UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA

DIPLOMOVÁ PRÁCE

Aplikace pro preprocessing, imputaci a analýzu proteomických dat

Katedra matematické analýzy a aplikací matematiky
Vedoucí práce: **doc. RNDr. Eva Fišerová, Ph.D.**
Vypracoval(a): **Bc. Tomáš Chupáň**
Studijní program: N0541A170026 Aplikovaná matematika
Studijní obor: Aplikovaná matematika
Forma studia: prezenční
Rok odevzdání: 2024

BIBLIOGRAFICKÁ IDENTIFIKACE

Autor: Bc. Tomáš Chupáň

Název práce: Aplikace pro preprocessing, imputaci a analýzu proteomických dat

Typ práce: Diplomová práce

Pracoviště: Katedra matematické analýzy a aplikací matematiky

Vedoucí práce: doc. RNDr. Eva Fišerová, Ph.D.

Rok obhajoby práce: 2024

Abstrakt: Cílem práce bylo vytvořit Shiny aplikaci v softwaru R, která poslouží jako interaktivní nástroj pro preprocessing, imputaci a analýzu proteomických dat. Nová aplikace *proteoME* přesně takovým nástrojem je. Po importu datové sady nabízí základní vizualizaci dat, jejich transformaci a normalizaci. Následujícím krokem je agregace na úroveň vzorků, při níž má uživatel k dispozici kromě několika možností postupu také vizualizace usnadňující volbu parametrů. Agregovaná data je také možné filtrovat dle procenta chybějících hodnot na několika úrovních, příp. chybějící hodnoty imputovat jednou z dostupných metod imputace. Samotná analýza takto připravených proteomických dat poskytuje několik možností testování shody distribuce abundancí proteinů mezi jednotlivými skupinami léčby – lze porovnávat libovolný počet takových skupin. Výsledky analýzy ve formě detailní tabulky s širokou paletou možností exportu, řazení řádků i filtrace hodnot se snaží maximálně vyhovět potřebám uživatele, podobně jako hlavní grafický výstup analýzy – *volcano plot*, který si uživatel taktéž může přizpůsobit dle svých požadavků. V textu je kromě návodu a ukázkového zpracování syntetické datové sady v aplikaci *proteoME* rozebráno také teoretické pozadí nezbytné pro schopnost pracovat s proteomickými daty, praktickým bonusem je pak kapitola nabízející návod, jak vyvinout Shiny aplikaci ve formě R balíčku – tímto přístupem byla vytvořena také aplikace *proteoME*.

Klíčová slova: proteomika, software R, Shiny aplikace, golem

Počet stran: 133

Počet příloh: 0

Jazyk: český

BIBLIOGRAPHICAL IDENTIFICATION

Author: Bc. Tomáš Chupáň

Title: Application for proteomics data preprocessing, imputation and analysis

Type of thesis: Master's

Department: Department of Mathematical Analysis and Application of Mathematics

Supervisor: doc. RNDr. Eva Fišerová, Ph.D.

The year of presentation: 2024

Abstract: The aim of this work was to create a Shiny application in R software that serves as an interactive tool for preprocessing, imputation and analysis of proteomics data. The new application *proteoME* is just such a tool. After importing the dataset, it offers basic data visualization, transformation and normalization. The next step is aggregation to the sample level, where the user is provided with several procedure options in addition to visualizations to facilitate parameter selection. Aggregated data can also be filtered by percentage of missing values at several levels, or missing values can be imputed using one of the available imputation methods. The analysis of proteomics data prepared in this way provides several possibilities for testing the distribution of protein abundances between treatment groups – any number of such groups can be compared. The results of the analysis, in the form of a detailed table with a wide variety of export options, row ordering and filtering of values, try to meet the user's needs as much as possible, as does the main graphical output of the analysis – *volcano plot*, which can also be customized by the user. In addition to a tutorial and demonstration of synthetic dataset processing in *proteoME*, the text also discusses the theoretical background necessary for the ability to work with proteomics data, with the practical bonus of a chapter offering instructions on how to develop a Shiny application in the form of an R package – the *proteoME* application was also developed using this approach.

Key words: proteomics, software R, Shiny application, golem

Number of pages: 133

Number of appendices: 0

Language: Czech

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně pod vedením paní doc. RNDr. Evy Fišerové, Ph.D. a všechny použité zdroje jsem uvedl v seznamu literatury.

V Olomouci dne ....

podpis

Obsah

Seznam obrázků

Seznam tabulek

Poděkování

Rád bych poděkoval vedoucí práce paní doc. RNDr. Evě Fišerové, Ph.D., která mi po celou dobu trpělivě pomáhala svými cennými radami a motivovala mě k tomu, abych z práce vytěžil maximum možného. Také bych chtěl poděkovat své rodině za vytvoření perfektního zázemí a neúnavnou podporu.

Úvod

V dnešní době nám moderní technologie umožňují sbírat širokou škálu datových sad z různých oborů. Jedním z těchto oborů je proteomika, která v tomto tisíciletí zažívá působivý „boom“ a prací s tímto typem dat se zabývá mnoho výzkumníků – avšak ne tolik, aby existovalo odpovídající množství nástrojů, které by jim každodenní úkoly usnadnily. Jde navíc o velmi specifickou, komplexní problematiku, u které se mnohdy ani lidé pracující přímo v tomto oboru neshodnou, jak při zpracování a analýzy tohoto typu dat postupovat. Dalším úskalím je křivka učení, která při začátcích na tomto poli roste jen velmi pomalu (např. u datových vědců, kteří slovo *proteomika* slyší poprvé v životě) a chvíli trvá, než se člověk zorientuje v tom, jaký je princip práce s těmito daty.

Tento text se snaží usnadnit začátek působení v tomto oboru prostřednictvím shrnutí základních postupů a úkonů potřebných při zpracování proteomických dat (a uvádí samotný pojem *proteomika* v dostupné formě). Tomu je věnována zejména první kapitola, která nabízí i konkrétní příklady používaných metod, jakož i možnosti grafických výstupů.

Jakýkoliv nástroj pro ulehčení práce s (nejen) proteomickými daty je jistě vítán – jednou z možností je Shiny aplikace vytvořená v prostředí R, které díky otevřenosti a umožnění přispět komukoliv svým balíčkem disponuje mj. také různými metodami využitelnými v proteomice. Aplikace, která

by zvládla efektivně pomoct vědcům z tohoto oboru, však téměř vždy bude obsáhlé dílo co do investovaného času i řádků kódu – proto není ideální variantou, aby se případný zájemce o podílení se na vývoji takového nástroje rovnou pustil do práce standardním způsobem bez patřičné přípravy a vhodné metodiky. Ve druhé kapitole je proto představen princip tvorby Shiny aplikací ve formě R balíčku. Jde o přístup jednak zajišťující větší přehlednost a kontrolu nad celým projektem, jednak také usnadňující samotnou práci při programování a následnou publikaci a údržbu aplikace.

Závěrečná kapitola je pak nejen logickým spojením prvních dvou částí, ale také realizací hlavního cíle této diplomové práce – tím je vytvořit nástroj pro preprocessing, imputaci a analýzu proteomických dat. Je zde představena nová Shiny aplikace *proteoME*, která byla vytvořena ve formě R balíčku a publikována pro použití širokou veřejností. Tato kapitola kromě představení aplikace poskytuje i návod pro kohokoliv, kdo by ji chtěl využít pro analýzu vlastních dat – součástí je i vzorové zpracování syntetických dat sledující jednotlivé kroky.

Kapitola 1

Práce s proteomickými daty

Cílem této kapitoly je představit čtenáři pojem *proteomika*, způsob sběru proteomických dat a v neposlední řadě postup při zpracování tohoto typu dat, jehož pochopení a schopnost aplikovat jej na reálnou datovou sadu jsou pro celou práci stěžejní.

Jednotlivé podkapitoly na sebe navazují a mají ambici sloužit jako ucelený soubor znalostí potřebných k tomu, aby čtenář mohl vkročit do světa proteomických dat a pracovat s nimi. Rozhodně nepůjde o kompletní výčet všech známých metod a postupů – čtenář by však měl získat základní přehled o dané problematice a díky nabyté orientaci si případně další možnosti dohledat.

1.1 Proteomika aneb Co je to za data?

Celá kapitola se sice týká oboru, který leží na pomezí biologie a chemie, budeme jej však nahlížet zejména z pohledu statistika. I ten ovšem potřebuje rozumět alespoň základům oblasti, z které data pochází, aby mohl správně volit používané metody či interpretovat výsledky své analýzy. Proto se v této

podkapitole podíváme blíže na to, čím se vlastně proteomika zabývá a jak se taková data sbírají.

1.1.1 Proteomika jako vědní obor

Začněme pojmem *proteom*, jehož podobnost se známějším pojmem *genom*[1] není náhodná. Wilkins ve svém článku [26] uvádí proteom coby soubor všech PROTEinů vytvořených v organismu na základě genOMu – bílkoviny[2] jsou totiž přímým produktem genetické informace. Pro (nejen) naše tělo jsou nesmírně důležité. Významně se podílí na většině buněčných funkcí, ať už jde o převod a zpracování biologické informace či látkovou a energetickou výměnu [16].

Proteomika je pak obor zabývající se systematickým studiem proteomu, v rámci kterého se snaží prohloubit poznání o funkcích, struktuře či množství proteinů v daném organismu, typu buňky či tkáně. Tato nauka si také klade za cíl popsat a vysvětlit roli proteinů v biologických procesech, to, jak interagují s jinými látkami, případně i změnu složení proteomu při změnách v organismu – typicky rozdíl v souboru bílkovin zdravého a nemocného jedince [4][16].

Uplatnění nachází proteomika zejména v medicíně – právě kvůli zkoumání příčin a mechanismů, které stojí za různými chorobami či patologickými stavy. Proniká však také do dalších oborů od farmacie (vývoj léčiv) přes ochranu životního prostředí až po zemědělství [4].

Již byla zmíněna úzká provázanost proteinů s genomem. Oproti genomice je však studium bílkovin daleko těžší, neboť složení proteinů v organismu je velmi nestálé a neustále se měnící. Na druhou stranu, pochopení těchto mechanismů by naše poznání (např. právě v oblasti medicíny) posunulo na

[1] Soubor všech genů daného organismu.
[2] Pouze český ekvivalent slova *protein*.

zcela novou úroveň – proto má smysl snažit se přijít na to, jak co nejlépe získat potřebná data a „vytěžit“ z nich co nejvíc [16].

1.1.2 Jak se sbírají proteomická data

Možností, jak tato data sbírat, je celá řada. Vhodnou metodologii volíme dle stanoveného cíle výzkumu [4]. V této práci se zaměříme pouze na jeden přístup, a to tzv. shotgun proteomiku.

V tomto přístupu máme na „vstupu“ neseparovanou směs proteinů. Příprava probíhá tak, aby v každém vzorku bylo stejné množství (celkový objem) proteinů. Z odebrané tkáně/tělní tekutiny[3] jednoho jedince se může připravit více vstupů – jde o tzv. technické replikáty (běžně 2–3). Platí však, že vzorek jedince je vždy odebrán v jeden čas a na jednotlivé replikáty rozdělen až v laboratoři – vzpomeňme na neustále se měnící složení proteinů v organismu. Takto připravené vzorky se pomocí speciální metody rozštěpí na peptidy (kratší řetězce aminokyselin, z kterých se proteiny skládají). Ty jsou separovány metodou HPLC (*high-performance liquid chromatography*) a následně sekvenovány tzv. tandemovou hmotnostní spektrometrií (často značenou MS/MS). Přístroj, který nám konečně z těchto chemicko-biologických vstupů poskytne číselné výstupy, je tedy hmotnostní spektrometr [4].

Zkusme se na tento proces podívat zjednodušeně. Hmotnostní spektrometr dokáže jednotlivé molekuly peptidů „roztřídit“ dle jejich hmotnosti, přičemž tyto molekuly projdou celým procesem ve dvou kolech (odtud tandemová hmotnostní spektrometrie). Po počátečním seřazení dle hmotnosti peptidů (resp. poměru hmotnosti k náboji, neboť molekuly jsou ionizovány) nastane kolize se speciálním plynem, který molekuly dál rozdělí na menší fragmenty. V druhém kole se tedy změří hmotnost těchto fragmentů. Přístroj

[3] Často se jedná o krev, sliny, ale také např. o dechový kondenzát.

je pak schopen detekovat intenzitu fragmentů, které přístrojem doputují k detektoru – výsledkem jsou tzv. spektra udávající to, kolik tam těchto zlomků peptidů je [26]. O zpracování jednoho replikátu hmotnostním spektrometrem hovoříme jako o „běhu“ (z anglického *run*, někdy se používá přímo anglický termín).

Na řadu přichází software[4], který výsledky MS/MS zpracuje a jako výstup vrátí datovou sadu použitelnou k další analýze. Ačkoliv máme k dispozici pouze informaci o krátkých sekvencích aminokyselin, dokážeme pomocí softwaru a rozsáhlé databáze proteinů spárovat detekované fragmenty s peptidy, z kterých (pravděpodobně) pochází. Jejich množství, resp. intenzita v jednotlivých replikátech je zde vyjádřena tzv. abundancemi (příp. se pracuje přímo s pojmem intenzita – záleží na použitém softwaru). Podobně se dají peptidy spárovat s původními proteiny (opět s nějakou mírou nejistoty), čímž se dostáváme zpět na proteinovou úroveň [26]. V následujících kapitolách se budeme bavit pouze o proteinové úrovni, avšak analýzy se dají provádět i na té peptidové. Softwarový výstup jednoho měření (běhu) se někdy označuje jako „soubor“ (z anglického *file*, opět se používá i přímo tento anglický termín) – s touto terminologií budeme pracovat i v tomto textu.

Výsledné datové sady jsou velice citlivé na použitý typ hmotnostního spektrometru a jeho nastavení stejně jako na parametry softwaru, který výsledná spektra zpracovává. Proto jsou výsledky získané jinými přístroji a při jiných hodnotách parametrů jen obtížně porovnatelné.

Je možné, že čtenář ani teď nemá představu, s čím že to má pracovat. V Tabulce 1.1 je část uměle vytvořené ukázkové datové sady ve formátu, v jakém ji vrátí program Proteome Discoverer při jednom z možných nastavení. V prvním sloupci *Accession* jsou unikátní identifikátory proteinů,

[4] Často se používá komerční Proteome Discoverer [31] či volně dostupný MaxQuant [23].

kterým se říká *accession number*. Tyto kódy jsou dohledatelné v databázi proteinů UniProt[5], kde jsou k dispozici detailní informace o proteinu.

Tabulka 1.1: Ukázka datové sady

Accession	F1	F2	F3
P04745	441 466,04	443 635,72	*NA*
B7ZMD7	*NA*	*NA*	12 566,05
H6VRF8	13 855 145,99	20 437 597,60	*NA*
P19961	43 636,42	26 260,37	3 714,54
P35527	290 059,17	459 357,33	1 514,41

V dalších sloupcích jsou abundance těchto proteinů v jednotlivých souborech (s ID F1, F2 ...). Jak bylo zmíněno výše, jeden subjekt může mít více replikátů svého vzorku, tedy k němu může náležet několik souborů, z nichž každý může mít zcela jiné výsledky (např. určitý protein je v jednom technickém replikátu subjektu detekován a ve zbylých dvou nikoliv). Všimněte si i chybějících hodnot – ty jsou v tomto typu dat velmi častým jevem (mnohem častějším, než bývá zvykem). Jejich původu a možnostem imputace se věnuje podkapitola 1.2.6. Užitečné je také uvést, jaké rozměry očekávat u plné datové sady. Sloupců s abundancemi (souborů) bývají vyšší desítky až nižší stovky. Řádků s proteiny však jsou zpravidla nižší jednotky tisíc.

1.1.3 Proč se sbírají proteomická data

Celý postup představený v předchozí části zní velmi komplikovaně (a je ještě komplikovanější, než bylo popsáno). Jaký je tedy vlastně účel sběru tohoto typu dat? Co chceme zjistit?

Zpravidla sledujeme nějaký biologický jev (např. rakovinu či jiné onemocnění), s kterým je (jako ostatně s každou změnou v těle organismu) spojena

[5]Dostupná zde: https://www.uniprot.org/

proměna proteomu. Analýzou tohoto typu dat pak chceme nalézt kandidátní proteiny typické pro tento jev [16]. Jinými slovy, zajímá nás, zda se ve skupině s projevem daného biologického jevu vyskytuje významně větší (či naopak menší) množství proteinu či skupiny proteinů než ve skupině, kde se tento jev neprojevuje[6]. V literatuře se tito kandidáti někdy označují jako *DEP* (z anglického *differentially expressed proteins*) [27]. Stejně tak můžeme mít vedle jedné „kontrolní“ skupiny (bez zkoumaného jevu) klidně i více skupin, kde se jev projevuje. To v případě, že pozorujeme více stádií nemoci (např. klinická stádia nádorů).

Vzhledem ke složitosti procesů nemůžeme na závěry analýzy těchto dat pohlížet jako na ultimátní odpověď na otázky typu „Jaká skupina proteinů je spojena s projevem této nemoci?“. Získané informace nám však slouží jako cenný zdroj hypotéz, které mohou vědci využít k plánování konkrétněji zacílených výzkumů.

Pokud bychom měli použít nějaké přirovnání: nejde nám o nalezení jehly v obrovské kupě sena, ale spíše o vytipování menší hromádky, v které by ta jehla možná mohla být. Zájem o rozvoj proteomických metod naznačuje, že tento přístup (ač na první pohled neoslní jednoznačnými výsledky dávajícími konkrétní odpověď) má svoje místo na poli moderní vědy a jeho další zdokonalování může lidstvo přiblížit k řešení zásadních problémů (nejen) v medicíně.

1.2 Postup při zpracování proteomických dat

V této části si postupně rozebereme jednotlivé fáze zpracování proteomických dat. Díky podkapitole 1.1.2 již víme, v jakém formátu je základní datová

[6]Tento přístup s porovnáváním skupin je jen jednou z možností, v tomto textu však bude jediná, kterou podrobně rozebereme.

sada s abundancemi proteinů. Ta nám však sama o sobě nestačí. Abychom dokázali porovnat proteom u (zjednodušeně řečeno) zdravých a nemocných jedinců, potřebujeme další dvě pomocné datové sady:

- **Anotace souborů** (též *file annotations*) – tabulka, ve které jsou jednotlivé soubory přiřazeny ke vzorku, z kterého pochází, příp. doplněny o označení replikátu. Dalšími využitelnými informacemi (pokud jsou dostupné) jsou datum a čas měření, které souvisí s rozdělením replikátů na tzv. „dávky“ (z anglického *batch*). Replikáty se totiž do hmotnostního spektrometru vkládají postupně po těchto dávkách (klidně i s odstupem několika dnů či týdnů mezi jednotlivými dávkami). Informace o dávce (číslo dávky, příp. i pořadí replikátu v rámci dávky) jsou nezbytné při vyšetřování tzv. *batch efektu*, tedy efektu dávky. Tomu se však v tomto textu věnovat nebudeme.

- **Anotace vzorků** (též *sample annotations*) – tabulka s informacemi o jednotlivých vzorcích (samplech) představujících naše subjekty. Nezbytnou informací je v případě porovnávání skupin tzv. *treatment group* (to, k jaké skupině subjekt patří – kontrola vs. nemoc, příp. její stádium). Dále však může obsahovat libovolné další údaje, např. pohlaví, věk ...

Ukázkový soubor s anotacemi souborů, resp. vzorků najdete v Tabulce 1.2, resp. v Tabulce 1.3.

