirs, and season.

```
library(dplyr)
library(tidyr)
library(vegan)
library(xtable)
library(ggforce)
library(ranacapa)
library(tikzDevice)
library(concaveman)

# Set working directory to source file location
if(Sys.getenv("RSTUDIO") == "1"){
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
}else{
  setwd(utils::getSrcDirectory()[1])
}

# Load data
```

```r
load("Tables.RData")

##### Statistical analyses using all.pipelines.detailed #####

### Pipeline, Reservoir, and Season

# Remove columns Reservoir, Season and Pipeline
statistic = all.pipelines.detailed[,! colnames(all.pipelines.detailed) %in% c("Reservoir", "Season", "Pipeline")]

### BETA

# Compute dissimilarity indices
beta.jaccard = vegdist(statistic, method = "jaccard")

# Minimum and Maximum values
min(beta.jaccard)
which(as.matrix(beta.jaccard) == min(beta.jaccard), arr.ind=TRUE)
max(beta.jaccard)
which(as.matrix(beta.jaccard) == max(beta.jaccard), arr.ind=TRUE)

# Calculate PCoA principal coordinates analysis
pc.jaccard <- as.data.frame(cmdscale(beta.jaccard, k = 2))

# Create new columns
pc.jaccard$Reservoir_Season_Pipeline = rownames(pc.jaccard)
pc.jaccard = pc.jaccard %>% separate(Reservoir_Season_Pipeline, c("Reservoir", "Season", "Pipeline"), " ", remove = F)

# Multivariate homogeneity of groups dispersions (variances)
beta.disp = betadisper(beta.jaccard, pc.jaccard$Pipeline)

# Tukey Honest Significant Differences
tukey = TukeyHSD(beta.disp)
# Export pair Tukey table
print(xtable(tukey$group, digits = c(1, 3, 3, 3, 3)), booktabs=TRUE, file = "Beta_Diversity_Jaccard_Pipelines_Tukey.tex")

# Pipeline
# Permutational Multivariate Analysis of Variance Using Distance Matrices
permanova = adonis(as.formula("beta.jaccard~Pipeline"), data = pc.jaccard)
# Export permanova table
print(xtable(permanova$aov.tab), booktabs=TRUE, file = "Beta_Diversity_Jaccard_Pipelines_Permanova.tex")

# Pairwise multilevel comparison using adonis
```

```
pair.permanova = pairwise_adonis(statistic, pc.jaccard$Pipeline, sim_method = "jaccard")
# Export pair permanova table
print(xtable(pair.permanova, digits = c(0, 1, 2, 2, 3, 2, 1)), booktabs=TRUE, file = "Beta_Diversity_Jaccard_Pipelines_Pair_Permanova.tex")

# Reservoir
# Permutational Multivariate Analysis of Variance Using Distance Matrices
permanova = adonis(as.formula("beta.jaccard~Reservoir"), data = pc.jaccard)

# Pairwise multilevel comparison using adonis
pair.permanova = pairwise_adonis(statistic, pc.jaccard$Reservoir, sim_method = "jaccard")

# Season
# Permutational Multivariate Analysis of Variance Using Distance Matrices
permanova = adonis(as.formula("beta.jaccard~Season"), data = pc.jaccard)

# Pairwise multilevel comparison using adonis
pair.permanova = pairwise_adonis(statistic, pc.jaccard$Season, sim_method = "jaccard" )

# Calculate eigenvalue percentage
eigenvalues = summary(eigenvals(beta.disp, model = "all"))
eigenvalues.percentage = eigenvalues["Proportion Explained", ]
eigenvalues.percentage = round(100 * eigenvalues.percentage, 2)

pc.jaccard = unite(pc.jaccard, "Reservoir Season", Reservoir, Season, sep = " ", remove = F)

# Export plot (Remove % of the plot and put it after export table)
tikz(file = "Beta_Diversity_Jaccard_PCoA_PRS.tex", width = 6, height = 3.7)
# Plot PCoA
ggplot(pc.jaccard, aes(x = V1, y = V2)) +
  geom_point(aes(shape = Pipeline, fill = `Reservoir Season`, color = `Reservoir Season`), size = 1, stroke = 1) +
  # stat_ellipse(aes(fill = Pipeline, color = Pipeline), geom = "polygon", alpha = 0.3, size = 0.5) +
  #geom_mark_hull(aes(fill = Reservoir_Season), alpha = 0.3, expand = unit(3, "mm")) +