1.2.1 Základní informace o datech

Po získání všech datových souborů je vhodné se na data podívat z několika pohledů, abychom si utvořili prvotní obraz a odpověděli si na možné otázky, příp. nás to, co uvidíme, upozorní na možné problémy v datech. Přístupným

Tabulka 1.2: Ukázka anotací souborů

fileID	sampleID	rep	batchNo	batchOrder	dateTime
F1	A1	1	1	1	2/10/2024 11:06
F2	A1	2	1	2	2/10/2024 18:38
F3	A2	1	1	3	2/11/2024 1:14
F4	A2	2	1	4	2/11/2024 6:27
F5	B1	1	2	1	2/28/2024 15:43
F6	B1	2	2	2	2/28/2024 23:33

Tabulka 1.3: Ukázka anotací vzorků

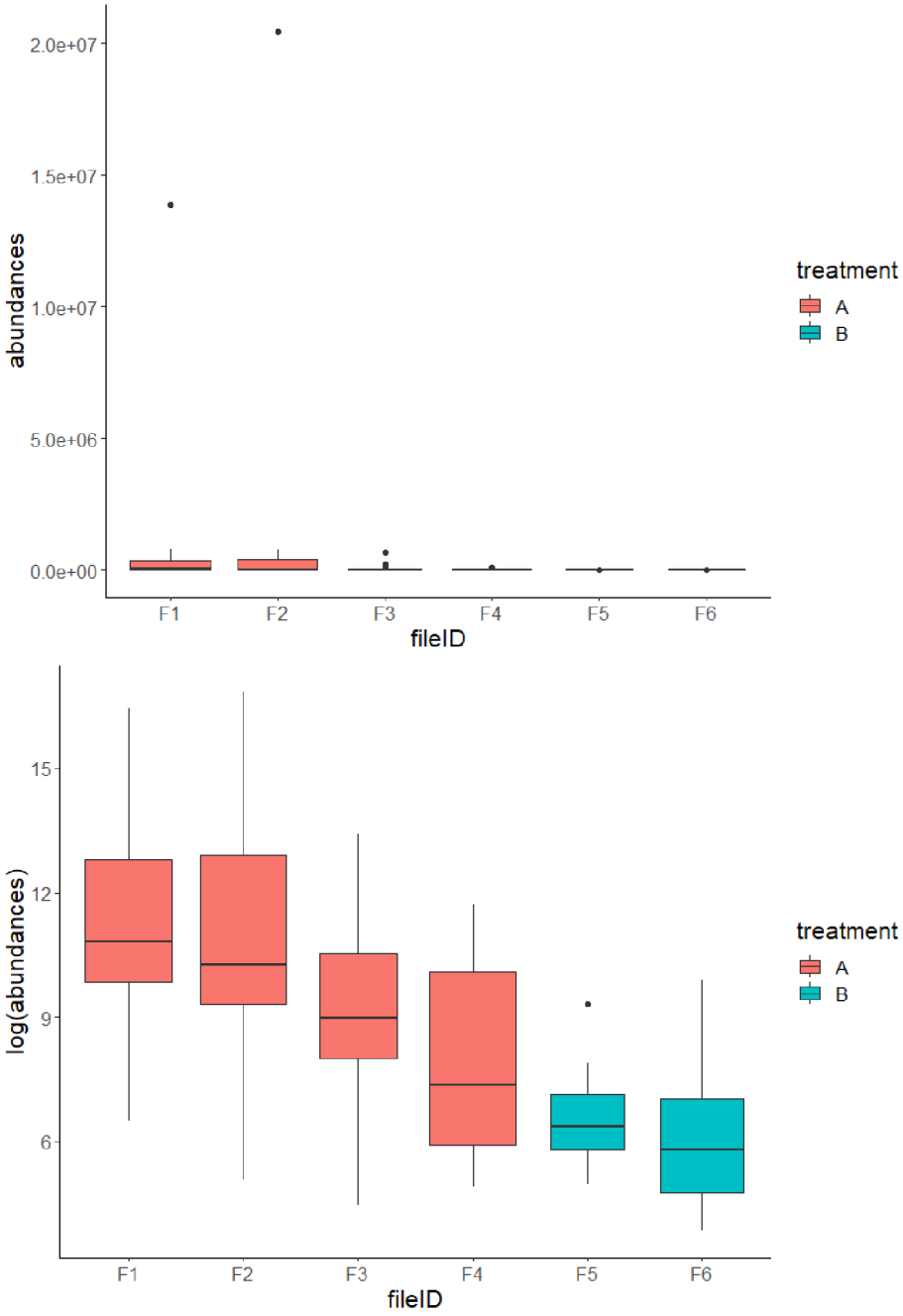
sampleID	treatment	sex	age	weight_kg
A1	A	M	56	95
A2	A	F	21	60
B1	B	F	33	65

nástrojem jsou vizualizace. Ukažme si tedy alespoň pár základních grafů[7], které můžeme za tímto účelem vytvořit.

1.2.1.1 Boxplot abundancí pro jednotlivé soubory

Tento typ grafu spočívá v tom, že u každého souboru vidíme, jakých hodnot abundance proteinů nabývají (včetně mediánu a horního a dolního kvartilu těchto hodnot). Je užitečné barevně odlišit boxploty dle skupiny (*treatment group*), stejně tak dostaneme dodatečnou informaci díky variabilní šířce boxplotů (odpovídající počtu detekovaných proteinů). Pokud máme možnost tvořit interaktivní grafy, můžeme tuto vizualizaci vylepšit infoboxem, který se objeví vždy při přesunutí kurzoru myši na konkrétní boxplot – v něm mo-

[7] V celé práci je u prvků grafů (popisky os, legenda ...) používána výhradně angličtina, neboť vizualizace odpovídají výstupům z aplikace *proteoME* (více v kapitole 3). Aplikace je totiž určena nejen českým uživatelům a anglický jazyk tak byl jedním z požadavků zadavatele.



Obrázek 1.1: Boxplot abundancí pro jednotlivé soubory – originální hodnoty (nahoře) a transformované hodnoty (dole)

hou být shrnuty jednak číselné charakteristiky (medián, horní a dolní kvartil, příp. minimum a maximum), jednak právě počet detekovaných proteinů u daného souboru[8].

První možnou odpověď na některou z otázek („Bude potřeba data transformovat?“) dostaneme při pohledu na Obrázek 1.1. Originální data (graf nahoře) jsou očividně velmi zešikmená a transformaci bude v pozdější fázi analýzy třeba provést (více toto téma rozebereme v části 1.2.2). Dole na Obrázku 1.1 můžete vidět výsledný graf, když na abundance použijeme logaritmickou transformaci se základem dva[9].

V případě, že máme soubory seřazené chronologicky dle data a času měření, můžeme pomocí tohoto grafu zaznamenat jeden z možných problémů vzniklých při měření. Pokud bychom totiž pozorovali jasný rostoucí trend abundancí, mohlo by to znamenat, že u vzorků měřených v pozdějších bězích přístroj detekuje i proteiny z běhů předchozích – to je možné typicky kvůli kontaminaci přístroje předchozím vzorkem.

Pro zkušené výzkumné pracovníky může být zajímavým poznatkem i to, pokud by jedna ze skupin (např. vzorky subjektů se zkoumanou nemocí) byla abundantnější než skupina druhá (např. kontrola). Zde se to však případ od případu liší a pro správné vyhodnocení je potřeba mít odborný vhled do medicínské stránky věci.

1.2.1.2 Histogram počtu detekovaných proteinů

Dalším grafem, který nám v úvodu analýzy může poskytnout základní informace o datové sadě, je histogram počtu detekovaných proteinů na úrovni

[8]Pro větší názornost lze doplnit i informaci o počtu chybějících hodnot v souboru a proporci detekovaných proteinů, tj. počet detekovaných proteinů v rámci souboru ku počtu všech detekovaných proteinů v celé datové sadě.

[9]Základ logaritmu roven dvěma je používán v celém textu. Pokud někde základ není specifikován, jde vždy o $\log_2(x)$.

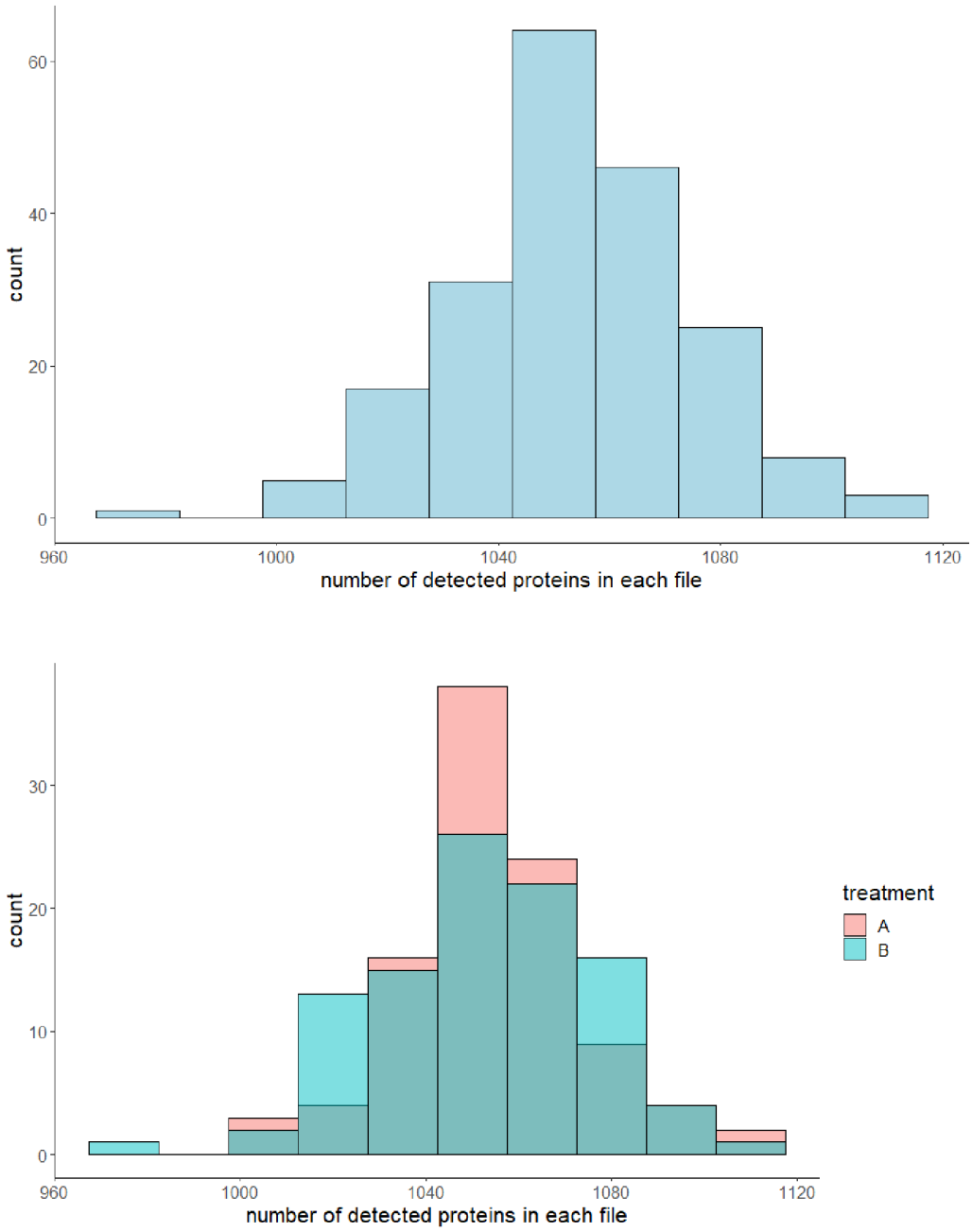
jednotlivých souborů. Pro získání těchto údajů stačí pro každý soubor zjistit, kolik nenulových hodnot je v daném sloupci přítomno – jako detekovaný protein tedy počítáme ten, který měl pro daný soubor kladnou abundanci [10]. Na ose y jsou pak četnosti (příp. proporce), jak jsme u histogramu zvyklí.

Máme dvě základní možnosti, jak histogram zobrazit. Tou první je ignorování rozdělení na skupiny dle treatmentu a vykreslení společného histogramu pro všechny soubory. V tomto případě nás může zajímat, kolem jakých hodnot se počet detekovaných proteinů obecně pohyboval, případně zda se v datech vyskytuje skupina souborů s neobvykle velkým či neobvykle malým počtem detekcí vzhledem k ostatním souborům v celé datové sadě. Takový histogram můžete vidět na Obrázku 1.2 nahoře.

Pokud však data rozdělíme dle sledovaných skupin, přidáme extra informaci o tom, zda se počet detekovaných proteinů může ve skupinách lišit, příp. zda jsou soubory s výrazně větším (či výrazně menším) počtem detekcí spojeny s konkrétní skupinou. Skupiny jsou odlišeny barevně, přičemž jednotlivé histogramy jsou částečně průhledné – vidíme tedy jejich „překrývání“. Doporučené je zvolit pro skupiny kontrastní barvy tak, abychom je i přes překrývání rozlišili. Příklad takové vizualizace je na Obrázku 1.2 dole.

Graf se dá samozřejmě různě obohatit. Nabízí se např. vertikální čára znázorňující medián počtu detekovaných proteinů (na celkové či skupinové úrovni). Se zapojením interaktivních prvků je možné při umístění kurzoru na konkrétní sloupec zobrazit infobox s rozsahem sloupce (v angličtině se používá termín *bin range*) a spočítanou četností (příp. proporcí).

[10] Analogicky jako nedetekované proteiny bereme ty, u nichž je ve sloupci daného souboru nulová hodnota či chybějící hodnota (*NA*). (Ne)detekovaný protein posuzujeme těmito kritérii jen v případě, že data nejsou transformovaná ani normalizovaná – např. po logaritmické transformaci už abundance samozřejmě nemusí být kladné, aby byl protein brán za detekovaný.



Obrázek 1.2: Histogram počtu detekovaných proteinů – bez rozdělení dle skupiny (nahoře) a s rozdělením dle skupiny (dole)

1.2.1.3 Číselné charakteristiky

Kromě grafů můžeme základní informace o datovém souboru získat i spočítáním některých číselných charakteristik – zde však (narozdíl od boxplotů) za cenu zjednodušeného pohledu na celou datovou sadu, nikoliv pro jednotlivé soubory. Výběr konkrétních charakteristik není nijak pevně dán a záleží spíše na osobních preferencích výzkumného pracovníka. Příklad tohoto typu výstupu můžete vidět v Tabulce 1.4.

Tabulka 1.4: Číselné charakteristiky datové sady – příklad

Minimum	Dolní kvartil	Medián	Průměr	Horní kvartil	Maximum	Variační rozpětí	Šikmost
13,38	536,01	2 699,62	273 802,7	16 038,95	52 140 660	52 140 647	12,87

Dodatečnou (a stále dobře zobrazitelnou – vizte Tabulku 1.5) informaci může přinést zohlednění *treatment group* – dokonce může pomoci podpořit optický dojem z boxplotů („Jedna skupina se mi zdá abundantnější než ta druhá.“), příp. poukázat na jeho chybnost.

Tabulka 1.5: Číselné charakteristiky datové sady dle skupiny – příklad

Skupina	Minimum	Dolní kvartil	Medián	Průměr	Horní kvartil	Maximum	Variační rozpětí	Šikmost
A	13,38	816,78	4 384,80	136 149,9	21 140,06	52 140 660	52 140 647	24,75
B	15,39	451,75	2 086,06	349 258,6	13 016,85	42 635 001	42 634 986	10,84

1.2.2 Transformace

Data, na základě nichž se počítaly číselné charakteristiky v Tabulkách 1.4 a 1.5, jsou sice uměle vytvořená, ale nesou s sebou jev, který je u proteomických dat velmi častý – jsou výrazně zešikmená směrem k velkým kladným hodnotám. Mnohé metody používané k analýze proteomických dat přitom předpokládají, že jsou data normálně rozdělená (či alespoň symetrická), příp.

jsou citlivé na extrémní hodnoty – originální podoba datové sady by tak použití těchto metod nedovolovala. Dostat data do vhodnější formy, se kterou se nám bude lépe pracovat, umíme ve většině případů poměrně snadno, a to jednoduchou transformací dat [15][19].

Nejpoužívanější metodou je logaritmická transformace[11], která si poradí s velkou kladnou šikmostí dat a učiní proteomická data více symetrická. Vyhodnotit efekt logaritmické transformace pomůže jak vhodná vizualizace (vizte Obrázek 1.1), tak srovnání číselných charakteristik. Ukázka takového srovnání je v Tabulce 1.6.

Tabulka 1.6: Ukázka vlivu logaritmické transformace na číselné charakteristiky

	Minimum	Dolní kvartil	Medián	Průměr	Horní kvartil	Maximum	Max−Min	Šikmost
Původní data	13,38	536,01	2 699,62	273 802,71	16 038,95	52 140 660,00	52 140 646,62	12,87
Logaritmovaná data	3,74	9,07	11,40	11,82	13,97	25,64	21,89	0,73

Problémem u logaritmování jsou ze zjevného důvodu nuly. Možná čtenáře napadne nahradit nuly znakem pro chybějící hodnotu (typicky *NA*) v duchu myšlenky „Když hmotnostní spektrometr našel množství rovno nule, tak protein vlastně nenašel.“ – tím bychom se však připravili o část informace. Lze totiž rozlišovat důvody chybějící hodnoty (více v části 1.2.6), což by se nám však tímto krokem ztížilo. Alternativními transformačními metodami tak mohou v případě výskytu nul v datech být:

- Logaritmování dle vzorce $y = log_2(x+1)$ – nulové hodnoty tak nulovými zůstanou a benefity logaritmické transformace se zachovají.
- Odmocninová transformace – není tak dobře interpretovatelná a při extrémní šikmosti nesymetrizuje data dostatečně, ale nuly pro ni nejsou problém.

[11] Se základem dva – jak už bylo zmíněno výše.

1.2.3 Normalizace

Variabilita v proteomických datech pochází z různých zdrojů. Některé z nich jsou nežádoucí a můžeme je ovlivnit úpravou experimentálních podmínek (*batch efekt*), např. stejná šarže chemikálií či stejný laborant připravující vzorky. Jiné jsou žádoucí a leží v nich odpovědi na výzkumné otázky – zde jde třeba o variabilitu způsobenou rozdílem mezi jednotlivými skupinami (např. zdraví versus nemocní). V datech se ovšem vyskytuje také variabilita, které se nemůžeme zbavit zásahem do nastavení experimentu a kterou v nich zároveň mít nechceme, neboť nepřináší požadované informace. Jako příklad uveďme různou rychlost degradace biologických vzorků či opotřebení materiálu/senzorů v hmotnostním spektrometru. Právě kvůli snížení „míry kontaminace“ touto variabilitou proteomická data normalizujeme [19].

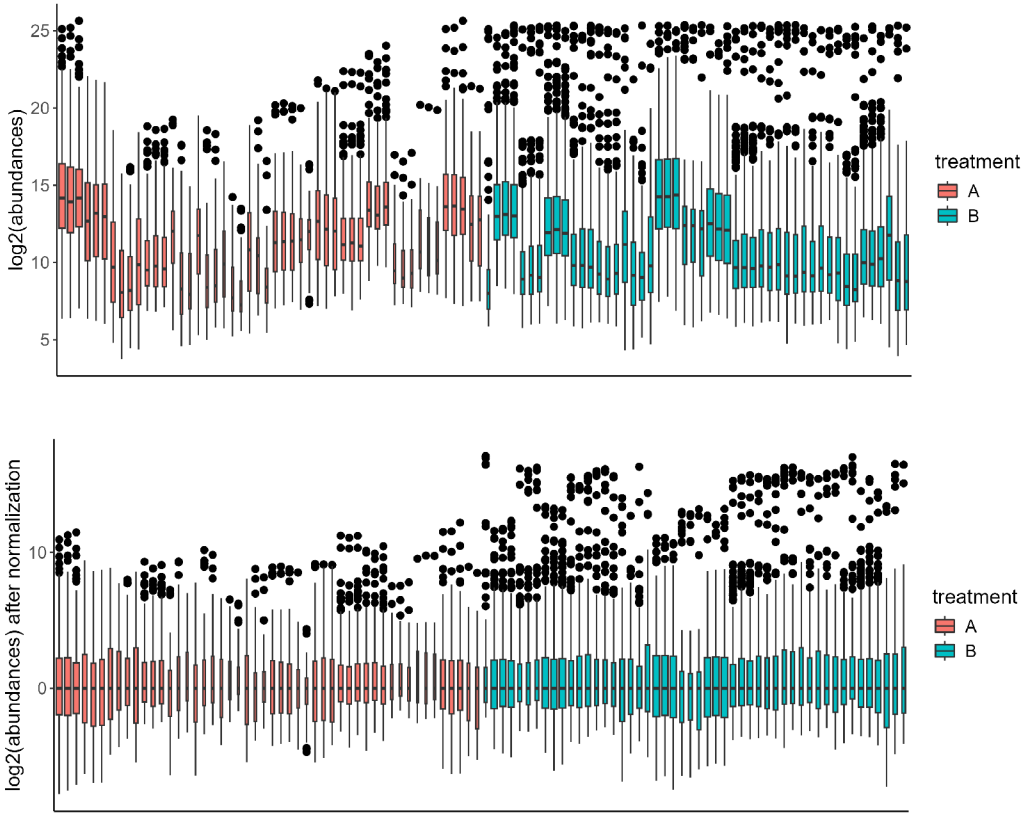
Při výběru metod normalizace je důležité volit takové, které sice nežádoucí efekt odstraní či redukují, avšak zároveň v datech zachovají informaci, kvůli které celý experiment provádíme. To se ukazuje být výzvou, neboť i v proteomice velmi často používané metody se dle některých autorů považují za nevhodné [19]. V této práci zmíníme tři metody s různým stupněm sofistikovanosti.

Tou první je **mediánová normalizace** – intuitivní metoda, která spočívá pouze v odečtení mediánu abundancí v rámci každého souboru. Při dodržení struktury datové sady dle Tabulky 1.1 s obecně m řádky a n sloupci získáme normalizované abundance y^*_{ij} namísto těch původních (ideálně už transformovaných – vizte část 1.2.2) y_{ij} pro každý sloupec $j = 2, \ldots, n$, jako

$$y^*_{ij} = y_{ij} - \text{median}\{y_{1j}, \ldots, y_{mj}\}, \forall i \in \{1, \ldots, m\}. \tag{1.1}$$

Tato metoda je citlivá k zachování „signálu“ v datech, pouze zarovná mediány

abundancí všech souborů na nulu – vizte Obrázek 1.3 [6].



Obrázek 1.3: Srovnání boxplotů transformovaných abundancí před mediánovou normalizací (nahoře) a po ní (dole)

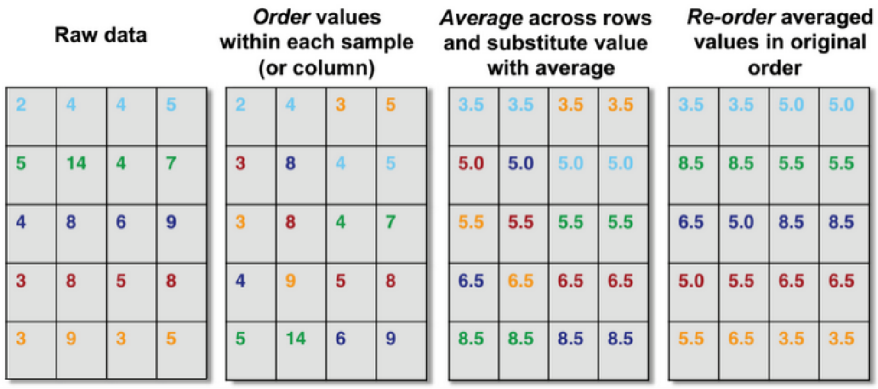
Další používanou metodou je **kvantilová normalizace**. Ta normalizuje datovou sadu tak, aby se srovnala pravděpodobnostní rozdělení abundancí jednotlivých souborů, a to právě přes rovnost kvantilů. Při vykreslení *quantile–quantile plotu* pro dva libovolné soubory s takto normalizovanými abundancemi pak body téměř tvoří přímku $y = x$. Algoritmus [2] této metody by se dal zapsat takto: Mějme matici $Y_{m \times n}$ s transformovanými abundancemi proteinů (sloupce tvoří hodnoty v jednotlivých souborech, řádky tvoří abundance pro konkrétní protein).

1. V každém sloupci seřaďte hodnoty od nejmenší po největší.
   - tuto novou matici označme Y'
2. Hodnoty v každém řádku matice Y' nahraďte řádkovým průměrem, tedy

$$y''_{ij} = \frac{1}{n} \sum_{j=1}^{n} y'_{ij}, \; \forall i \in \{1, \ldots, m\}, \; \forall j \in \{1, \ldots, n\}. \tag{1.2}$$

3. Uspořádejte sloupce matice Y'' tak, aby se každá hodnota vrátila do původního řádku – získáte matici normalizovaných abundancí Y^*.

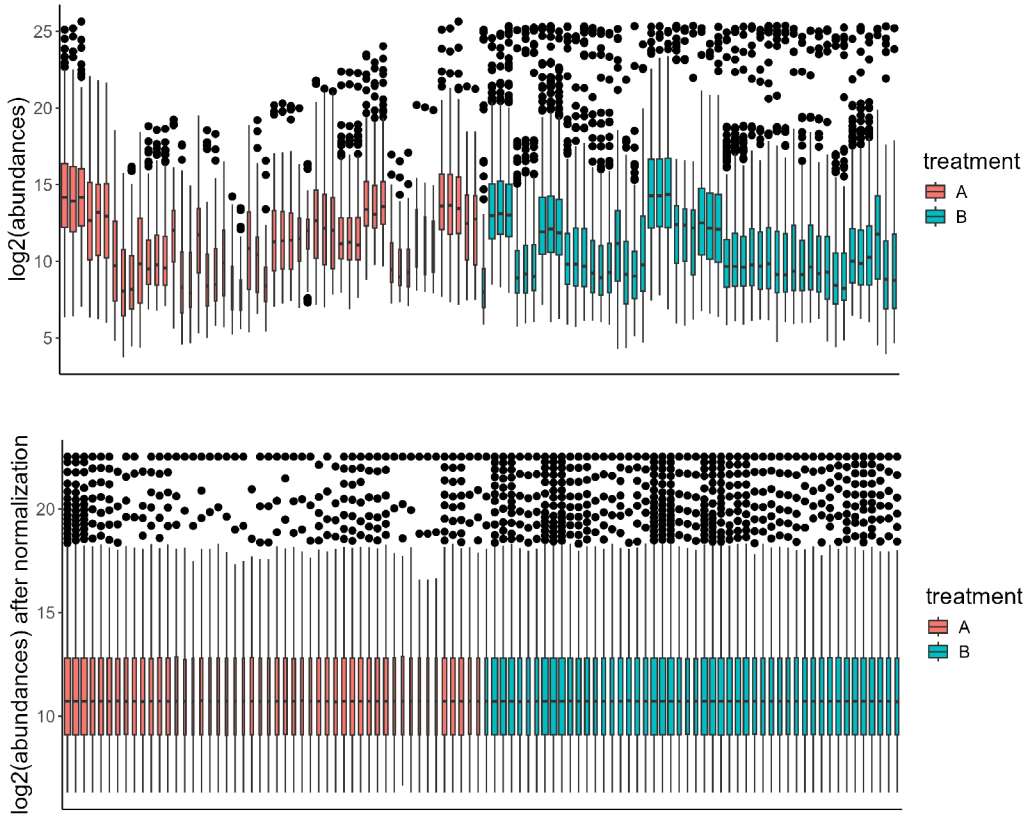
Příklad aplikace této metody můžete vidět na Obrázku 1.5.



Obrázek 1.4: Algoritmus kvantilové normalizace, zdroj: [29]

I přes velmi častý výskyt chybějících hodnot u proteomických dat se v literatuře při popisu této metody do algoritmu tato varianta nezahrnuje. Např. funkce `normalize.quantiles` z R balíčku `preprocessCore` [35] má v dokumentaci tvrzení, že si s *NA* poradí, avšak ani v jednom z článků uvedených v referenci se o chybějících hodnotách v tomto kontextu nic nepíše. Přesto se tento problém obejít dá. Jednou z možností je tyto buňky při seřazení hodnot

ve sloupci zařadit až na úplný konec (za největší hodnoty), řádkové průměry počítat bez zahrnutí těchto prázdných buněk (a ani je ničím nenahrazovat) a při zpětném uspořádání hodnot ve sloupci je tak vrátit do původního řádku nezměněné.



Obrázek 1.5: Srovnání boxplotů transformovaných abundancí před kvantilovou normalizací (nahoře) a po ní (dole)

Kvantilová normalizace však není příliš vhodná v případech, kdy se u některých proteinů koncentrují hodnoty do konců rozdělení abundancí (v porovnání s ostatními proteiny) – tedy jsou o dost větší či menší. V takovém případě by při použití kvantilové normalizace dostaly všechny hodnoty daného proteinu v rámci sloupců stejné (či téměř stejné) pořadí a mohlo by se

stát, že se abundance daného proteinu nahradí jejich průměrnou hodnotou, což by mohlo bránit správným výsledkům statistické inference [3].

Pro tyto případy si uvedeme třetí metodu, tzv. **median-balanced quantile normalization** (či zkráceně MBQN)[12]. Ta předpokládá, že každá hodnota z datové matice $Y_{m\times n}$ (vizte algoritmus kvantilové normalizace výše) je složena z několika složek, konkrétně takto:

$$y_{ij} = x_{ij} + o_i + b_j + \epsilon_{ij},\ \forall i \in \{1,\dots,m\},\ \forall j \in \{1,\dots,n\}. \tag{1.3}$$

Symbol x_{ij} představuje „skutečný“ biologický signál (o tuto informaci nám jde), o_i představuje člen specifický pro daný protein (proteinový posun), b_j je systematická chyba souboru a ϵ_{ij} je náhodná chyba měření, pro kterou platí: $\epsilon_{ij} \sim N(0, s^2)$. Myšlenka MBQN stojí na převedení abundancí proteinů na společnou škálu odečtením řádkového mediánu abundancí od každého proteinu, následně se provede výše popsaná kvantilová normalizace, po níž se řádkové mediány k normalizovaným datům opět přičtou. Algoritmus [3] této metody by tedy vypadal takto:

1. Pro každý řádek matice $Y_{m\times n}$ spočítejte odhad parametru o_i, $i \in \{1,\dots,m\}$ jako

$$\hat{o}_i = \text{median}\{y_{i1},\dots,y_{in}\}. \tag{1.4}$$

2. Odečtěte od každého řádku odpovídající odhad $\hat{o}_i$, čímž získáte upra-

[12]Pod zkratkou MBQN lze chápat i alternativní metodu, mean-balanced quantile normalization. Účel i použití obou metod jsou stejné, pouze se v algoritmu místo mediánu použije aritmetický průměr. Obě tyto metody zastřešuje pojem TRQN, tedy tail-robust quantile normalization [3].

venou matici Y' s nulovými řádkovými mediány:

$$y'_{ij} = y_{ij} - \hat{o}_i,\ \forall i \in \{1, \ldots, m\},\ \forall j \in \{1, \ldots, n\}. \tag{1.5}$$

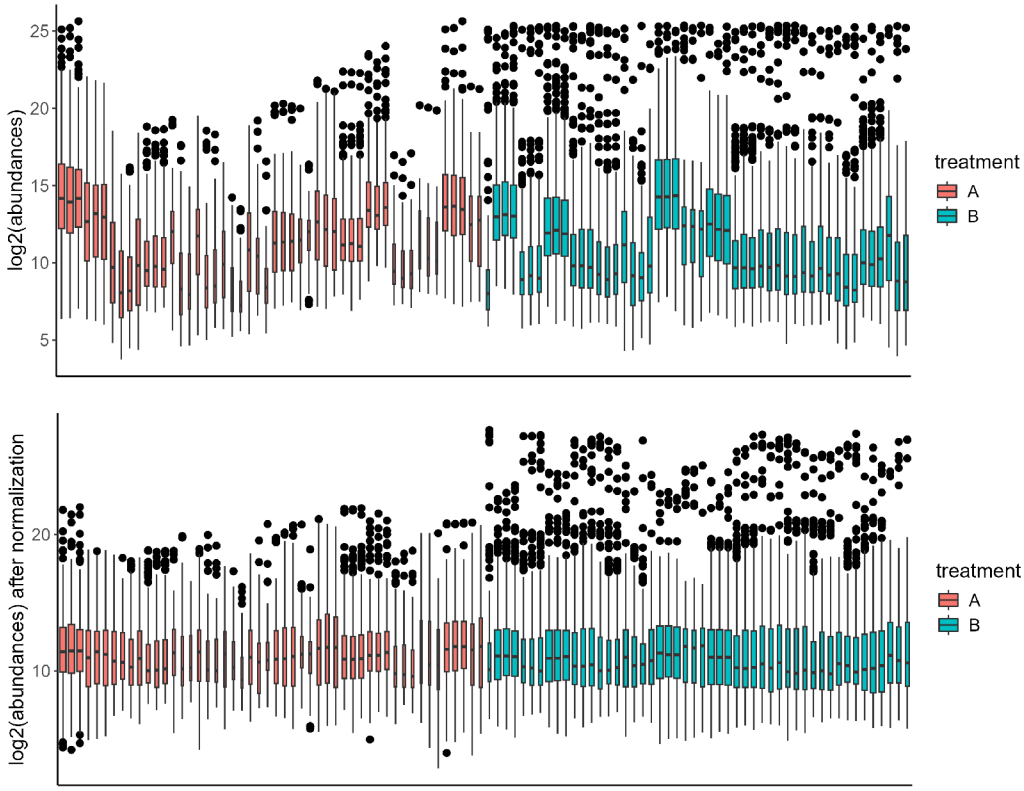
3. Proveďte kvantilovou normalizaci (dle algoritmu odpovídajícímu Obrázku 1.4) – nově vzniklou matici označme Y''.

4. K řádkům matice Y'' znovu přičtěte dříve spočítané odhady $\hat{o}_i$, čímž dostanete výslednou matici Y^* se znormalizovanými abundancemi, tedy

$$y^*_{ij} = y''_{ij} + \hat{o}_i,\ \forall i \in \{1, \ldots, m\},\ \forall j \in \{1, \ldots, n\}. \tag{1.6}$$

Posledním krokem algoritmu tak úrovně transformovaných abundancí přiblížíme původnímu škálování [3]. Efekt této metody normalizace můžete vidět na Obrázku 1.6.

V mnoha nástrojích pro zpracování proteomických dat (jakož i článcích zabývajících se touto problematikou) se vyskytují metody normalizace založené na přeškálování sloupců na konstantní součet. Idea stojí na faktu, že v každém vzorku je na počátku stejné množství (celkový objem) proteinů, jak již bylo zmíněno v části 1.1.2. Dle některých autorů [19] však tuto myšlenku nelze obhájit a při hlubším zkoumání nedává smysl ji používat ani z biologického, ani ze statistického hlediska. Podívejme se na tento problém především z pohledu statistika. Pokud každý sloupec přeškálujeme tak, aby byly sloupcové součty konstantní (zachovají se tedy poměry), dostanou se nám do dat náhodné korelace napříč jednotlivými soubory, které zašumí informaci, po níž se pídíme. Po přeškálování se z datové sady navíc stanou data kompoziční, s nimiž je nutné pracovat za využití metod a postupů, které jsou tomu speciálně uzpůsobené (pro „konvenční“ metody aplikované na tento typ dat



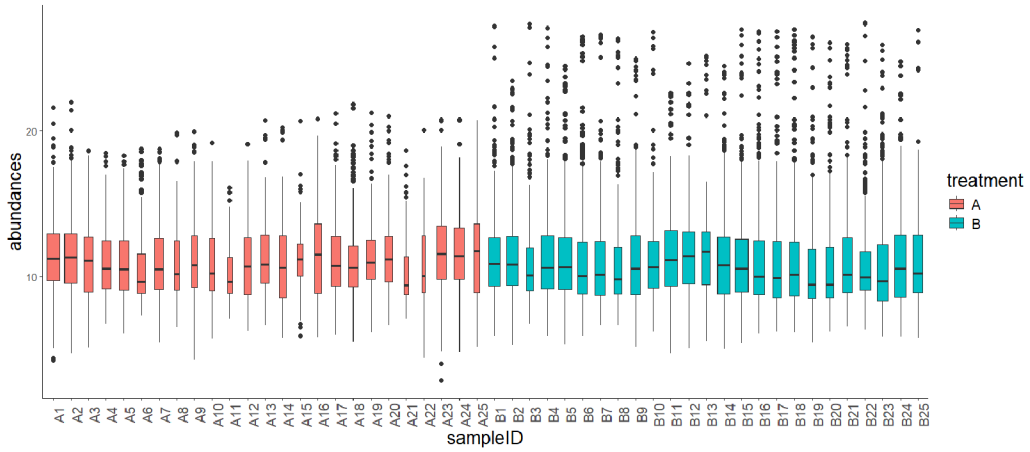
Obrázek 1.6: Srovnání boxplotů transformovaných abundancí před MBQN (nahoře) a po ní (dole)

nedostáváme smysluplné výsledky) [19]. Proto se tedy v tomto textu nebudeme normalizací dat na konstantní sumu dále zabývat.

1.2.4 Agregace

Dosud jsme se o abundancích proteinů bavili na úrovni jednotlivých souborů (či běhů), u nichž však víme, že mohou pocházet od stejného pacienta (vzorku). Při analýze (vizte část 1.2.7) tohoto typu dat však potřebujeme pracovat s abundancemi na úrovni vzorků. Je tedy potřeba údaje pro jednotlivé soubory agregovat dle vzorků, z nichž pochází. Pokud by se v datech

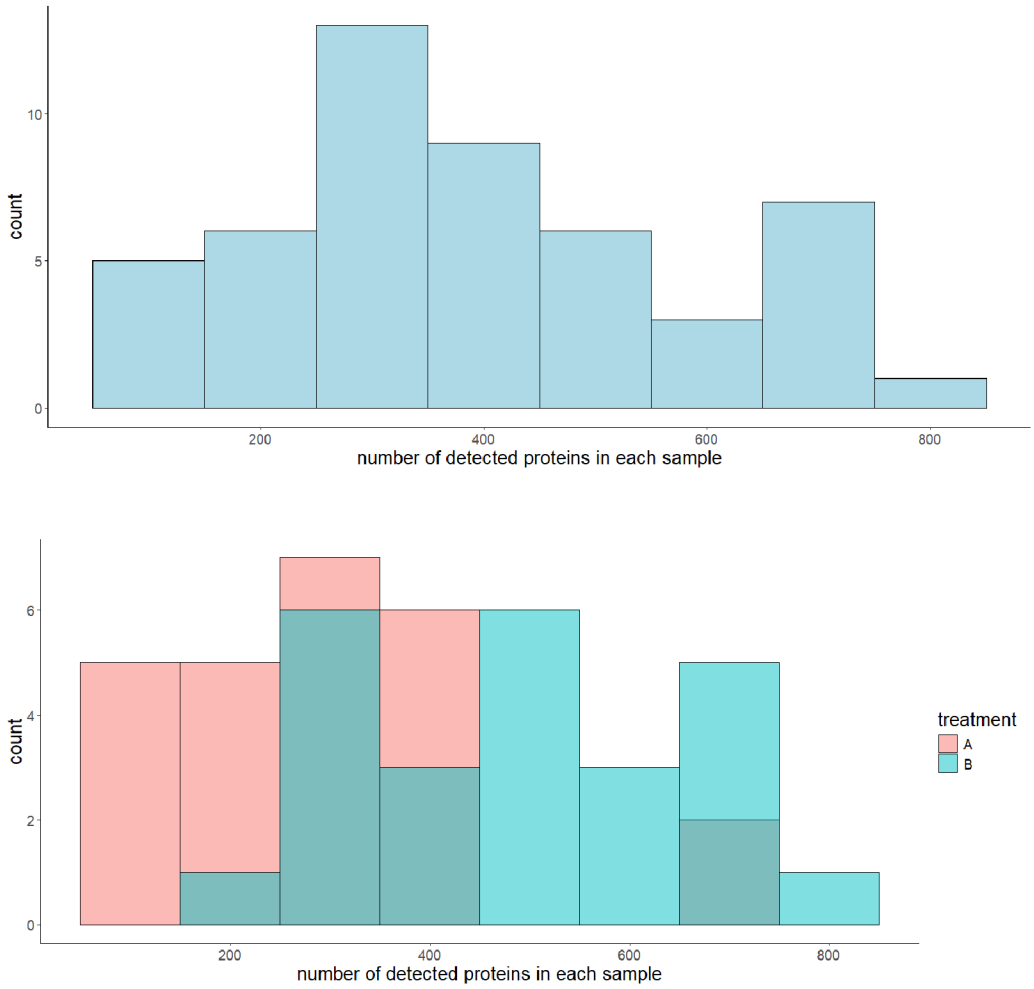
nevyskytovaly chybějící hodnoty, šlo by o jednoduchou úlohu – pouze bychom pro každý vzorek spočítali jeho abundanci jako průměr či medián abundancí příslušných souborů (replikátů) vzorku. Počet replikátů pro daný vzorek nebývá nijak velký (často dva až tři replikáty na jeden vzorek), ani výpočetně tak nejde o složitou věc. Chybějící hodnoty ovšem vnáší do celé problematiky nové světlo.



Obrázek 1.7: Boxplot abundancí pro jednotlivé vzorky (po transformaci a normalizaci souborů a jejich agregaci na úroveň vzorků)

Představme si, že máme pro každý vzorek tři replikáty. V případě, že chybí hodnoty u všech tří replikátů, bude i agregovaná hodnota pro daný vzorek *NA*. Pokud byl protein detekován ve všech třech replikátech, agregovanou hodnotou abundance vzorku se stane již zmiňovaný průměr či medián těchto tří hodnot. Co však dělat v situacích mezi těmito dvěma krajními? Pokud byl protein detekován pouze v jednom souboru ze tří, můžeme se spolehnout na to, že nešlo o chybu a ve vzorku se skutečně vyskytuje? A pokud ne, dva replikáty ze tří s přítomností proteinu už jsou dostačující? Na tyto otázky bohužel neexistuje jednoznačná odpověď. Povaha výzkumu, výsledek měření i zkušenosti/preference analytika mohou požadavky na agregaci měnit. Proto

je dobré nechat o odpovědích na výše uvedené dotazy rozhodnout člověka, který s daty pracuje a má do výzkumu určitý vhled.



Obrázek 1.8: Histogram počtu detekovaných proteinů po agregaci – bez rozdělení dle skupiny (nahoře) a s rozdělením dle skupiny (dole)

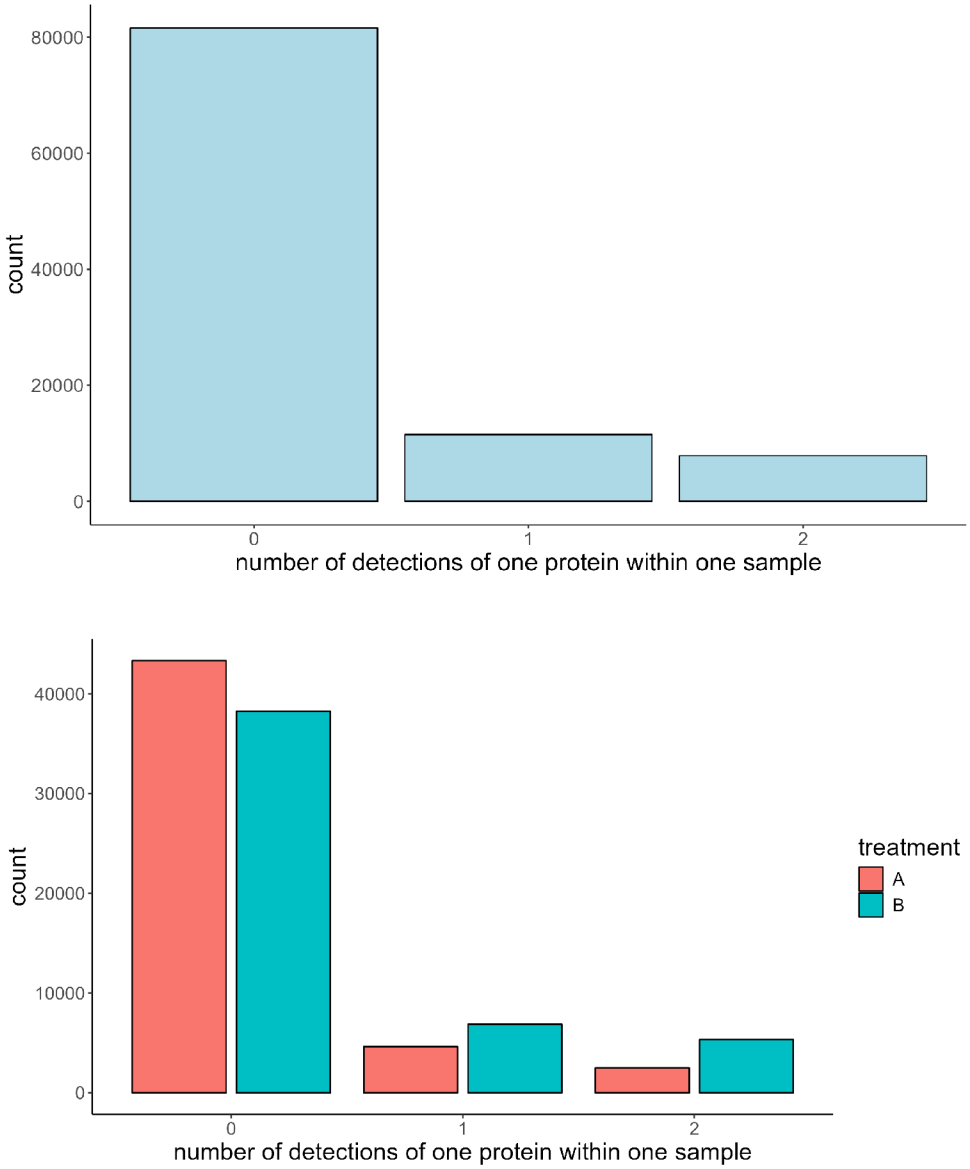
V některých případech můžeme agregovat abundance na úroveň vzorků i při detekci proteinu v pouhém jednom replikátu (to by se tedy pouze opsala jediná změřená hodnota v rámci vzorku), jindy můžeme požadovat „100%“ detekci pro spočítání abundance proteinu ve vzorku a v opačném

případě (v některém replikátu vzorku se protein nenašel) zapíšeme chybějící hodnotu. Případně cokoliv mezi tím. Výsledky analýzy mohou být ke způsobu provedení agregace poměrně citlivé, s absencí jednoznačného stanoviska a „správného“ postupu při agregaci tak můžeme zkoušet více cest a pozorovat, jak to závěry ovlivní.