  xlab(paste("PCoA1[", eigenvalues.percentage[["PCoA1"]], "]", sep = "")) +
  ylab(paste("PCoA2[", eigenvalues.percentage[["PCoA2"]], "]", sep = "")) +
  theme(axis.text.x = element_text(angle = 45, size = 6, color = "black",
                                   face = "plain", vjust = 1, hjust = 1),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
```

```
        legend.text = element_text(size = 6, color = "black"),
        legend.title = element_text(size = 8, face = "plain"),
        legend.margin = margin(t = 0, unit='cm'),
        legend.key = element_rect(fill = NA, color = NA),
        strip.text.x = element_text(size = 6, face = "plain", color = "black"),
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
        panel.grid.major = element_line(colour = "lightgray", size = 0.3),
        panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + scale_shape_manual(values = c("Anacapa" = 21,
                                    "Barque" = 22,
                                    "MetaBEAT" = 23,
                                    "MiFish" = 24,
                                    "SEQme" = 25)) +
  scale_fill_manual(values = c("Klíčava Autumn" = "white",
                               "Klíčava Summer" = "#B79F00",
                               "Římov Autumn" = "white",
                               "Římov Summer" = "#00BFC4",
                               "Žlutice Autumn" = "white",
                               "Žlutice Summer" = "#F564E3")) +
  scale_color_manual(values = c("Klíčava Autumn" = "#B79F00",
                                "Klíčava Summer" = "black",
                                "Římov Autumn" = "#00BFC4",
                                "Římov Summer" = "black",
                                "Žlutice Autumn" = "#F564E3",
                                "Žlutice Summer" = "black")) +
  guides(fill=guide_legend(override.aes=list(shape=21)))
dev.off()

### Pipeline

# Remove columns Reservoir, Season
statistic = all.pipelines.detailed[,! colnames(all.pipelines.detailed) %in% c("
    ↪ Reservoir", "Season")]
statistic = aggregate(. ~ Pipeline, data = statistic, FUN = sum)
rownames(statistic) = statistic$Pipeline
statistic$Pipeline = NULL

# Computes dissimilarity indices for pipelines
beta.jaccard = vegdist(statistic, method = "jaccard")
min(beta.jaccard)
max(beta.jaccard)
```

```
beta.jaccard

# Export jaccard table
print(xtable(as.matrix(beta.jaccard)), booktabs=TRUE, file = "Beta_Diversity_Jaccard_
    ↪ Pipelines.tex")

### Reservoir

# Remove columns Pipeline, Season
statistic = all.pipelines.detailed[,! colnames(all.pipelines.detailed) %in% c("
    ↪ Pipeline", "Season")]
statistic = aggregate(. ~ Reservoir, data = statistic, FUN = sum)
rownames(statistic) = statistic$Reservoir
statistic$Reservoir = NULL

# Computes dissimilarity indices for Reservoirs
beta.jaccard = vegdist(statistic, method = "jaccard")
min(beta.jaccard)
max(beta.jaccard)
beta.jaccard

# Export jaccard table
print(xtable(as.matrix(beta.jaccard)), booktabs=TRUE, file = "Beta_Diversity_Jaccard_
    ↪ Reservoirs.tex")

### Season

# Remove columns Reservoir, Pipeline
statistic = all.pipelines.detailed[,! colnames(all.pipelines.detailed) %in% c("
    ↪ Reservoir", "Pipeline")]
statistic = aggregate(. ~ Season, data = statistic, FUN = sum)
rownames(statistic) = statistic$Season
statistic$Season = NULL

# Computes dissimilarity indices for Seasons
beta.jaccard = vegdist(statistic, method = "jaccard")
beta.jaccard
```

Source Code A.22: Calculate beta diversity Jaccard dissimilarity indices and create a chart for pipelines, reservoirs, and season.