Pro prozkoumání datové sady po agregaci můžeme použít podobné vizualizace, které jsme použili už na úrovni souborů, pouze přepočítané pro agregovaná data (vizte Obrázky 1.7 a 1.8). Kromě toho je ovšem vhodné mít k dispozici také grafy, které nám pomohou při rozhodování ještě před samotnou agregací (právě při určování parametrů agregace). Jednoduchou vizualizací je v tomto případě sloupcový graf s počtem detekcí pro každou dvojici protein–vzorek[13]. Jak už bylo zmíněno, v každém vzorku může být (v závislosti na počtu replikátů) daný protein detekován maximálně $d_j$krát, kde d_j je počet replikátů j-tého vzorku. Na základě souhrnných četností těchto počtů detekcí můžeme vytvořit graf podobný těm na Obrázku 1.9. Nabízí se jak celkový pohled, tak rozdělení dle skupin léčby. Tyto vizualizace nám zároveň ukazují, o kolik pozorování se připravíme, pokud při agregaci budeme požadovat zvolené procento detekcí v rámci vzorku pro agregaci abundancí. Zůstaňme ještě u Obrázku 1.9 – pokud si zvolíme, že se abundance proteinu ze souborů agregují do hodnoty vzorku pouze v případě 100% detekce (na uvedeném příkladu dva ze dvou souborů vzorku s detekcí), přijdeme tak o všechna vyčíslená pozorování ve sloupci s označením „1“ – tedy o ty dvojice protein–vzorek, kde máme detekovanou abundanci jen v jednom souboru ze dvou. Pokud máme tu možnost, můžeme opět přidat interaktivní infobox s uvedeným absolutním i relativním počtem (celkově či v rámci skupiny).

K dalšímu, podrobnějšímu pohledu na tutéž informaci může sloužit tzv.

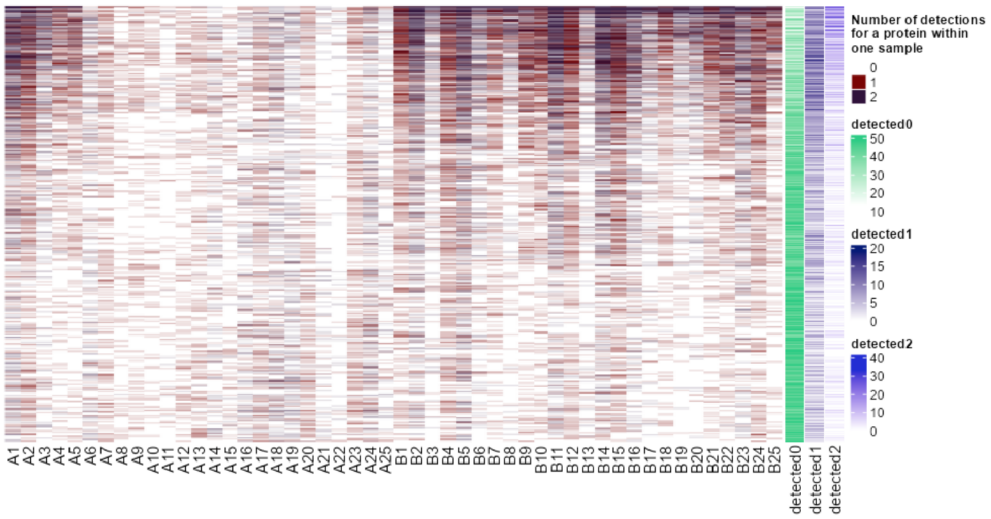
[13]Součet výšek jednotlivých sloupců je roven součinu celkového počtu proteinů a celkového počtu vzorků.



Obrázek 1.9: Barplot počtu detekcí proteinu v rámci vzorku – bez rozdělení dle skupiny (nahoře) a s rozdělením dle skupiny (dole)

heatmapa (vizte Obrázek 1.10) zobrazující pro každou dvojici protein–vzorek to, v kolika replikátech daného vzorku byl protein detekován. Zde tedy máme

k dispozici detailní pohled na celou „matici detekcí“, z které se počítaly i sloupcové grafy na Obrázku 1.9.



Median of samples with a given number of protein detections (overall and by treatment group)

detected	overall	A	B
0	42	22	21
1	5	2	3
2	2	1	1

Obrázek 1.10: Heatmapa s počty detekcí proteinu v rámci vzorku zahrnující souhrnné anotační sloupce pro jednotlivé počty detekcí a tabulku s mediánem počtu vzorků s danou četností detekce

K základní heatmapě jsou navíc připojeny anotační sloupce s řádkovými součty pro jednotlivé počty detekcí – ty ukazují, kolikrát byl daný protein detekován v rámci vzorku nulakrát, jednou atd. Z těchto anotačních sloupců můžeme spočítat libovolnou číselnou charakteristiku (např. průměr či medián) a připojit tuto informaci ke grafu ve formě tabulky – opět možno souhrnně či dle skupin. Při celkovém pohledu na tento typ vizualizace se pak opět můžeme lépe rozhodnout, jak „přísní“ při agregaci budeme.

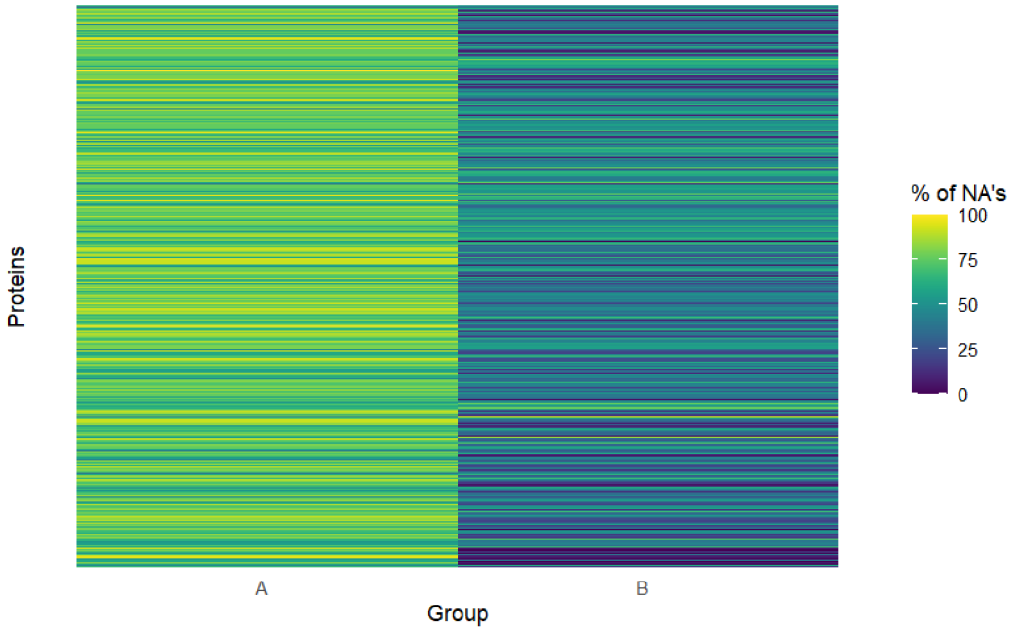
1.2.5 Filtrace

Jakmile máme data agregována na úroveň vzorků (poté, co jsme je na úrovni souborů transformovali a normalizovali), jsme „téměř“ připraveni k analýze za účelem nalezení kandidátních proteinů (výše zmíněné *DEP*). Proč téměř? V datovém souboru se totiž stále vyskytuje jev, který jakoukoliv práci s daty významně komplikuje – chybějící hodnoty. O možnostech jejich imputace se budeme bavit v části 1.2.6, v této podkapitole se však zaměříme na krok, který nám pomůže množství chybějících hodnot v datech zredukovat. Tímto krokem je filtrace.

Filtraci v této fázi práce s proteomickými daty aplikujeme na řádky datového souboru, filtrujeme tedy proteiny, které nebyly detekovány v dostatečném množství souborů. Nikde není stanovená přesná hranice, co je oním „dostatečným množstvím“ – znovu tedy záleží na zkušenostech a záměrech výzkumného pracovníka. Řekněme, že požadavky na množství detekcí proteinu ve vzorcích budeme vyjadřovat v procentech. Hypotetický vědec chystající se na filtraci dat si tedy řekne: „Ponechám v datech jen ty proteiny, které byly detekovány alespoň v 75 % vzorků.“ V tomto momentě je však důležité se zamyslet, jaký je cíl analýzy. V případě, že máme vzorky rozlišeny dle skupiny léčby (např. nemocní a kontroly), toto úzké uvažování (založené na jedné hodnotě aplikované na celá data) by mohlo být nedostatečné.

V některých případech se samozřejmě výše uvedený postup aplikovat dá, avšak někdy je vhodnější zahrnout do rozhodovacího procesu právě skupiny. Můžeme v datech např. ponechat pouze proteiny, které byly detekovány alespoň v n % vzorků **v každé ze skupin**. Cílem může být dostatečné množství dat pro analýzu zaměřenou na porovnání abundancí proteinů v těchto skupinách. Další cestou může být ponechání proteinů detekovaných alespoň v n % vzorků **v alespoň jedné skupině**. Zde nám nevadí třeba i naprostá absence

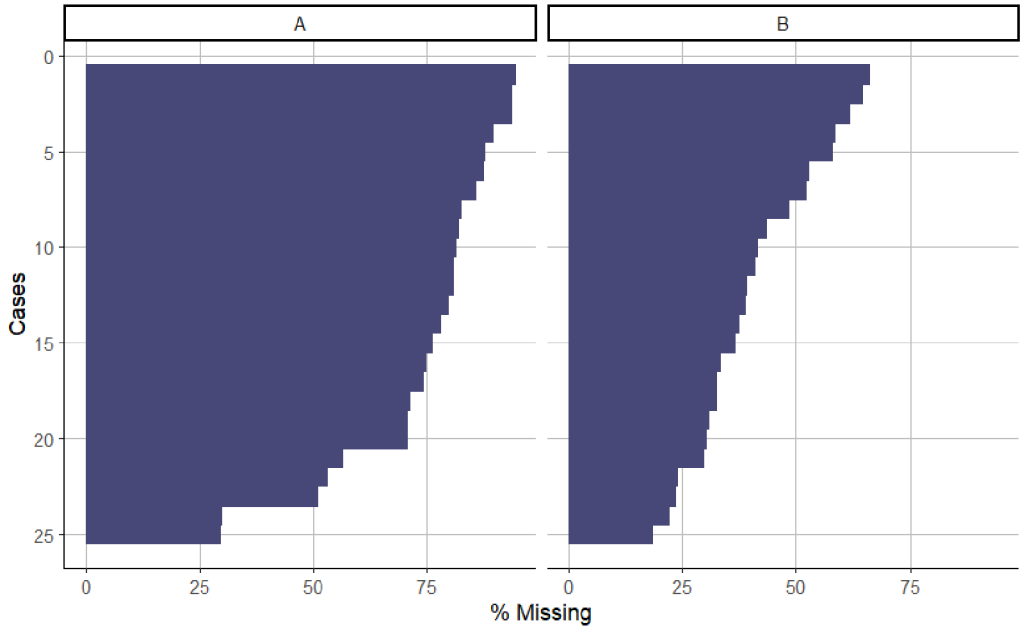
proteinu ve vzorcích jedné skupiny, protože při dostatečném množství detekcí ve skupině druhé to může ukazovat na to, že protein je např. specifický pro zkoumanou nemoc (zatímco u zdravých jedinců se nevyskytuje).



Obrázek 1.11: Vizualizace procenta chybějících hodnot na úrovni proteinů (dle skupiny léčby)

Při volbě n coby minimálního požadovaného procenta detekcí proteinu (ať už v kterémkoliv ze tří výše zmíněných pohledů) si opět můžeme pomoct jak vhodnou vizualizací, tak třeba i jen obyčejným číselným údajem – kolik řádků (či jaké procento z nich) si při konkrétní hodnotě n z dat odfiltrujeme. Co se vizualizace týče, můžeme se na chybějící hodnoty podívat z obou úrovní. Relativní zastoupení chybějících hodnot na úrovni proteinů (tedy kolik procent vzorků je v rámci každého proteinu nedetekovaných) je zobrazeno na Obrázku 1.11. Stejně tak můžeme z totožné datové matice vytvořit graf s relativním zastoupením *NA* hodnot na úrovni jednotlivých vzorků (tedy kolik procent proteinů je v rámci každého vzorku nedetekovaných). Ukázka

takového grafu je na Obrázku 1.12. U obou grafů je vhodné rozlišovat skupinu léčby, obzvlášť tehdy, pokud tento faktor zapojujeme do způsobu filtrace (vizte výše).



Obrázek 1.12: Vizualizace procenta chybějících hodnot na úrovni vzorků (dle skupiny léčby)

1.2.6 Imputace

Odfiltrováním nedostatečně detekovaných proteinů sice výchozí situaci před samotnou analýzou vylepšíme, přítomnost chybějících hodnot však stále může negativně ovlivnit závěry – ať už jde o snížení statistické síly testů či nedostatečně zastoupené vzorky (obzvlášť při rozdělení na skupiny s cílem jejich porovnání). Proces, kdy zvolenou metodou „zaplňujeme díry“ v datech (tedy v našem případě nahrazujeme *NA* číselnou hodnotou), nazýváme imputace. Než si však uvedeme příklady metod sloužících k imputaci abun-

dancí, podívejme se podrobněji na původ a vlastnosti chybějících hodnot, které hrají při volbě vhodné metody významnou roli [15].

Chybějící hodnoty v proteomických datech můžeme rozdělit na tři základní kategorie:

- chybějící zcela náhodně (MCAR – z anglického *missing completely at random*)
- chybějící náhodně (MAR – z anglického *missing at random*)
- chybějící nenáhodně (MNAR z anglického *missing not at random*)

Výskyt MCAR je zcela nezávislý na pozorovaných hodnotách a je způsoben např. nepřesně seřízeným přístrojem či chybnou přípravou vzorků. MAR jsou zobecněním MCAR [14] a zahrnují např. důsledky špatně přiřazeného peptidu k proteinu (vizte část 1.1.2), malou efektivitu ionizace, nesprávně provedené štěpení na peptidy a jiné faktory biologického či technického charakteru. Přítomnost MNAR je naopak s hodnotou abundancí úzce svázaná – jde o situaci, kdy se protein ve vzorku nacházel v množství menším než je detekční limit přístroje (příp. se tam nevyskytoval vůbec). Některé metody imputace jsou vhodné spíše na MNAR, jiné na MAR a MCAR. Cílem vědců je zvolit pro konkrétní datovou sadu takovou metodu, která bude mít nejmenší negativní dopad na následnou analýzu (což se ukazuje být velkou výzvou) [9][15][18].

Poměr výskytu náhodně a nenáhodně se vyskytujících chybějících hodnot se těžko odhaduje, v článku [12] však předkládají názor, že se obecně věří v to, že MNAR v proteomických datech dominují. Důležité však je, že bývají přítomné náhodné i nenáhodné *NA* – proto se při výběru metod nelze spoléhat pouze na pokrytí jedné kategorie (resp. vždy vyvažujeme riziko vzniku chyb na jedné a druhé straně). Zároveň je vhodné podívat se podrobněji na

[14] V některé literatuře tyto dva pojmy nerozlišují a používají pouze jeden z nich [12][18].

to, jaké typy metod pro imputaci máme k dispozici a na jaký typ chybějících hodnot jsou vhodnější. V článku [15] rozlišují tyto 4 typy imputačních metod:

1. naivní imputace
2. imputace založená na pozorovaných proteinech
3. imputace založená na globální podobě datové sady
4. souborná imputace (zahrnující více modelů/technik v jednom)

Naivní imputace spočívá v nahrazení všech chybějících hodnot konstantou či náhodně generovaným číslem a je tak nejjednodušší a nejrychlejší metodou. Dobrých výsledků dosahuje při dominanci MNAR v datech – typicky jde o nahrazení malou hodnotou odpovídající tomu, že abundance proteinu byla spíše v levé části rozdělení abundancí (např. pod detekčním limitem). Při výskytu MCAR a MAR se však tyto metody spíše nehodí. Druhá kategorie imputačních metod už využívá dostupné hodnoty z ostatních proteinů a patří sem jak různé regresní postupy, tak např. metody založené na „nejbližších sousedech“ – vizte dále. Imputace využívající globální strukturu dat stojí na metodách jako PCA (*principal component analysis*, tedy metoda hlavních komponent) či rozklad na vlastní čísla. Souborná imputace má zase blízko k *machine learningu* (ML). Tyto metody (kategorie 2-4) se naopak hodí spíše pro imputaci MAR a MCAR [15].

To, pro jakou metodu se rozhodneme, závisí na několika faktorech. Kromě již zmiňovaného typu chybějící hodnoty jde např. o množství vzorků, rozdělení dat či proporci chybějících hodnot. Imputace založená na ML přístupu potřebuje více dat (a tedy více vzorků), naopak naivní imputace či metody založené na pozorovaných proteinech si poradí i s menším počtem vzorků.

Dále existuje několik přístupů (zejména v kategoriích 2 a 3) vyžadujících normálně rozdělená data. Co se relativního zastoupení *NA* týče, v případě výskytu chybějících hodnot nad 30 % selhávají souborné imputační metody, nad 50 % už je pak víceméně jedno, kterou metodu použijeme (tendenci ke špatným výsledkům mají u takto děravých dat všechny), přičemž platí: čím více chybějících hodnot, tím jednodušší přístup volit k jejich imputaci [15].

Uveďme nyní alespoň tři příklady metod imputace[15]. U „nejjednodušší“ kategorie těchto metod (naivní imputace) figuruje již zmíněná síla při imputaci nenáhodných *NA* a špatná výkonnost u doplnění náhodných chybějících hodnot. Dle článku [15] však dosahuje překvapivě dobrých výsledků u smíšeného typu dat (s výskytem MCAR, MAR i MNAR) metoda imputace pomocí minima vzorku (*SampMin*). Ta každé chybějící dvojici protein–vzorek přiřadí takovou hodnotu abundance, která je z přítomných hodnot v daném vzorku (sloupci) nejmenší. Tato naivní metoda by měla významně převyšovat ostatní imputační přístupy ze stejné kategorie [15].

Jako druhý příklad uveďme metodu k nejbližších sousedů (*kNN* – z anglického *k-nearest neighbors*). Má více variant, my však v tomto textu zmíníme tu založenou na euklidovské vzdálenosti proteinů. Pro imputaci chybějících hodnot proteinu se zde použije vážený průměr abundancí jeho nejbližších sousedních proteinů. Ti se hledají na základě abundancí u těch vzorků, kde byl detekován jak imputovaný protein, tak potenciální „sousedi“ – právě z této podmatice (vybrané řádky a sloupce) se počítají euklidovské vzdálenosti od imputovaného řádku/proteinu. Více je algoritmus v kontextu proteomických dat rozebrán například v textu [22]. V R můžeme pro imputaci pomocí *kNN* použít funkci `impute.knn` z balíčku `impute` [41]. Výsledky metody jsou samozřejmě ovlivněny parametrem k, který určuje, z kolika nejbližších proteinů

[15] Podrobněji se tomuto tématu včetně rozhodovacího procesu volby imputační metody věnuje článek [15].

hodnotu abundance počítáme[16] – dle některých studií to však nemá takový vliv, jak by se na první pohled mohlo zdát, resp. výsledky by při malých změnách parametru měly být víceméně podobné [9][15][22].

Poslední metodou imputace, která bude v tomto textu zmíněna, jsou tzv. náhodné lesy (*RF* – z anglického *random forests*). Tento přístup v sobě kombinuje výsledky z několika rozhodovacích stromů a patří tak do kategorie souborných imputačních metod. Jednotlivé rozhodovací stromy se sestaví z náhodně zvolených výběrů z datové sady, pro doplnění chybějících hodnot se výsledky zprůměrují – tento proces se opakuje v několika iteracích, dokud není naplněno zastavovací kritérium (typicky založené na rozdílu mezi aktuálním a předchozím krokem – datovými maticemi s imputovanými hodnotami). Pro imputaci v softwaru R je k dispozici funkce `missForest` ze stejnojmenného balíčku, která využívá algoritmus[17] podrobně popsaný v článku [21] – pro hlubší pochopení této metody se tedy odkazuji na něj. Zastavovací kritérium této funkce je založeno na výpočtu vzdálenosti matic aktuálního a předchozího kroku dle vzorce

$$\Delta = \frac{\sum(\mathbf{X}_{new}^{imp} - \mathbf{X}_{old}^{imp})^2}{\sum(\mathbf{X}_{new}^{imp})^2}, \tag{1.7}$$

přičemž k zastavení dojde po první iteraci, při které hodnota Δ vzroste. Náhodné lesy vykazují vynikající výsledky pro imputaci chybějících hodnot typu MAR a MCAR, při smíšeném původu *NA* (typickém pro reálná data) si však také nevede vůbec špatně a ze srovnávacích studií vychází jako jedna z nejvhodnějších metod pro proteomická data při neznalosti poměru náhodně a nenáhodně chybějících hodnot. Její nevýhodou je sice větší časová a výpo-

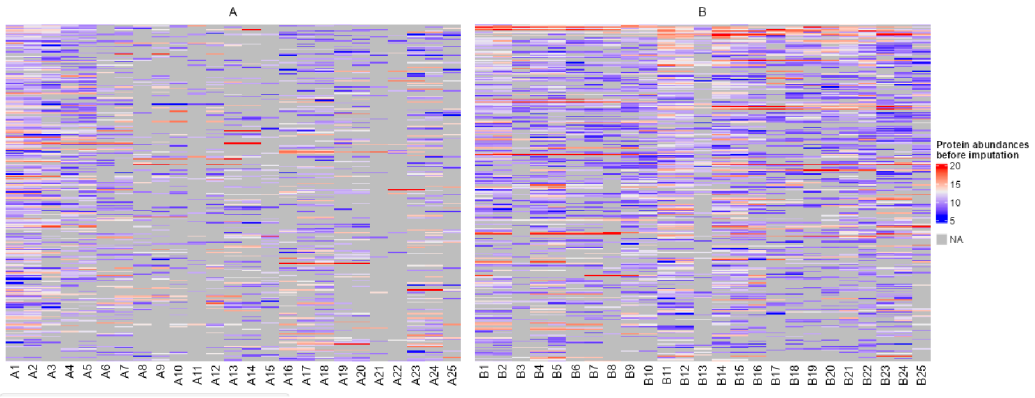
[16] Zmíněná funkce `impute.knn` používá defaultně hodnotu 10.

[17] Algoritmus předpokládá datovou sadu v podobě „řádky=pozorování, sloupce=proměnné“, pro použití této funkce na datovou sadu se strukturou popsanou v tomto textu je tak třeba data transponovat.

četní náročnost oproti jiným metodám, při dnešním výkonu počítačů se však tyto rozdíly stávají zanedbatelnými [12][15].

Zajímavým vylepšením imputačních metod různých typů by mohlo být rozdělení procesu doplňování chybějících hodnot dle skupiny léčby, které by v případě směřování analýzy k porovnávání skupin mohlo značně vylepšit její výsledky, jak navrhují autoři článku [18]. Přebudování algoritmů (jak z výše uvedených příkladů, tak i dalších metod zmíněných přehledně například v textu [15]) tak, aby skupinu léčby zohledňovaly, a jejich zprogramování pro následné použití v SW by tak mohlo být přínosem pro tento obor.

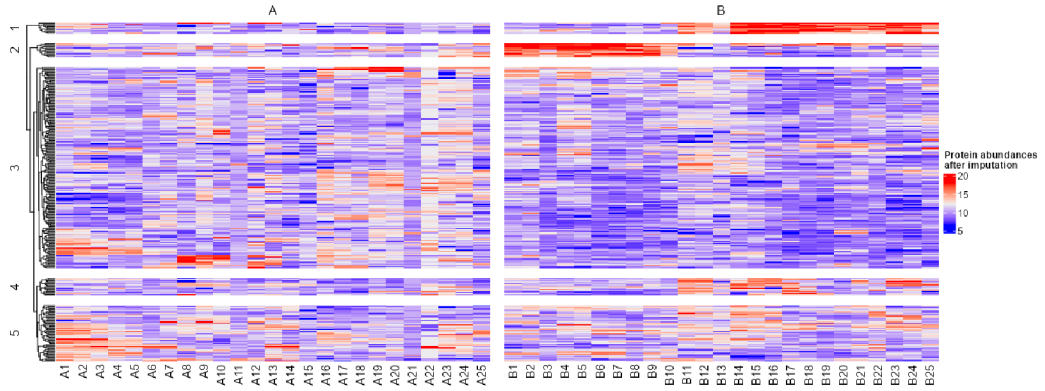
I při imputaci dat je užitečné mít k dispozici vizualizaci zobrazující, jaký efekt bude proces doplnění chybějících hodnot mít na podobu datové sady. U rozsáhlých datových souborů (jakými proteomická data určitě jsou) se dají využít heatmapy abundancí. Heatmapu s abundancemi před imputací můžete vidět na Obrázku 1.13.



Obrázek 1.13: Heatmapa abundancí proteinů před imputací

Na Obrázku 1.14 už jsou data imputována, konkrétně pomocí *kNN* s parametrem $k = 10$. V případě dostupnosti interaktivních nástrojů lze volit různé imputační metody a jejich parametry a pozorovat, jaký efekt to na datovou sadu má. Doplňující informaci poskytne rozdělení řádků a sloupců

způsobem, jakým to vidíme právě na Obrázku 1.14. Sloupce (vzorky) jsou rozděleny dle skupiny léčby, řádky (proteiny) tvoří shluky vytvořené metodou hierarchického shlukování[18], kterou používá R balíček `ComplexHeatmap` [10] použitý pro vytvoření tohoto obrázku. Díky rozdělení obou dimenzí se nám data (při vhodně zvolené imputaci a existujícím signálu v datech) rozdělí na skupiny proteinů, které se liší v rámci skupin léčby – což může být přesně to, co výzkumného pracovníka zajímá.



Obrázek 1.14: Heatmapa abundancí proteinů po imputaci

1.2.7 Analýza

Všechny předchozí části podkapitoly 1.2 byly důležité a byly v nich rozebrány podstatné kroky při práci s proteomickými daty. Nicméně, všechny by se daly brát jako „příprava“ pro tu fázi zkoumání, kvůli které se tímto tématem zabýváme. Tou fází je statistická analýza dat. V kontextu tohoto textu jde o porovnání abundancí proteinů v jednotlivých skupinách léčby a nalezení kandidátních proteinů (*DEP*), jejichž následné zkoumání může přinést

[18] `ComplexHeatmap` používá při shlukování euklidovskou vzdálenost pro výpočet matice vzdáleností a metodu *complete linkage* pro určení vzdálenosti mezi dvěma shluky.

zajímavé poznatky (nejen) k léčbě různých onemocnění. Detailní rozbor možných postupů, metod a statistických testů by sám vydal na další diplomovou práci, v této části si tak uvedeme spíše jednoduché příklady a specifika analýzy proteomických dat a představíme jednu z nejpoužívanějších vizualizací výsledků takového porovnání – tzv. *volcano plot*[19].

V situaci, kdy máme v datové sadě pouze dvě skupiny (např. zdraví a nemocní), je nejjednodušší variantou dvojice testů, která se učí už v základních kurzech statistiky – dvouvýběrový t-test[20][14] a dvouvýběrový Wilcoxonův test [5]. Oba testy se dají použít pro porovnání distribucí, což je přesně to, co nás u prováděné analýzy zajímá – volba mezi nimi je pak spíše individuální[21]. Při interpretaci výsledků používáme rozdíl skupinových mediánů. Zvolený test provádíme postupně pro každý protein (řádek datové matice) s pozorováními rozdělenými dle skupiny léčby. V případě, že máme v datech m proteinů, aplikujeme zvolený test m-krát, obdržíme m p-hodnot – to vše s nepříjemným důsledkem v podobě problému násobného testování hypotéz. Tímto postupem totiž hrozí, že mnoho ze „statisticky významných“ výsledků[22] jsou tzv. falešně pozitivní výsledky. V literatuře věnující se proteomice se často používá sledovaný parametr *FDR* (z anglického *false discovery rate*) zachycující očekávanou proporci falešně pozitivních výsledků v množině proteinů označených za *DEP* – tuto hodnotu chceme samozřejmě co nejnižší [14].

Zmíněný problém se dá částečně eliminovat adjustací p-hodnoty, nejznámější Bonferroniho korekce však při takovém množství hypotéz není vhodná,

[19] Český ekvivalent „vulkánový graf“ se používá spíše zřídka.

[20] Pod tento pojem nyní pro jednoduchost zařaďmě i jeho variantu pro případ různých skupinových rozptylů – Welchův test.

[21] Oba testy mají samozřejmě jiné teoretické předpoklady, avšak v praxi se u proteomických dat (i kvůli malému množství pozorování) ne vždy ověřují.

[22] Na zvolené hladině významnosti α.

neboť při posuzování statistické významnosti na hladině $\frac{\alpha}{m}$ pro m v řádech tisíců by nám prakticky žádný protein jako statisticky významně se lišící v abundanci mezi skupinami nevyšel [14]. V proteomice se proto častěji používá adjustace p-hodnoty metodou Benjamini–Hochberg [1], jejíž algoritmus je přehledně popsán např. v příspěvku [30]. Po získání adjustovaných p-hodnot si pak můžeme proteiny seřadit od nejnižší takto upravené p-hodnoty („nejvýznamněji se lišícího“ proteinu v rámci skupinového porovnání) – tento seřazený seznam můžeme považovat za hlavní výsledek analýzy zobrazující nejnadějnější kandidáty, jejichž další zkoumání by mohlo zlepšit znalosti o zkoumané nemoci.

Pokud máme v datech více skupin léčby (např. zdraví, lehká forma nemoci, těžká forma nemoci), můžeme rozdíl v abundanci proteinů napříč těmito skupinami testovat třeba pomocí jednofaktorové analýzy rozptylu (ANOVA) [11] či Kruskalova–Wallisova testu [11]. Znovu využíváme v obou variantách raději rozdíl skupinových mediánů. V situaci více skupin navíc vyvstává úkol zjistit, které skupiny se v abundanci proteinů významně liší (neboli je třeba provést mnohonásobné porovnávání). U ANOVA lze použít např. Tukeyho metodu [11], po testování pomocí Kruskalova–Wallisova testu je zase k dispozici porovnání dvojic Dunnovým testem [7], který je v R k dispozici v balíčku `rstatix` [42] pod příkazem `dunn_test()`. I zde je vhodné po otestování všech proteinů spočítat adjustované p-hodnoty[23] a jako výstup analýzy vytvořit tabulku s řádky seřazenými vzestupně dle adjustované p-hodnoty. Jako příklad takového výstupu uveďme Tabulku 1.7. Kromě tabelárního zobrazení výsledků je vždy na místě pokusit se o názornou vizualizaci – zde se přímo nabízí zmiňovaný *volcano plot*. Ten vykreslujeme pro příslušnou dvojici

[23] A adjustované p-hodnoty je dobré brát jako rozhodující i při určování významně se lišících dvojic při mnohonásobném porovnávání – v R tyto funkce často adjustovanou p-hodnotu přímo vrací, ale vždy je to třeba ověřit v dokumentaci konkrétní funkce.

Tabulka 1.7: Možné zobrazení výsledků analýzy

Accession	medianA	medianB	medianC	diffmedianAB	diffmedianAC	diffmedianBC	pvalue	adj_pvalue	signif_diff
P01040	10,628	16,752	14,893	-6,125	-4,265	1,859	0,000	0,001	B-A,C-A
H6VRF8	10,829	16,100	14,182	-5,271	-3,353	1,918	0,000	0,003	B-A,C-A
P19961	9,419	12,624	13,943	-3,205	-4,525	-1,319	0,001	0,015	C-A
B7ZMD7	11,293	14,007	14,439	-2,714	-3,146	-0,432	0,011	0,068	C-A
P13645	12,902	15,391	14,101	-2,489	-1,199	1,289	0,084	0,262	*NA*

skupin, u datových souborů se třemi a více skupinami léčby tak potřebujeme jeden takový graf pro každý skupinový pár. Každý řádek z výsledkové tabulky (resp. každý protein) představuje v tomto grafu jeden bod. Na ose x je rozdíl[24] skupinových mediánů, hodnotu „nulového rozdílu“ mezi skupinami zde tedy představuje číslo nula[25]. Často chceme mít absolutní hodnotu tohoto rozdílu alespoň v nějaké výši (volba konkrétní hodnoty je velmi individuální), můžeme tedy v daných x-ových souřadnicích vykreslit svislé čáry vymezující proteiny dostatečně se lišící v rámci obou skupin. Na ose y lze při použití dvouvýběrového testu vykreslit původní či adjustovanou p-hodnotu, avšak transformovanou tak, aby nevýznamné výsledky stlačila k nule a naopak zvýraznila malé p-hodnoty poukazující na statisticky významný rozdíl. Touto transformací může být např. záporně vzatý logaritmus[26], tedy:

$$p^\star = -\log_{10}(p), \tag{1.8}$$

kde p je (původní či adjustovaná) p-hodnota a $p^\star$ její transformovaná hodnota vykreslovaná na ose y. I zde pracujeme se zvolenou hladinou významnosti a v příslušné výšce (po transformaci naší hladiny významnosti způsobem, jako jsme to provedli s p-hodnotou) můžeme vykreslit vodorovnou čáru –

[24]Někdy se místo rozdílu vykresluje tzv. *fold change*, tedy podíl skupinových mediánů, příp. $\log_2$(fold change) – čímž jednoduchou úpravou dostaneme rozdíl logaritmovaných skupinových mediánů. My však v této fázi předpokládáme, že máme data transformovaná, proto si vystačíme s pouhým rozdílem.

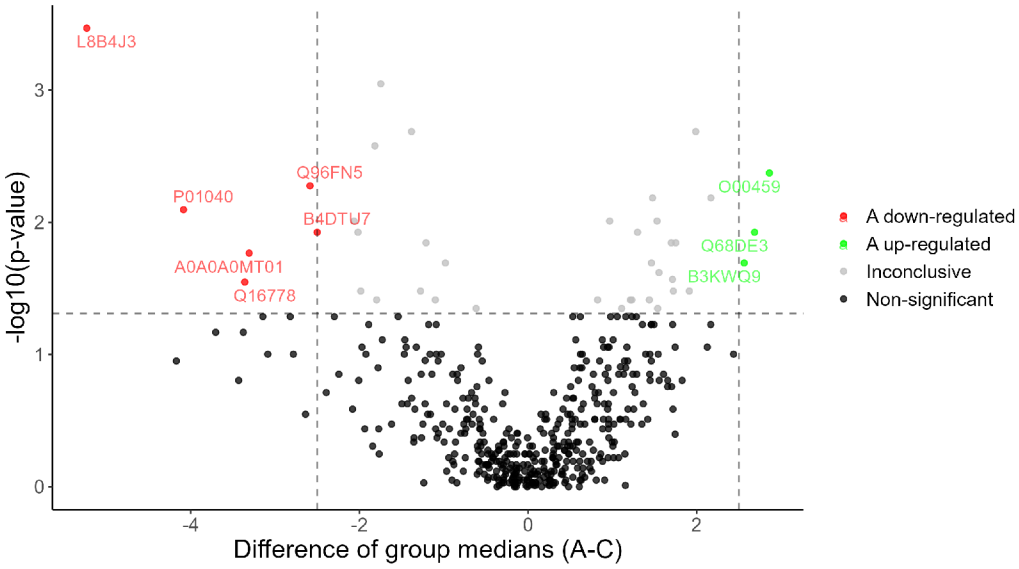
[25]Narozdíl od podílu, kde by tento nulový rozdíl představovala jednička.

[26]Tradiční volbou je v tomto případě logaritmus se základem deset.

celkově jsou tak pro nás zajímavé ty proteiny, které se vyskytují nad touto vodorovnou čárou a zároveň v prostoru napravo od svislé čáry v kladné části osy x či nalevo od svislé čáry v záporné části osy x. Pozorování pod vodorovnou čárou můžeme označit za nevýznamná, pozorování nad ní (avšak mezi oběma svislými čárami) zase za nejednoznačná – p-hodnota testu pro ně sice vyšla pod námi stanovenou hladinou významnosti, avšak rozdíl skupinových mediánů nebyl (dle našich požadavků) dostatečně vychýlen k jedné či druhé skupině. [20][28].

Tím, že p-hodnota ve výsledné tabulce přísluší celkovému testu (nikoliv výsledku mnohonásobného porovnání po ANOVA či Kruskalově–Wallisově testu), je třeba dát pozor na interpretaci výsledků u *volcano plotu* zvolené dvojice skupin – to, že v rámci testu vyšel statisticky významný rozdíl v abundanci daného proteinu mezi skupinami, ještě neznamená, že je tento rozdíl významný pro zobrazovanou dvojici. Proto je v těchto situacích vhodnější použít pro hodnoty osy y p-hodnoty z mnohonásobného porovnání (Tukey či Dunn) přímo pro vykreslovanou dvojici, příp. přidat do tabulky i sloupce s p-hodnotami pro mnohonásobné porovnávání jednotlivých dvojic. Příklad možné vizualizace *volcano plotu* můžete vidět na Obrázku 1.15. Opět je možné zapojit interaktivní prvky (infobox pro každý bod s kódem proteinu, rozdílem skupinových mediánů a p-hodnotou testu) či doplnit kódy proteinů do grafu v těch případech, kdy byl rozdíl mezi skupinami jak statisticky významný, tak dostatečně velký (dle požadavků vědeckého pracovníka).

Tímto jsme tedy připraveni pracovat s proteomickými daty, rozumíme jejich specifikům či nástrahám, zároveň víme, co je s nimi potřeba před analýzou udělat a jakými prostředky toho dosáhnout. Mezi vědci věnujícími se této problematice však nemusí každý z nich ovládat nějaký programovací jazyk a být tak schopen provést všechny zmíněné kroky – tito kolegové jsou



Obrázek 1.15: Volcano plot s výsledky t-testu a adjustovanou p-hodnotou

tak odkázáni na uživatelsky přívětivé nástroje, které jsou schopny výše uvedené kroky provést. Vzhledem ke komplexitě problematiky jde při vytváření takových nástrojů o rozsáhlejší projekty (co do časové náročnosti i množství kódu). Proto si v následující kapitole představíme koncept, který nám vytváření a správu takového projektu usnadní.

Kapitola 2

Shiny aplikace jako balíček v R

Programovací jazyk R disponuje obrovským množstvím tzv. balíčků, které doplňují jeho základní funkce a slouží k nespočtu účelů. Jedním z nich je balíček `shiny` [36] umožňující uživatelům vytvářet webové aplikace. Tato kapitola však nebude (i kvůli zachování přiměřeného rozsahu práce) zaměřená na jazyk R jako takový, obecný postup vytváření balíčku[1] či základy Shiny aplikací[2]. Základní znalost obojího bude dokonce předpokladem (či minimálně doporučením) pro snadné pochopení celé kapitoly. Ta se místo těchto obecných základů bude snažit poskytnout návod, jak tato témata spojit – tedy jak vytvořit Shiny aplikaci ve formě R balíčku.

Možná teď čtenáře napadá, proč celou situaci takhle komplikovat. Shiny aplikace běžně sestává z jednoho až dvou souborů, ve kterých jsou odděleny kódy generující uživatelské rozhraní (UI) a serverová část zastávající veškeré výpočty, tvorbu vizualizací a dynamického obsahu. Proč tedy něco tak „jednoduchého“ zakomponovávat do na první pohled složité struktury R balíčku a znásobit počet vytvořených souborů? Odpověď poskytne první

[1] Více k této problematice najdete např. v diplomové práci [13].
[2] O tvorbě Shiny aplikací se můžete dočíst v publikaci [24].

velká Shiny aplikace, do jejíchž „střev“ nahlédnete. Kdo se totiž rozhodne pro komplexní aplikaci se spoustou funkcionalit, zjistí, že za dobu práce na tomto projektu vyprodukoval klidně i nižší jednotky tisíc řádků kódu. Co asi tak nastane, když se autor projektu rozhodne přidat (či ještě hůř – odebrat) nějakou funkci či změnit cokoliv v tomto provázaném systému? A jak by se v kódu orientoval někdo, kdo přišel k projektu později a má se do vývoje rozpracované aplikace zapojit?

Od určité komplexity aplikace je tak vhodné zvolit jiný, více strukturovaný přístup, který umožňuje sofistikovanější management projektu a nabízí programátorům lepší orientaci v samotném kódu díky modularitě výsledné aplikace. A přesně to bude cílem této kapitoly.

Postupně si rozebereme základní myšlenky výše uvedeného přístupu a to, na co myslet, když podobný projekt připravujeme. Představíme si balíček `golem` [39] sloužící coby užitečný rámec a pomocník při tvorbě robustních Shiny aplikací. Vysvětlíme si pojem *shiny modul*, který je základním stavebním kamenem takových aplikací. Také si ukážeme, na čem se dá vystavět komunikace mezi moduly. Na závěr této části představíme možnosti publikace hotového projektu.

Vycházet u toho budeme z knihy [8], která nabízí unikátní vhled do zmíněného přístupu, přesto občas nejde do takových detailů, které by nováček v problematice potřeboval. Právě to stálo za začleněním tématu do této práce.

2.1 Plánování projektu

Cílem této podkapitoly je upozornit na věci, které by měl mít vývojář aplikace na paměti, než se pustí do psaní kódu. Jde o více či méně obecné poznatky, které možná člověk pracující na IT projektech považuje za ba-

nální, z pohledu statistika jde však o myšlenky rozšiřující pohled na celou problematiku.

Mezi nápadem vytvořit aplikaci pomocí `shiny` balíčku a hotovým produktem je spousta kroků a aspektů, které musí vývojář (či celý vývojářský tým) promyslet. Jedním z aspektů je komplexita výsledné aplikace. Kim ve své knize popisuje komplexní systém jako něco, co už jedinec není schopen vidět jako jeden celek a chápat, jak do sebe jednotlivé části systému zapadají [17]. Nabízelo by se tak mít jako jeden z cílů při prvotním plánování projektu snížení celkové komplexity. Tak přímočaré to ale bohužel není.