```
library(tidyr)
library(ggplot2)
library(rstatix)
library(EnvStats)
```

```r
library(normtest)
library(tikzDevice)
library(RVAideMemoire)

# Set working directory to source file location
if(Sys.getenv("RSTUDIO") == "1"){
  setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
}else{
  setwd(utils::getSrcDirectory()[1])
}

# Load data
load("Tables.RData")

# Create column for Reservoir and Season together
all.pipelines.only.positive.control = unite(all.pipelines.only.positive.control,
    ↪ Reservoir_Season, c(Reservoir, Season), remove=F, sep=" ")

# Export ggplot to Latex
tikz(file = "Positive_Control_PRS.tex", width = 6, height = 3)
# Plot Reservoir_Season_Pipeline
ggplot(all.pipelines.only.positive.control, aes(x = Pipeline, y = Percentage_of_the_
    ↪ total)) +
  geom_point(aes(fill = Pipeline), shape = 21, size = 3) +
  xlab(NULL) +
  ylab("Percentage of assigned reads") +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.y = element_text(size = 6, color = "black"),
        axis.title = element_text(size = 10, face = "plain"),
        legend.text = element_text(size = 6, color = "black"),
        legend.title = element_text(size = 8, face = "plain"),
        legend.margin = margin(t = 0, unit='cm'),
        legend.key = element_rect(fill = NA, color = NA),
        strip.text.x = element_text(size = 6, face = "plain", color = "black"),
        strip.background = element_blank(),
        #plot.margin = margin(10, 10, 10, 50),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold",
                                  margin = margin(10, 0, 10, 0)),
        panel.background = element_rect(fill = 'white'),
        panel.grid.major = element_line(colour = "lightgray", size = 0.3),
        panel.grid.minor = element_line(colour = "lightgray", size = 0.1),
        panel.border = element_rect(colour = "black", fill = NA, size = 0.5)
  ) + facet_grid(~Reservoir_Season, switch = "x") +
  scale_fill_discrete(name = "Pipelines")
dev.off()
```

```r

# Minimum percentage
min(all.pipelines.only.positive.control$Percentage_of_the_total)
all.pipelines.only.positive.control[all.pipelines.only.positive.control$Percentage_of
    ↪ _the_total == min(all.pipelines.only.positive.control$Percentage_of_the_total)
    ↪ ,]

# Maximum percentage
max(all.pipelines.only.positive.control$Percentage_of_the_total)
all.pipelines.only.positive.control[all.pipelines.only.positive.control$Percentage_of
    ↪ _the_total == max(all.pipelines.only.positive.control$Percentage_of_the_total)
    ↪ ,]

# Mean and median of the percentage list
median(all.pipelines.only.positive.control$Percentage_of_the_total)
mean(all.pipelines.only.positive.control$Percentage_of_the_total)

# Check normality
byf.shapiro(Percentage_of_the_total~Pipeline, data =
    ↪ all.pipelines.only.positive.control)

# fit linear models
mod.percentage = aov(Percentage_of_the_total~Pipeline, data=
    ↪ all.pipelines.only.positive.control)
# ANOVA
anova.test = anova(mod.percentage)
anova.test
# Tukey
tukey.test = TukeyHSD(mod.percentage)
tukey.test
# Check if p-value < 0.05
tukey.test = as.data.frame(tukey.test[["Pipeline"]])
tukey.test[tukey.test$`p adj` < 0.05,]

# Extract pipelines
pipelines = unique(all.pipelines.only.positive.control$Pipeline)

# For each pipeline do t.test, wilcox, median, median, min and max
for (pipeline in pipelines) {
  dat = all.pipelines.only.positive.control[all.pipelines.only.positive.control$
      ↪ Pipeline == pipeline,]

  # Print pipeline name and data
  print(pipeline)
  print(dat$Percentage_of_the_total)

```

```
  # Statistical tests
  print(paste("chi square: ", round(varTest(dat$Percentage_of_the_total, alternative
      ↪ = "greater", sigma.squared = 5)$p.value, digits = 3)))
  print(paste("t.test: ", t.test(dat$Percentage_of_the_total)$p.value))
  print(paste("wilcox.test: ", wilcox.test(dat$Percentage_of_the_total)$p.value))

  # Mean and median of the percentage list
  print(paste("Median: ", median(dat$Percentage_of_the_total)))
  print(paste("Mean: ", mean(dat$Percentage_of_the_total)))

  # Min and max of the percentage list
  print(paste("Min: ", min(dat$Percentage_of_the_total)))
  print(paste("Max: ", max(dat$Percentage_of_the_total)))

  print("")
}
```

Source Code A.23: Calculate positive control Maylandia zebra detection and create a chart showing the difference between pipelines.