Autoři publikace [8] správně poukazují na dva úhly pohledu, kterými můžeme komplexitu nahlížet:

- komplexita z pohledu vývojáře (též implementační komplexita)
- komplexita z pohledu uživatele

Implementační komplexita spočívá v tom, do jaké míry dokáže vývojář uvažovat o aplikaci jako o celku a orientovat se v procesech, které se dějí „uvnitř“. Při vysoké míře tohoto druhu složitosti například nedokážeme při chybě rozpoznat, co ji zapříčinilo (typická je složitá orientace v kódu). Také je těžké určit, jaké reakce vyvolá konkrétní akce [8].

Uživatelská komplexita je reprezentována např. množstvím funkcí, které má uživatel k dispozici, ale také složitým rozpoznáním logického postupu v aplikaci (není zřejmé, jak na sebe jednotlivé kroky navazují a kudy dál). Vysoká míra této komplexity zvyšuje pravděpodobnost, že uživatel udělá v průběhu používání aplikace chybu, zároveň je těžší naučit se aplikaci používat (uživatele může příliš komplexní aplikace od používání dokonce odradit). Jak možná čtenář tuší – tyto dva druhy komplexity jdou proti sobě. Nejde se jich

tedy zbavit minimalizací obou typů. Je to o nalezení vhodné hranice mezi nimi [8].

Příkladem hledání takového balancu může být interaktivita. Obecně platí, že více interaktivity může potenciálně způsobit větší problémy (výpočetní náročnost, nepředvídatelné chování aplikace) než méně interaktivity. Pokud se vývojář rozhodne mít v aplikaci spoustu nástrojů okamžitě reagujících na akci uživatele, zpravidla musí pro pokrytí možných chyb (a snížení uživatelské komplexity) zvýšit implementační komplexitu aplikace. Proto je dobré u zařazování interaktivních prvků promyslet, zda jsou opravdu přínosné [8].

Vývoj aplikace se dá rozdělit na dvě části – v IT komunitě se používají pojmy *frontend* a *backend*, které se do shiny prostředí dají jednoduše převést na UI a serverovou část. Autoři knihy [8] však zmiňují, že není ideální pracovat na obou částech od začátku zároveň, ale je vhodnější jejich vývoj rozdělit. Příprava UI by měla vést hlavně k tomu, že pomůže abstraktní projekt „Aplikace s účelem XY“ zhmotnit v něco, pod čím si všichni účastníci projektu dokážou představit něco konkrétnějšího, snáze jim dojde, jaké prvky a funkce chtějí do aplikace zahrnout. Možná teď čtenáře napadne, jak vytvořit Shiny aplikaci bez serveru. Co všechny ty grafy, tabulky a další výstupy, které se běžně počítají na serverové části? Ano, zcela bez serveru se neobejdeme, ale v této fázi plánování UI (kdy mohou probíhat i časté změny požadavků) by bylo vytváření serverového kódu přímo pro danou aplikaci překážkou. Místo toho se dá využít R balíček `shinipsum` [40]. Ten obsahuje funkce pro vygenerování náhodných grafů, tabulek, textů, výstupů ze statistických modelů atd. Pomocí něj tak můžeme vytvořit funkční prototyp UI bez toho, abychom již v této plánovací fázi strávili čas nad funkcemi serveru, které možná ani nevyužijeme [8].

Podobně můžeme začít z opačného konce a budovat části backendu, které

na aplikaci nejsou závislé. Sem patří specifické funkce a algoritmy, které se doporučují vytvořit zvlášť (a řádně zdokumentovat). Budou k dispozici jednak samostatně (v rámci R balíčku), jednak je můžeme volat v rámci aplikace, což opět zpřehlední celý kód. Oba výše zmíněné přístupy mají za cíl vytvořit pevnější základy projektu a dopředu si co nejkonkrétněji určit, jak bude aplikace vypadat a jaké funkce bude plnit. Poté již může dojít k postupné integraci připravených částí do jednoho funkčního celku a pokračování ve vývoji aplikace [8].

Při plánování jednotlivých funkčních prvků aplikace má vývojář jasnou představu o tom, jak je třeba ze strany uživatele aplikaci používat, aby správně fungovala – podle toho je také následně programována. Určitě je vhodné u prvků, u nichž je zajištění správné podoby na uživateli (typicky formát a podoba datové sady či jiné vstupy), sepsat jasně dané pokyny a dát uživateli možnost si je v aplikaci zobrazit. Tady však narážíme na základní omyl vývojáře. Držme se příkladu s nahráváním dat. Vývojář má aplikaci připravenou tak, že pro její správné fungování potřebuje např. názvy sloupců v určitém tvaru, hodnoty musí být určitého typu (např. pouze čísla či chybějící hodnoty) atd. Proto dá k dispozici interaktivní nápovědu, ve které bude přesný tvar datové sady popsán. Očekává tedy, že uživatel zapne aplikaci, klikne na nápovědu, nastuduje si vše o podobě datové sady, upraví svoje data a následně je nahraje do aplikace. Takto se však většina uživatelů nechová a jejich používání aplikace je mnohem nahodilejší, než by autora programu vůbec napadlo. A pokud se stane, že uživatel po spuštění pouze nahraje nějakou datovou sadu v nevyhovující formě, je vhodné zahrnout kontrolní mechanismy a následné informativní upozornění, co udělal uživatel špatně (nepřipravenost aplikace na její chybné užívání a vyskakování červených hlášení *„Error: . . . “* může uživatele dokonce odradit). Tomuto způsobu psaní

kódu se v angličtině říká *defensive programming* – snaží se počítat s nečekaným chováním uživatele, předcházet chybám a kontrolovat správnost vstupů [8]. U situací, na něž se můžeme připravit, existuje také možnost sdělit uživateli vizuálně příjemnou a informativní formou, co dělá špatně. K tomu slouží např. R balíčky `shinyFeedback` [43] pro kontrolu vstupů a `shinyalert` [34] k tvorbě vyskakovacích upozornění. Pokud přesto dojde k takové chybě, že aplikace selže, uživatel by se to měl taktéž dozvědět přívětivým a informativním způsobem. K tomu slouží např. R balíček `sever` [37].

Již před začátkem samotného programování je dobré mít základní představu o tom, co vše bude aplikace umět. I s detailním plánem se však mnohdy stane, že již zabudované části se vývojáři rozhodnou odstranit, příp. zjistí, že jsou ve „slepé uličce“ a musí se vydat jiným směrem. Nejen proto je u větších projektů nezbytné využívat tzv. *version control*. Jde o nástroj umožňující mít na jednom místě[3] pohromadě nejen všechen kód v jeho aktuální podobě, ale také všechny předchozí verze kódu (ideálně s popisem provedených změn). Tento nástroj tak umožní např. kdykoliv se „vrátit v čase“ ze slepé uličky na místo, z kterého se znovu můžeme vydat jiným směrem (s vědomím, že aplikace v tomto bodě bez problémů fungovala). Version control také umožňuje paralelní spolupráci na totožném kódu ze strany více osob a následnou bezproblémovou implementaci těchto částí od více autorů dohromady. Pravděpodobně nejznámějším nástrojem tohoto typu je Git (a platforma GitHub sloužící ke správě projektů používajících Git)[4]. Již při plánování projektu je vhodné naučit se s některým z těchto nástrojů pracovat a využívat ho od samotného začátku [8].

[3] Přístupný jednotlivci–autorovi, celému týmu autorů či veřejně, a to online, nikoliv někde v úložišti počítače.

[4] Více se o Gitu a GitHubu můžete dočíst např. zde: https://docs.github.com/en/get-started

2.2 Proč R balíček?

Z uvedených nároků na komplexní aplikaci (větší přehlednost kódu, rozdělení aplikace na menší části a lepší možnosti managementu projektu) přímo neplyne, že tou správnou cestou je vytváření Shiny aplikace jako balíčku v R. Proč je tedy tento přístup vhodný? Neboť nám umožní zahrnout do jedné struktury potřebné náležitosti, bez kterých se takový projekt neobejde, případně využije nástroje, které nám celý proces usnadní a pro R balíčky již existují. V publikaci [8] podtrhují tyto:

- Metadata – základní informace o aplikaci jako název, číslo verze, k čemu slouží, jakož i kontakt na vývojáře pro případ, že má uživatel s aplikací problém.
  - to vše zahrnuje u R balíčku soubor `DESCRIPTION`
- Dependencies – anglický výraz zahrnující informace o tom, jaké funkce z jiných R balíčků aplikace využívá (je na nich doslova „závislá“).
  - tyto informace jsou elegantně zahrnuty v souborech `DESCRIPTION` a `NAMESPACE` (součást každého R balíčku)
- Více menších souborů s kódem na jednom místě.
  - struktura R balíčku přímo určuje místo pro uložení všech R skriptů (podsložka `R`)
- Dokumentace – soubor textů představující v podstatě „návod“, jak aplikaci používat a co všechno umí. R balíčky pracují s třemi typy dokumentace:

  - Soubor `README` – obsahuje pouze základní informace (jak balíček s aplikací nainstalovat a co umí). Často ve formátu *.md* (markdown), dokument se pak snadno „sestaví“ právě např. na úvodní stránce balíčku na GitHubu.
  - `Vignettes` – podrobnější dokumentace s detailním návodem, jak aplikaci používat a co jednotlivé nástroje umí.
  - Dokumentace funkcí – nápověda k funkcím (ale také k jednotlivým modulům – vizte část 2.3), kterou najdete přímo v RStudiu na záložce *Help*.

- Testování – možnost aplikaci testovat pomocí nástrojů vyvinutých pro R balíčky. V této práci pro ně nezbyl prostor, v knize [8] se jim však věnuje celá kapitola.

- Publikování – v momentě, kdy je aplikace ve formě R balíčku, je mnohem jednodušší ji stáhnout a nainstalovat do počítače, neboť ji lze veřejně publikovat např. na GitHubu (kde si uživatel může zvolit, zda si stáhne celou kopii zahrnující všechny soubory, nebo si ji jen nainstaluje dle návodu v `README`). To platí pro uživatele se znalostí R. O možnostech pro ty, kteří tento jazyk neovládají, více pojednává část 2.6.

2.3 Shiny moduly

Komplexní aplikace mohou způsobit řadu problémů:

- Nespočet objektů, které potřebují unikátní ID (všechna tlačítka, vstupy atd.).

- Podobné úseky kódu, které se na několika místech opakují.

- Celková nepřehlednost (vše v jednom či dvou rozsáhlých souborech).

To vše a mnohé další vyřeší autorům Shiny aplikací osvojení si přístupu tzv. *shiny modulů*.

2.3.1 Základní idea a ukázkový příklad

Hlavní myšlenkou je rozdělení celé aplikace na menší části tak, aby se funkčně či prostorově (co do umístění ve výsledné aplikaci) podobné úseky kódu utvořily obecně a daly se pak klidně i několikrát použít kdekoliv v aplikaci. Parametrizace modulů pak umožní nastavit specifické požadavky pro daný úsek aplikace. V podstatě tak tvoříme několik menších aplikací, jakýchsi stavebních kostek s konkrétním účelem, z kterých se výsledná aplikace složí. Celý konstrukt má pak strukturu pyramidy. Na vrcholu jsou tradiční UI a server, které se však skládají z jednotlivých modulů – ty mohou mít libovolný počet úrovní (tedy i samotné moduly mohou být složeny z jiných modulů). Tento přístup nemusí být nutně spojen s Shiny aplikací ve formě R balíčku, funguje i při „konvenční“ tvorbě tohoto typu programů [8].

Efektivita tohoto přístupu spočívá i ve vyřešení problému konfliktů stejných ID pro sice stejně fungující objekty (např. akční tlačítko generující tabulku s daty), avšak na jiných místech aplikace (např. 3 různá tlačítka pro 3 různé tabulky). Každý objekt musí mít totiž unikátní ID. A řešit to ve větší aplikaci tak, že vytvoříme deset akčních tlačítek s ID `run1` až `run10` pro vygenerování tabulek s ID `table1` až `table10` není optimální. Ukažme si využití modulů na jednoduchém příkladu.

Chceme vytvořit primitivní Shiny aplikaci, která bude mít 3 akční tlačítka zobrazující 3 různé tabulky. Rozložení těchto objektů bude takové, že příslušná dvojice tlačítko–tabulka bude vždy na stejném řádku. „Tradičním“ přístupem bychom mohli postupovat takto:

```
library(shiny)

#User interface:
ui = fluidPage(
       fluidRow(
         column(width = 2,
                actionButton(inputId = "run1",
                             label = "Show me Table 1")),
         column(width = 10,
                tableOutput("table1"))
       ),
       fluidRow(
         column(width = 2,
                actionButton(inputId = "run2",
                             label = "Show me Table 2")),
          column(width = 10,
                tableOutput("table2"))
       ),
       fluidRow(
         column(width = 2,
                actionButton(inputId = "run3",
                             label = "Show me Table 3")),
         column(width = 10,
                tableOutput("table3"))
       )
     )

#Server:
server = function(input,output,session){
         #'Zobraz tabulku při stisknutí akčního tlačítka' 3krát:
         observeEvent(input$run1,{
           output$table1 = renderTable(table(iris$Species))
         })
         observeEvent(input$run2,{
           output$table2 = renderTable(table(mtcars$cyl))
         })
         observeEvent(input$run3,{
           output$table3 = renderTable(table(mtcars$am))
         })
}

#Spuštění aplikace:
shinyApp(ui,server)
```

I u takto jednoduchého příkladu vidíme, jak jsme nuceni neustále opakovat to samé, pouze s drobnými změnami parametrů. Ukažme si nyní, jak by

se totožná aplikace dala vytvořit pomocí shiny modulů:

```
#UI část modulu: funkce s parametry 'id' (id modulu, povinný parametr
#pro všechny moduly) a 'tab_no' - číslo tabulky (použijeme jej
#v označení akčního tlačítka):
table_ui = function(id,tab_no){
#'Namespacing' funkce - automaticky přidá před zvolené ID
#předponu specifickou pro daný modul (zjednodušeně: posloupnost ID
#zapojujících se modulů dle hierarchie celé aplikace - to však
#nemusíme detailně zkoumat, tato funkce to za nás vyřeší):
  ns=NS(id)
#Obsah UI zabalený do tagList() funkce:
  tagList(
    column(width = 2,
           actionButton(inputId = ns("run"),
#ID objektů musí být v UI zabalena do ns()
                        label = paste("Show me Table",tab_no))),
    column(width = 10,
           tableOutput(outputId = ns("table")))
  )
}
#Serverová část modulu (opět s parametry 'id' a 'tab_no').
#'id' opět povinný parametr, a co je hlavní, při vložení modulu do vyšší
#struktury (přímo UI či server aplikace, příp. modul výše v hierarchii)
#pomocí funkcí table_ui() a table_server() musí mít příslušná
#dvojice STEJNÉ 'id')
table_server = function(id,tab_no){
  moduleServer(id,
    function(input,output,session){
#Zde je kód víceméně stejný jako v běžném serveru aplikce,
#jedinou změnou je příkaz:
      ns = session$ns
#zajišťuje propojení namespacingu mezi serverem a ui v rámci modulu
#jinde v serveru už není třeba funkci 'ns' řešit
      observeEvent(input$run,{
        output$table = renderTable(table(
          switch(tab_no,
                 iris$Species,
                 mtcars$cyl,
                 mtcars$am)))
       })
  })
}
#Následně z připravených modulů sestavíme UI a server aplikace:
app_ui = fluidPage(
           fluidRow(
             table_ui(id = "table1",tab_no = 1)),
           fluidRow(
```

```
            table_ui(id = "table2",tab_no = 2)),
          fluidRow(
            table_ui(id = "table3",tab_no = 3))
        )
app_server = function(input,output,session){
  table_server(id = "table1",tab_no=1)
  table_server(id = "table2",tab_no=2)
  table_server(id = "table3",tab_no=3)
}
#Spuštění aplikace:
shinyApp(app_ui,app_server)
```

Ačkoliv výše uvedený kód vypadá obsáhle, působí to tak spíše opticky kvůli komentářům. Moduly nám při vytváření aplikací ulehčí spoustu práce a zejména zpřehlední strukturu aplikace. Úspora času je ještě významnější při konstrukci složitějších modulů, které zároveň v aplikaci využíváme mnohokrát (např. modul pro vizualizaci dat, který datovou sadu přichystá do požadovaného tvaru a vykreslí příslušný graf). S pomocí balíčku `golem` (více v podkapitole 2.4) navíc moduly netvoříme na „nepopsaný skript“, ale využíváme přichystanou kostru.

Na tomto místě uvedeme i užitečnou technickou poznámku k víceúrovňovým modulům. V případě, že UI modulu vkládáte nikoliv přímo do UI aplikace, ale do modulu, který je v hierarchii výše, je nutné zabalit jeho ID opět do funkce `ns()`. Pokud bychom se inspirovali u výše zmíněného příkladu a mezi `table` modulem a hlavním UI a serverem aplikace by byla ještě další modulová úroveň, vkládali bychom `table_UI` do „nadmodulu“ takto: `table_ui(id = ns("table1"),tab_no = 1))`. Jak bylo zmíněno v komentářích kódu, v serveru není třeba po spárování namespacingu pomocí `ns = session$ns` tuto funkci řešit, serverový protějšek k `table_UI` bychom tak do serveru nadřazeného modulu zahrnuli úplně stejně jako do hlavního serveru: `table_server(id = "table1",tab_no = 1)`.

Nejen při zavádění nových funkcionalit do aplikace, avšak i při jakýchkoliv

změnách jsou moduly velmi užitečné. Požadovanou změnu (nová barva grafu, jiný popisek tlačítka) stačí provést na jednom místě a propíše se do všech míst, kde jsme daný modul použili.

Jak lze z uvedeného příkladu vidět, u modulu tvoříme zvlášť UI a zvlášť server podobně jako u samotné aplikace. V příkladu to pro `table` modul byly části `table_ui` a `table_server`. Tím se vracíme k výše uvedenému, tedy že s tímto přístupem vlastně tvoříme několik menších aplikací, kdy každá zastává část funkcí „velké“ Shiny aplikace, která je z nich pak složena [8].

2.3.2 Komunikace mezi moduly

Jednou z nejproblematičtějších částí tohoto přístupu je předávání informací z jednoho modulu do druhého. Všechny akce, vstupy a výstupy se totiž při používání standardního kódu pro tvorbu Shiny aplikací týkají pouze daného modulu, kde vznikly. Mějme např. aplikaci rozdělenou na postranní panel s možností úprav grafických a jiných parametrů a různými tlačítky a „tělo“ aplikace s výstupy ve formě grafů či tabulek. Pokud máme postranní panel jako jeden modul a tělo aplikace jako druhý, musíme mezi nimi vystavět jistý komunikační „most“, aby se např. při stisknutí tlačítka *Vykresli graf!* na postranním panelu zobrazil požadovaný obrázek v těle aplikace. Možností, jak komunikaci mezi moduly vystavět, je několik. V této části rozebereme pouze jednu z nich – v originále pojmenovanou **„stratégie du petit r“** (tedy strategie malého r) [8].

Hlavní myšlenkou je vytvoření seznamu reaktivních hodnot pomocí funkce `reactiveValues()` v souboru se serverem aplikace (tedy na té nejvyšší úrovni) – tento seznam se pojmenovává právě dle názvu strategie jako `r`. Do serverové části každého modulu se pak toto `r` přidá na pozici parametru funkce, ve které je veškerý kód serverové části modulu. Tím se v rámci modulu zpří-

stupní celý reaktivní seznam, do nějž lze zároveň cokoliv ukládat. Prvky z tohoto seznamu (ať už jde o datové sady, tabulky, grafy atd.) jsou pak dostupné ve všech modulech, které obsahují `r` v parametrech funkce [8].

Ukažme si použití tohoto přístupu na jednoduchém příkladu. Mějme Shiny aplikaci složenou ze dvou modulů – `sidebar` (tedy postranní panel) a `body` (tělo aplikace). Budeme používat v R zabudovanou datovou sadu `iris`. Na postranním panelu bude jeden číselný vstup určující počet řádků z datové sady, které následně aplikace náhodně zvolí. Dále tam budou 2 akční tlačítka – jedno pro náhodný výběr z datové sady `iris`, druhé pro zobrazení/aktualizaci grafu[5]. Po stisknutí tlačítka *Sample n observations from iris data set* se do `r` uloží náhodně zvolených n řádků z datové sady `iris` (kde n je z číselného vstupu). Při stisknutí druhého akčního tlačítka se v rámci tohoto modulu v `r` vytvoří číselná hodnota, která se zvyšuje s každým dalším stisknutím tohoto tlačítka. Nejde vůbec o konkrétní hodnotu, avšak o provedenou změnu, akci, která se dá následně pozorovat v dalším modulu.

```
library(shiny)
library(shinydashboard) #balíček pro vytváření aplikací ve formě dashboardů
#(se strukturou záhlaví - postranní panel - tělo aplikace)
library(DT) #balíček pro zobrazování dat v pěkných tabulkách s množstvím
            #funkcí
mod_sidebar_ui = function(id){
  ns <- NS(id)
  tagList(
    numericInput(inputId = ns("nrows"),label = "Sample size (n)",
                 min = 1,max = 150,value = 50,step = 1),
#iris má 150 řádků, proto max = 150
    actionButton(inputId = ns("use_data"),
                 label = "Sample n observations from iris data set"),
    actionButton(inputId = ns("render"),
                 label = "Render scatter plot")
  )
}
```

[5] Aplikaci se stejnou funkcí by šlo vytvořit i pouze s jedním akčním tlačítkem (graf by se zobrazil automaticky s novým náhodným výběrem), avšak zde jsou ukázány dva způsoby přenášení impulzů mezi moduly – každé tlačítko tak na pozadí funguje trochu jinak.

```
#sidebar Server Functions
#' @param r a 'reactiveValues()' list. It stores the sample data set and
#' the response to pressing the 'render' action button.
#'
#' @rdname mod_sidebar
#' @export
mod_sidebar_server <- function(id,r){
  moduleServer(id,
    function(input, output, session){
      ns = session$ns

      observeEvent(input$use_data,{
        r$data=iris[sample(x = 1:150, size = input$nrows),]
      })
      observeEvent(input$render,{
        r$render_button=ifelse(is.null(r$render_button),
                               1,r$render_button+1)
      })
        })
}
```

Část kódu z řádků začínajících `#'` slouží k následnému automatickému vytvoření dokumentace k modulu, kde jsou mj. popsány parametry modulu. Vzhledem k univerzálnosti `r` je dobré právě na tomto místě podrobně popsat, jaká je role `r` v konkrétním modulu a jaké hodnoty se do něj ukládají (nebo naopak se z něj využívají). Více si dokumentaci rozebereme až v části 2.5.

Podívejme se na druhý modul zahrnující tělo aplikace. Uživatelské rozhraní je rozděleno na dva řádky – v obou je box s obsahem. V tom prvním je tabulka s daty, ve druhém bodový graf závislosti délky a šířky kališních lístků (založen na datech v tabulce). Na serveru pak probíhá sestavení těchto dvou objektů s využitím vstupů z modulu pro postranní panel. Detailnější komentáře můžete nalézt přímo v kódu u příslušných objektů.

```
mod_body_ui = function(id){
  ns = NS(id)
  tagList(
    fluidPage(
      fluidRow(
        box(width=8,title = "Data table",status = "primary",
```

```
            DTOutput(ns("table")))
      ),
      fluidRow(
        box(width=12,title = "Scatter plot",status = "primary",
            plotOutput(ns("plot")))
      )
    )
  )
}
# body Server Functions
#' @param r a 'reactiveValues()' list. It uses r$data from the sidebar
#' module to render a table and a plot. Plot rendering is triggered
#' by the change of r$render_button.
#' @rdname mod_body
#' @export
mod_body_server = function(id,r){
  moduleServer(id,
    function(input, output, session){
      ns = session$ns
#####Table ----
      output$table=renderDT({
        req(!is.null(r$data)) #Sestavení tabulky je tímto pozdrženo do
                              #momentu, kdy už má co vykreslit
        datatable(r$data, rownames=F,selection="none")
      })
#####Plot ----
#Po stisknutí tlačítka 'render' na postranním panelu se přes 'r'
#začne sestavovat datová sada pro vykreslení grafu. Zde jde přímo
#o r$data, ale v jiných případech se dá na tomto místě originální
#datová sada transformovat pro potřeby daného grafu.
      dTOplot=eventReactive(r$render_button,{
        req(!is.null(r$data))
        r$data
      })
#Graf se sestaví při prvním vytvoření reaktivní proměnné dTOplot()
#a následně reaguje na její změny (spuštěné akčním tlačítkem).
      output$plot=renderPlot({
        req(!is.null(dTOplot()))
        plot(dTOplot()$Sepal.Length,dTOplot()$Sepal.Width)
      })
  })
}
```

Nyní tedy zbývá vytvořit „hlavní“ server a UI a zapojit do nich vytvořené moduly. V serveru je klíčové mít zadefinované malé `r` coby reaktivní list. Zároveň do něj vložíme libovolnou první položku (na obsahu opět nezáleží,

dokonce ji ani nikde jinde nemusíme použít). Při zapojování serverové části modulů pak musíme zahrnout parametr `r` a vložit do něj vytvořený reaktivní list. Tím jej modulům zpřístupníme [6].

```
#UI:
app_ui = dashboardPage(
  dashboardHeader(title = "Example app"),
  dashboardSidebar(width = '25%',mod_sidebar_ui("sidebar_1")),
  dashboardBody(mod_body_ui("body_1"))
)
#Server:
app_server = function(input, output, session) {
  r=reactiveValues( #zadefinování reaktivního objektu r
      d=c() #první (libovolná) položka seznamu - v tomto momentě
            #je jedno, co to bude
  )
  mod_sidebar_server("sidebar_1",r = r)
  #do parametru 'r' posouváme "naše" r
  mod_body_server("body_1",r = r)
}
#Spuštění aplikace:
shinyApp(app_ui,app_server)
```

Na příkladu tedy můžete vidět hned dvě cesty, jak pomocí `r` předávat informace z jednoho modulu do druhého. V prvním případě (tabulka) využíváme přímo uloženou položku reaktivního listu (opět zabalenou v nějakém reaktivním prostředí typu `render`, `observe` atd.). V druhém případě (graf) si pomáháme „umělou“ položkou a využíváme typicky prostředí reagující na události (`eventReactive`, `observeEvent`). To je vhodné právě pro reakci na „spouštěče“ z jiných modulů.

Specifickým problémem v komunikaci mezi moduly jsou reaktivní prvky či podmíněné panely závislé na aktivní záložce aplikace (při používání rozložení `tabsetPanel()`). Při zadefinování tohoto rozložení v rámci modulu se musí jeho ID taktéž zabalit do funkce `ns()`, např. `tabsetPanel(id =`

[6] Nesmíme však zapomenout tento parametr zahrnout v samotné funkci, která serverovou část modulu vytváří – viz výše.

`ns("tabs"), ...`). Při použití podmíněných panelů[7] v rámci jiného modulu (než toho s prostředím `tabsetPanel()`) se však objeví dva problémy:

- potřeba přenést ID z jednoho namespacingu do jiného – tento problém jsme již řešili v příkladech výše
- ID záložky se ve funkci `conditionalPanel(...)` zadává ve formě textového řetězce

Řešení tu však je. Mějme v rámci UI jednoho modulu zavedené rozložení `tabsetPanel(id = ns("tabs"), ...)` a v něm zadefinovanou záložku `tabPanel("Exploratory data analysis",value = "eda", ...)`. Na serveru tohoto modulu můžeme zavést nový objekt do malého `r`, který se bude reaktivně měnit v závislosti na aktuálně navštěvované záložce (resp. bude obsahovat právě ID aktivní záložky) a zároveň tuto informaci „zpřístupní“ ostatním modulům (v případě používání výše rozebrané „stratégie du petit r“):

```
observeEvent(input$tabs,{
  r$tabset_value=input$tabs
})
```

V serverové části modulu, který obsahuje podmíněné panely, pak z této reaktivní proměnné vytvoříme výstupní objekt ve formě textového řetězce spolu s úpravou nastavení tak, aby byl neustále k dispozici:

```
output$tabset_value=renderText(r$tabset_value)
outputOptions(output, 'tabset_value', suspendWhenHidden = FALSE)
```

V UI tohoto modulu pak můžeme podmíněný panel zavést takto:

[7] Typicky měnící se nabídka menu – každá záložka má v tomto případě jiné.

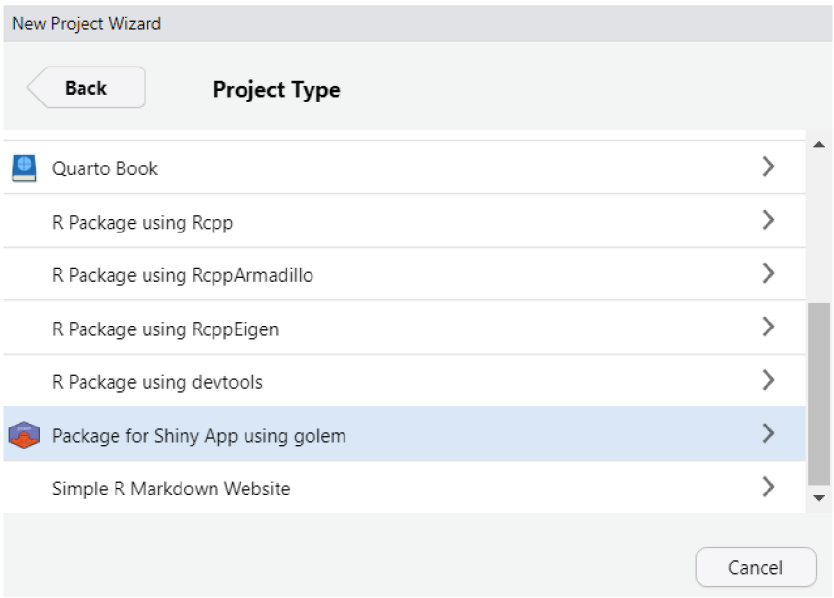
```
conditionalPanel(condition = "output.tabset_value =='eda'",ns=ns,
#...
#content of this conditionalPanel()
)
```

Tímto přístupem lze tedy reaktivitu v závislosti na aktivní záložce používat i napříč moduly.

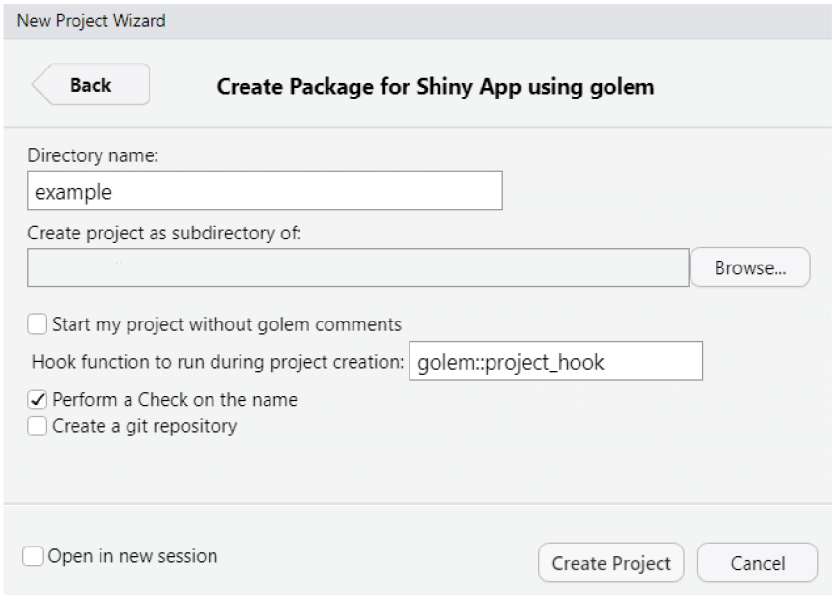
2.4 Golem

V předchozích částech této kapitoly byly uvedeny některé myšlenky důležité při vytváření Shiny aplikace jako R balíčku a také základy shiny modulů. To samo však jako návod pro někoho, kdo by plánoval se do takového projektu pustit, nestačí. Naštěstí existuje R balíček `golem` [39], který je skvělým průvodcem při vytváření Shiny aplikace popsaným způsobem, jejím vývoji a produkci. I nezkušenému uživateli pomůže řídit svůj projekt tak, aby měl již od prvních řádků kódu přehlednou strukturu. Zároveň nabízí množství „vývojářských funkcí“ užitečných při práci na aplikaci – ať už jde o pomoc s dokumentací, vytváření kostry modulů či kontrolu kódu. Postupně si projdeme základní aspekty tohoto balíčku, které zájemcům postačí jako dobrý start a pro orientaci v problematice [8].

Nejprve je třeba balíček `golem` do R nainstalovat – to můžete udělat např. pomocí příkazu `install.packages("golem")`. Shiny aplikaci ve formě balíčku pak snadno vytvoříte přes tlačítka `File -> New Project...`, zbylé jednoduché kroky jsou znázorněny na Obrázcích 2.1 a 2.2. Stále je to R projekt, proto se ujistěte, že při práci na balíčku máte v RStudiu vždy **aktivní** právě tento projekt. Strukturu vytvořeného balíčku na úložišti můžete vidět na Obrázku 2.3.

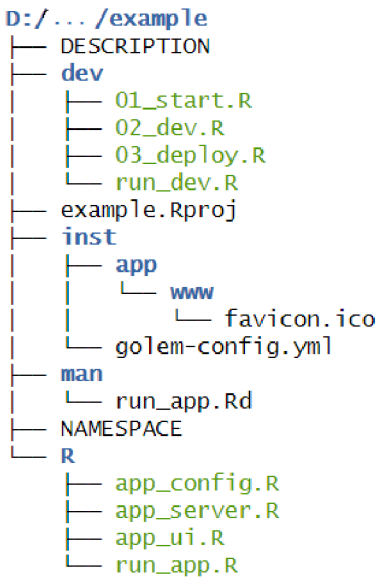


Obrázek 2.1: Vytvoření nového `golem` projektu



Obrázek 2.2: Vytvoření balíčku pro Shiny aplikaci pomocí `golem`

První kroky učiníte v souboru `01_start.R` ze složky *dev*, která obsahuje ony „vývojářské“ skripty pro správu projektu. Začít můžete např. vyplněním údajů o balíčku, které se po spuštění připraveného kódu propíší do souboru



Obrázek 2.3: Struktura R balíčku po jeho vytvoření pomocí `golem`

`DESCRIPTION` (vizte podkapitolu 2.2). Na Obrázku 2.4 je tato část zmíněného skriptu ukázána – podobným stylem jsou připraveny i ostatní funkce – přehledně, s komentáři a snahou o co největší pomoc i nezkušeným uživatelům [8]. Dále umožní uživateli:

- propsat údaje do nastavení `golem` balíčku
  - k tomu slouží funkce `golem::set_golem_options()` – doporučuje se spustit ji ihned po vyplnění údajů o balíčku a spuštění `golem::fill_desc()` (opět vizte Obrázek 2.4)
- nainstalovat další potřebné R balíčky (které jsou dále využívány)
- nastavit licenci balíčku[8]
- připravit soubor `README`

[8] Srovnání jednotlivých softwarový licencí naleznete např. na webové stránce https://choosealicense.com/licenses/.

- využívat GitHub
- připravit testovací infrastrukturu
- vytvořit soubory `golem_utils_ui.R` a `golem_utils_server.R` s pomocnými funkcemi

Vše je zpracované tak, že uživatel nepotřebuje nutně vidět do útrob těchto funkcí (pokud sám nechce), celé je to velice intuitivní a umožní to hladkou přípravu infrastruktury balíčku.

```
## Fill the DESCRIPTION ----
## Add meta data about your application
##
## /!\ Note: if you want to change the name of your app during development,
## either re-run this function, call golem::set_golem_name(), or don't forget
## to change the name in the app_sys() function in app_config.R /!\
##
golem::fill_desc(
  pkg_name = "example", # The Name of the package containing the App
  pkg_title = "PKG_TITLE", # The Title of the package containing the App
  pkg_description = "PKG_DESC.", # The Description of the package containing the App
  author_first_name = "AUTHOR_FIRST", # Your First Name
  author_last_name = "AUTHOR_LAST", # Your Last Name
  author_email = "AUTHOR@MAIL.COM", # Your Email
  repo_url = NULL, # The URL of the GitHub Repo (optional),
  pkg_version = "0.0.0.9000" # The Version of the package containing the App
)
```

Obrázek 2.4: Ukázka skriptu `01_start.R`

Po tomto „nastartování“ projektu se můžeme přesunout k souborům `02_dev.R` a `run_dev.R`, které jsou užitečným pomocníkem při celém vývoji aplikace. Uveďme alespoň některé funkce, které se opravdu hodí využívat, přičemž většina je již součástí těchto R skriptů, avšak některé je vhodné si do nich doplnit (a mít je tak po ruce).

Pro udržení aktuálnosti souborů `DESCRIPTION` a `NAMESPACE` v průběhu práce na balíčku (při kterém často používáme další funkce z jiných R balíčků) je vhodné pravidelně spouštět funkce `attachment::att_amend_desc()` [44] a `golem::document_and_reload()`. Ta první doplní soubor `DESCRIPTION` na

základě rozboru kódů ze složky *R* (vizte dále) o používané balíčky. Alternativou je funkce `use_package("pkg.you.want.to.add")` z balíčku `usethis` [45], která se taktéž nachází v souboru `02_dev.R`. Jde o manuální cestu, avšak vývojář má nad procesem větší kontrolu a přehled o tom, které balíčky přidával (příp. může za funkci přidat libovolný komentář). Pokud jsou potřeba balíčky, které nejsou umístěné v centrálním úložišti CRAN (a autor balíčku je při vývoji získal např. z GitHubu či BioConductoru), je třeba je do souboru `DESCRIPTION` propsat pomocí jiné funkce z balíčku `usethis`, a to `use_dev_package()`. S balíčky z GitHubu si bez problémů poradí a správně propíše i cestu k nim do sekce `Remotes` na konci souboru `DESCRIPTION`, avšak u balíčků z BioConductoru se autorovi textu osvědčila spíše alternativní cesta – propsat daný balíček do `DESCRIPTION` standardně a přidat do tohoto souboru také řádek s textem `biocViews:`. Tento trik pak zajistí hladké fungování balíčku, který je závislý na jiných balíčcích z BioConductoru. Zmíněnou úpravu lze vidět na Obrázku 2.5 (zvýrazněný text).

```
Depends:
    R (>= 2.10)
biocViews:
Imports:
    amap,
    colourpicker,
    ComplexHeatmap (>= 2.15.4),
    config (>= 0.3.1),
    data.table,
```

Obrázek 2.5: Úprava souboru `DESCRIPTION` při využívání balíčků z BioConductoru

S importem používaných funkcí z těchto balíčků souvisí funkce `document_and_reload()`, která automaticky sestaví dokumentaci balíčku na základě speciální části R souborů s kódy, obnoví soubor `NAMESPACE` a zároveň

„aktivuje“[9] balíček – pomocí příkazu `run_app()`[10] je pak možné současnou verzi aplikace spustit a vyzkoušet ji. Problematika importu funkcí z jiných R balíčků bude více rozebrána v části 2.5.

Jak již bylo zmíněno, v tomto přístupu se počítá s využíváním shiny modulů. Soubor s kostrou shiny modulu můžete vytvořit pomocí funkce `golem::add_module(name = "example", with_test = TRUE)` s argumenty určující jméno modulu a to, zda se má vytvořit i kostra souboru pro testování modulu. Kostru modulu můžete vidět na Obrázku 2.6.

```
#' example UI Function
#'
#' @description A shiny Module.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#'
#' @noRd
#'
#' @importFrom shiny NS tagList
mod_example_ui <- function(id){
  ns <- NS(id)
  tagList(

  )
}

#' example Server Functions
#'
#' @noRd
mod_example_server <- function(id){
  moduleServer( id, function(input, output, session){
    ns <- session$ns

  })
}

## To be copied in the UI
# mod_example_ui("example_1")

## To be copied in the server
# mod_example_server("example_1")
```

Obrázek 2.6: Kostra modulu vytvořená pomocí balíčku `golem`

Pomocí funkcí `golem::add_utils()` a `golem::add_fct()` zase vytvoříte

[9] Zde ekvivalent „zavolání“ již hotového a nainstalovaného R balíčku pomocí funkce `library()`.

[10] Automaticky vytvořená funkce v rámci libovolného `golem` balíčku.

soubory, do kterých můžete definovat drobné pomocné funkce (též utility) či větší funkce, které v kódu Shiny aplikace použijete. Hlavním benefitem je větší přehlednost kódu. Tyto soubory se (stejně tak jako moduly) vytváří v podadresáři *R*, který mj. obsahuje i hlavní server a UI aplikace, resp. jejich kostru nachystanou k použití. Je důležité zmínit, že tento podadresář **nesmí** obsahovat žádnou další podsložku [8].

Pokud je potřeba do aplikace zabudovat interní datovou sadu, ve skriptu `02_dev.R` je třeba najít funkci `usethis::use_data_raw()`, pomocí které se automaticky vytvoří v adresáři balíčku nová podsložka *data-raw* a v ní soubor, ve kterém si datovou sadu připravíte (tj. můžete zde nahrát[11] data, libovolně je upravit) a následně spustíte zde nachystanou funkci `usethis::use_data(dataset_name)`, která v adresáři balíčku vytvoří novou podsložku *data* a do ní uloží připravenou datovou sadu ve formátu *.rda*. Takto připravená data jsou pak pro celou aplikaci dostupná a v kódu pro server či UI už se nemusí zavádět znovu.

Do skriptu `02_dev.R` se hodí přidat funkci `rcmdcheck()` z balíčku `rcmdcheck` [38]. Jejím spuštěním se provede kontrola celého balíčku pokrývající různé problémy. Jsou u funkcí zdokumentovány všechny použité argumenty? Fungují vytvořené testy? Jsou v `NAMESPACE` zaneseny všechny používané funkce? Pokud je cokoliv v balíčku v nepořádku, tento příkaz to vývojáři řekne.

Soubor `03_deploy.R` souvisí s publikováním aplikace a bude rozebrán samostatně v části 2.6. Přesuňme se nyní krátce do umístění *inst/app/www* – do této podsložky se doporučuje ukládat jakékoliv nové externí soubory (obrázky, markdown dokumenty, CSS, JS). Souvisí to s dvěma funkcemi:

[11] Pro větší flexibilitu je při využívání GitHubu (což se velmi doporučuje) vhodné případný soubor s daty nahrát na GitHub a v tomto přípravném skriptu tak nahrávat data z pevného odkazu, nikoliv z lokálního úložiště.

- `golem_add_external_resources()` ze souboru `app_ui.R`
  - ta je nachystaná na přejímání externích zdrojů (jako např. vlastní CSS soubory) právě ze zmíněného umístění
  - zároveň se do ní mohou zdroje přidávat i pomocí funkcí (např. `shinyjs::useShinyjs()` [33] pro využití reaktivně se objevujících a mizejících prvků)
  - souvisí s prvky dostupnými při běhu aplikace (nepočítají se na serveru, využívá je prohlížeč aplikace)
- `app_sys()` ze souboru `app_config.R`
  - ta slouží k přistupování k souborům během generování aplikace (defaultně hledá soubor ve složce *inst*)
  - pracuje na serverové části
  - často v rámci „generující“ funkce, např. `includeMarkdown(app_sys("app/www/help.Rmd"))` (zde ji tedy v rámci *inst* složky nasměrujeme do podsložky *www* a vygenerujeme dokument typu markdown)

Všechny zmíněné typy souborů tak můžeme mít přehledně na jednom místě a pomocí těchto funkcí je v aplikaci využívat [8].

Posledním základním podadresářem ze struktury na Obrázku 2.3 je složka *man* – do té se však pouze automaticky vytvářejí soubory s dokumentací balíčku (k jednotlivým funkcím, modulům apod.), bude o ní tedy řeč až v části 2.5.

2.5 Dokumentace

Dokumentace balíčku je stejně nezbytnou součástí, jako funkce (či aplikace), které jej tvoří. Na první pohled jde o něco, čím umožňujeme (a v případě kvalitního provedení velmi usnadňujeme) práci s balíčkem uživatelům, ale vývojářům to přidělává práci. To však není tak docela pravda. Vývoj balíčku většinou trvá delší čas a právě vývojáři jsou ti, kteří ocení, když budou mít náhled do myšlenek a záměrů svého minulého já [8].

Dokumentování by se dalo rozdělit na dva typy:

- tzv. *vignettes* – dokumentace delšího formátu sloužící coby „manuál“ k balíčku či jeho částem
- dokumentace funkcí – popisuje konkrétní funkci, její použití, argumenty...

V našem případě s Shiny aplikací slouží vignettes přímo coby návod k použití vytvořené aplikace, který by měl uživateli představit, k čemu lze aplikaci použít, vysvětlit fungování jednotlivých prvků, příp. může být doplněn i o ilustrativní screeny aplikace. Dle povahy aplikace (a náročnosti jednotlivých mechanismů) může jít buď o jeden souhrnný soubor, nebo může být vytvořeno více návodů pro dílčí části. V obou případech lze kostru takového návodu vytvořit pomocí funkce `usethis::use_vignette("name.of.file")`, která se nachází v souboru `02_dev.R`. Kostra se umístí do automaticky vytvořeného podadresáře *vignettes*, do kterého se pak generují i další případné návody (funkci lze použít opakovaně). Vytvořený soubor je ve formátu *.Rmd* – při jeho tvorbě se tak postupuje jako u jakéhokoliv markdown dokumentu, zde navíc s možností používat výpočty (či vytvářet grafy, produkovat výstupy modelů atd.) pomocí softwaru R prostřednictvím částí kódu v angličtině označovaných jako *chunk*. Ze všech takto připravených *.Rmd* souborů pak sestavíme

HTML vignettes pomocí funkce `devtools::build_vignettes()`, která je opět k dispozici i v rámci skriptu `02_dev.R`. Pro kontrolu sestavených návodů můžeme doplnit tento skript o funkci `utils::browseVignettes("name.of.your.pkg")`, kterou se dá sestavený HTML návod zobrazit v prohlížeči.

Dokumentace funkcí je, resp. byla poměrně specifická disciplína, která se velmi usnadnila díky balíčku `roxygen2` [46], který umožňuje vývojářům pracujícím v R dokumentovat funkce v rámci stejného skriptu, ve kterém je definovaná příslušná funkce, a to pomocí řádků kódu tvořících tzv. *roxygen bloky*. Ty jsou od „běžného" kódu odlišeny tím, že začínají symbolem `#'`.[12] V těchto blocích se pak informace o funkci zadávají pomocí tzv. „štítků" (z anglického *tag*). Jejich kompletní seznam naleznete v dokumentaci balíčku [32], my na příkladu z Obrázku 2.7 (dokumentace modulu) rozebereme alespoň některé.

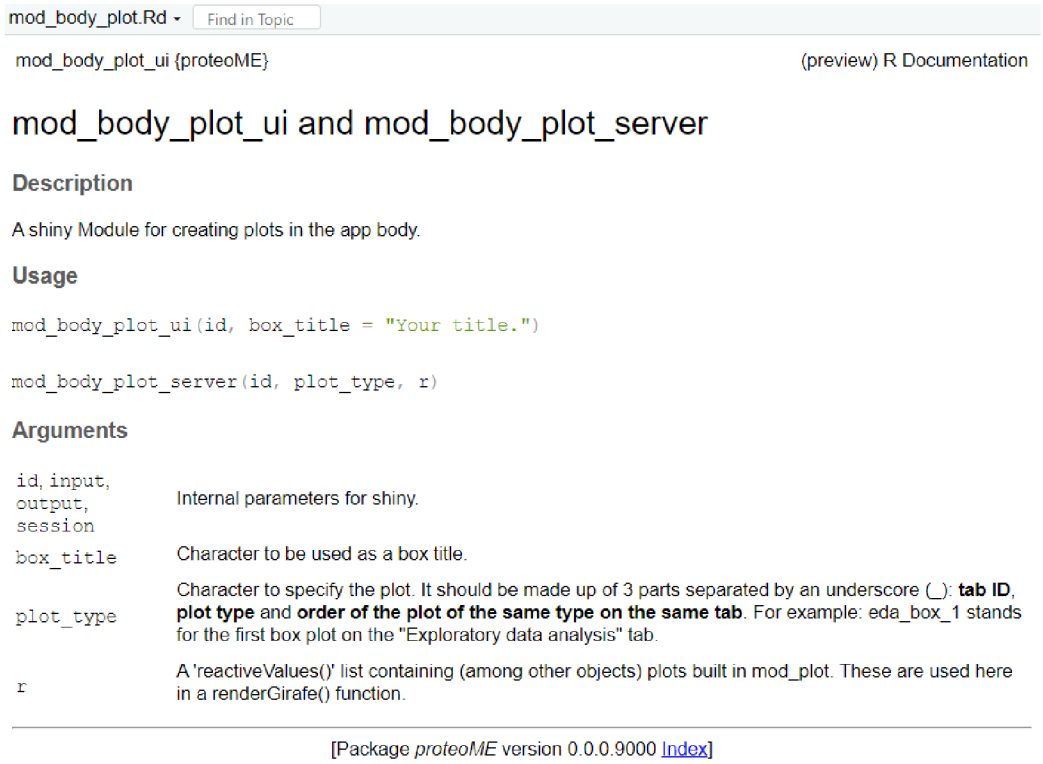
```
# body_plot UI Function

#' @title mod_body_plot_ui and mod_body_plot_server
#' @description A shiny Module for creating plots in the app body.
#'
#' @param id,input,output,session Internal parameters for {shiny}.
#' @param box_title Character to be used as a box title.
#'
#'
#' @rdname mod_body_plot
#'
#' @export
#'
#' @importFrom ggiraph renderGirafe girafeOutput
```

Obrázek 2.7: Příklad dokumentace funkcí prostřednictvím balíčku `roxygen`

Štítky `@title` (název) a `@description` (popis) používáme pro uvedení funkce a přehled toho, k čemu slouží. Pokud má funkce jakékoliv parametry, je potřeba je uvést pod štítkem `@param`. Tento blok kódu před samotnou funkcí může sloužit pouze coby „zápisník vývojáře" a nemusí být uživateli

[12] V tomto textu jste je již mohli zaznamenat např. na Obrázku 2.6.

dostupná. Pokud však přidáme štítek `@export`, vytvoří se automaticky *.Rd* soubor v podadresáři *man* (s názvem uvedeným pod štítkem `@rdname`) – z toho se pak generuje stránka s nápovědou k funkci, kterou jistě každý uživatel RStudia zná a běžně používá. U modulů máme v jednom souboru dvě funkce (pro UI a server) – i přes rozdělení roxygen bloku na dvě části je však možné použít stejný soubor (stejné `@rdname`) a mít tak dokumentaci celého modulu v rámci jednoho souboru. U serverové části roxygen bloku se v tomto případě již nemusí udávat název a popis [32]. Příklad výsledné stránky s dokumentací modulu na záložce RStudia *Help* můžete vidět na Obrázku 2.8.



Obrázek 2.8: Příklad stránky s dokumentací modulu na záložce Rstudia *Help*

Při zabudování datové sady do balíčku je vhodné vytvořit dokumentaci

i pro ni. Postupuje se podobně jako u dokumentování funkcí, avšak je zde drobný problém – datovou sadu jsme nevytvářeli v rámci žádného skriptu, který by se nacházel ve složce *R* a v němž by roxygen bloky s údaji o datové sadě mohly být. Doporučuje se proto vytvořit v této složce soubor speciálně pro dokumentaci dat (např. `data.R`). V něm můžete vytvořit dokumentaci pro všechny datové soubory – za každým roxygen blokem následuje pouze název datové sady v uvozovkách. Příklad takové dokumentace můžete vidět na Obrázku 2.9. Štítek `@format` slouží ke shrnutí základních údajů datové sady (např. počet řádků a sloupců) a vysvětlení jednotlivých proměnných. Štítek `@source` zase může obsahovat zdroj, z kterého data pochází. Zároveň u této dokumentace nikdy nepoužíváme štítek `@export` (v nápovědě RStudia však bude dokumentace dostupná i tak) [25].

```
#' @title Example data with run annotations
#'
#' @description This dataset contains run annotations for *data_example* dataset
#' to be used in a proteoME Shiny app.
#'
#' @format
#' A data frame with 200 rows and 6 columns:
#' \describe{
#'  \item{runID}{A column with run IDs in a form of F# (where # is a number). Each ID
#'  has to be unique.}
#'  \item{sampleID}{A column with sample IDs (one for each patient). It consists of
#'  a letter indicating a treatment group and a number indicating the patient
#'  number within the group. E.g. ID A2 stands for the second patient within a
#'  treatment group A. The same sample ID can be assigned to several run IDs.}
#'  \item{rep}{A column with replicate number. E.g. Run F4 is the second
#'  replicate for a patient A2.}
#'  \item{batchNo}{Batch number.}
#'  \item{batchOrder}{Order of the run within the batch.}
#'  \item{dateTime}{Date and time of the measurement in a \%m/\%d/\%Y \%R format.}
#'
#' }
#' @source <https://raw.githubusercontent.com/TomChupan/data/main/data_example_run2.csv>
#'
"ann_run_example"
```

Obrázek 2.9: Příklad dokumentace zabudované datové sady

V části 2.4 již bylo zmíněno, jak přidat používané balíčky do souboru `DESCRIPTION`. Jednotlivé funkce[13] k importování se potřebují propsat do

[13]Málokdy jsou potřeba veškeré funkce z nějakého R balíčku, proto je lepší do vlastního

souboru `NAMESPACE`. To zastane funkce `golem::document_and_reload()`, o které byla v tomto kontextu řeč také v části 2.4. Jak to souvisí s dokumentováním? Importované funkce se totiž zapisují k ostatním dokumentovaným údajům v rámci roxygen bloku a `golem::document_and_reload()` si je odsud vytáhne a propíše je do `NAMESPACE`. Jde o štítek ve tvaru `@importFrom package function` – jeho použití lze vidět na Obrázku 2.7. V případě importu více funkcí z jednoho balíčku se pouze v rámci příslušného řádku přidávají další funkce oddělené mezerou. Pokud by bylo z nějakého (dobrého) důvodu potřeba importovat celý balíček, lze pro to použít parametr ve tvaru `@import package` – takto je defaultně importován[14] balíček `shiny`, jehož funkce tak můžeme libovolně používat napříč aplikací a o jejich import se starat nemusíme. U modulů stačí díky výše zmíněné společné dokumentaci uvádět používané funkce pouze v prvním roxygen bloku nad UI (nezáleží na tom, zda se pak funkce používají v UI či serverové části modulu) [8][32].

2.6 Publikování

V momentě, kdy je aplikace připravena a dokumentace podrobně sepsána, nastává okamžik, ke kterému celý projekt od začátku směřoval – publikace balíčku. Pomocníkem při tomto procesu je již zmíněný soubor `03_deploy.R` z podadresáře *dev*. Prvním krokem[15] by měla být kontrola balíčku pomocí zde uvedené funkce `devtools::check()` – ta plní podobnou úlohu jako výše zmíněná funkce `rcmdcheck()`, avšak obě pracují trochu jinak a obě mohou vývojáři poskytnout jiné, užitečné komentáře a podněty k opravám. Klíčovým závěrem je počet chyb, varování a poznámek (*errors*, *warnings*, *notes*) –

balíčku importovat pouze dílčí funkce, které jsou skutečně použity.

[14] Toho si můžete všimnout v roxygen bloku souborů `app_ui.R` a `app_server.R`.

[15] Nezapomínejte být pořád v aktivním projektu daného balíčku – to platí i pro následující kroky.

v ideálním případě by všechny tyto „nežádoucí“ kategorie měly být s nulovým výskytem, avšak případné poznámky jsou průchozí. Pokud však přetrvávají chyby či varování, snažte se je před pokračováním opravit či vyřešit [8].

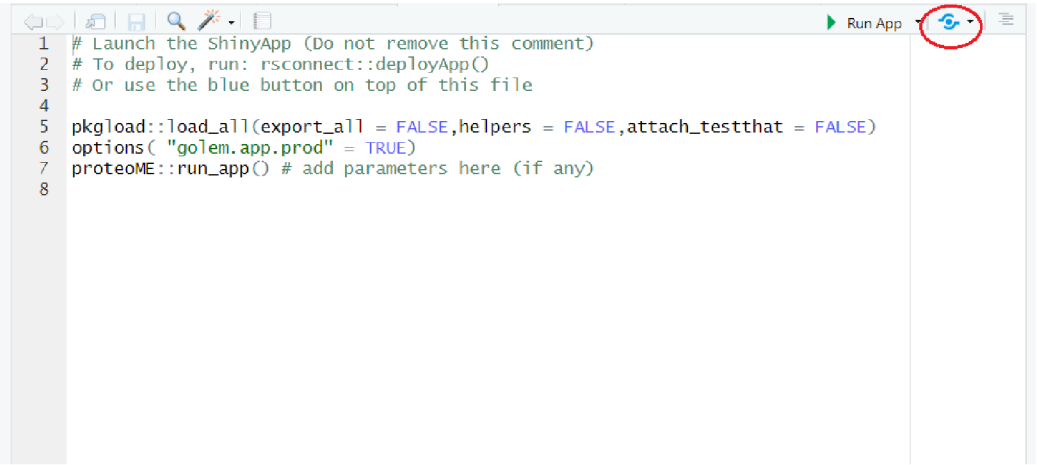
Před veřejnou publikací je vhodné zkusit balíček nejprve nainstalovat na vaše zařízení a ověřit jeho funkčnost. To lze provést příkazem `remotes::install_local()` a následně restartovat R a zavolat balíček pomocí funkce `library("name.of.your.pkg")` (a samozřejmě spustit aplikaci příkazem `run_app()`). Vše běží, jak má? Pokud jste instalaci provedli na zařízení, na kterém jste celou dobu balíček vyvíjeli, má již nainstalované všechny potřebné *dependencies* pro hladký chod vašeho balíčku a aplikace. Proběhlo by vše takto bez problémů, pokud by se balíček instaloval na nové zařízení? Je lepší to zkusit. Sestavit balíček do tzv. *Package Archive File* (ve formátu *.tar.gz*, z kterého lze v RStudiu balíčky instalovat) lze pomocí funkce `devtools::build()` ze souboru `03_deploy.R`. Archiv s balíčkem pak lze přenést na libovolné zařízení se softwarem R a pokusit se o jeho instalaci tam – opět pomocí funkce `remotes::install_local()`, přičemž nyní je třeba jako argument uvést cestu k souboru s balíčkem. Instalace pomocí této funkce zajistí správné nainstalování všech *dependencies* [8].

Pokud vše proběhlo v pořádku, balíček může být nabídnut veřejnosti. Jsou dva základní způsoby, jak k tomu přistoupit. Tím prvním je umístění balíčku do některého z dostupných veřejných repozitářů balíčků (CRAN, BioConductor, GitHub) – nejjednodušší cestou je právě GitHub, kam stačí nahrát celý adresář s balíčkem do stejnojmenného osobního repozitáře (který musí být veřejně dostupný[16]). Kdokoliv si jej pak může do svého R nainstalovat pomocí funkce `devtools::install_github()`, do níž jako argument uvede `"user.name/pkg.name"` odpovídající přezdívce autora na GitHubu a názvu

[16]Na GitHubu se dají vytvořit i soukromé repozitáře, avšak publikace balíčku někam, kam se nikdo nedostane, jaksi nedává smysl.

balíčku[17] [8].

Druhou cestou je umožnit užívání aplikace i těm, kteří R nepoužívají, příp. se jim v tomto prostředí aplikace používat nechce. V tomto případě lze aplikaci umístit někam na server – pro její používání pak stačí kliknout na daný odkaz a aplikace se objeví v prohlížeči. Pro uživatele zcela snadné. Zde si zkusíme v pár krocích shrnout postup ze strany vývojářů tak, aby to bylo snadné i pro ně. Ne každý má svůj server, na který může aplikaci umístit. Představíme zde řešení dostupné na stránce https://www.shinyapps.io, kde se stačí zaregistrovat jednou z běžných cest[18] a v případě konzervativních plánů pro užívání (nikoliv masové rozšíření aplikace) zvolit *free plan*, který má sice omezený počet publikovaných aplikací a počet aktivně využívaných hodin, ale je zdarma. Pokud je aplikace ambicióznější, lze samozřejmě zvolit placený plán s přívětivějšími parametry za měsíční či roční poplatek [8].



Obrázek 2.10: Nastartování procesu publikace na server

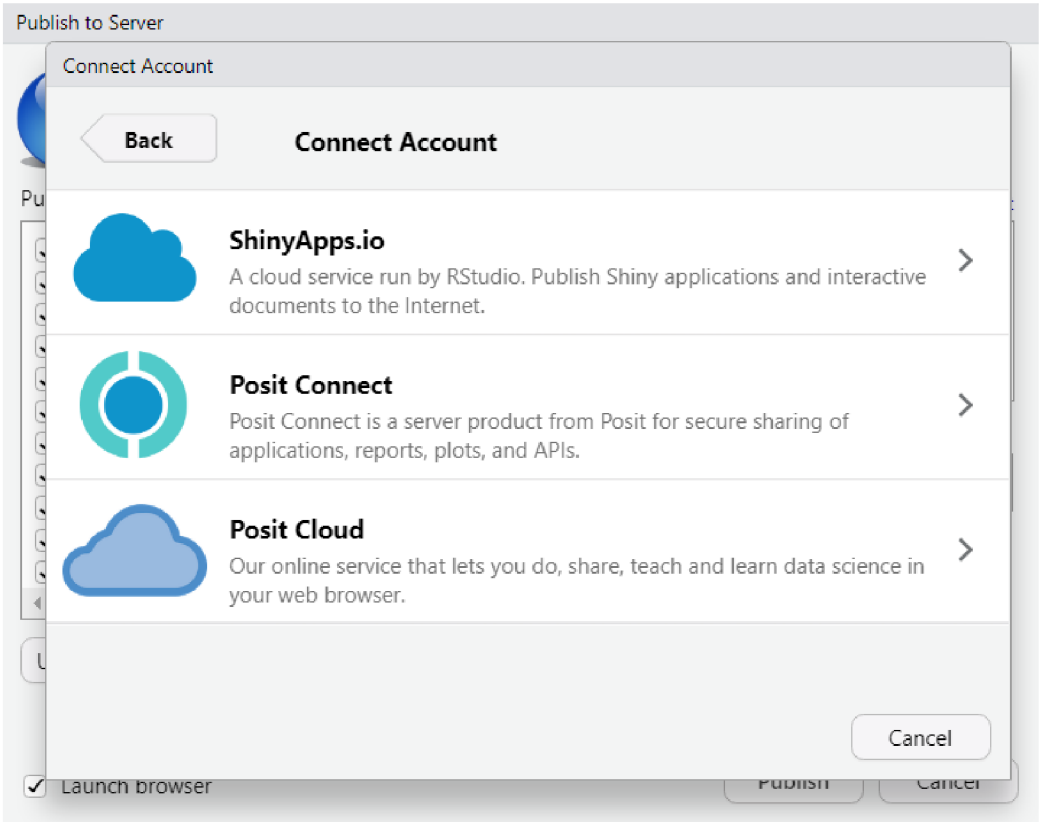
V souboru `03_deploy.R` lze provést přípravný krok pro publikaci do

[17] Technicky jde o název GitHub repozitáře, nikoliv název balíčku, avšak již bylo zmíněno, že se musí shodovat – zde na to proto ještě jednou upozorněme.

[18] Kromě e-mailu či Google účtu je tu i praktická možnost přihlásit se přes vlastní GitHub účet – v případě publikace balíčku na GitHubu tak lze mít vše pod jedním účtem.

výše zmíněného prostředí pomocí funkce `golem::add_shinyappsio_file()`, která vytvoří mj. soubor `app.R`, z kterého se bude aplikace na serveru spouštět. V tomto souboru lze také spustit proces publikace, a to pouhým stisknutím modrého tlačítka v „záhlaví“ souboru v rámci RStudia – vizte Obrázek 2.10. Po stisknutí tohoto tlačítka vás RStudio v případě, že je to první publikovaná aplikace, vyzve k nastavení a volbě publikačního účtu a platformy. Ze tří dostupných možností zvolte *ShinyApps.io* (vizte Obrázek 2.11). Dále pokračujte dle pokynů zobrazených v tomto okně. Po nastavení účtu se nacházíte v okně, kde jsou defaultně k publikaci zakliknuté všechny soubory z adresáře balíčku – to je potřeba zachovat a publikovat jej opravdu celý. Publikaci pak stačí potvrdit tlačítkem *Publish*. Pokud se při procesu publikace objeví nějaké problémy (např. nenainstalované balíčky, které proces potřebuje), postupujte dle pokynů v konzoli RStudia [8].

Po úspěšné publikaci je váš balíček dostupný všem, kterým odkaz pošlete (případně lze odkaz přidat i do `README` balíčku, které se ukazuje jako úvodní stránka v daném repozitáři na GitHubu). Míra a forma rozšíření této informace se bude pravděpodobně odvíjet i od úrovně plánu na *ShinyApps.io* (a souvisejícího omezení na počet aktivních hodin užívání aplikace). Tak či tak jste nyní u konce projektu. Nebo stále na jeho začátku? Pravděpodobně budou přicházet návrhy na vylepšení a požadavky na další zpřístupněné funkce. Vy jste však díky tvorbě aplikace pomocí postupu uvedeného v této kapitole připraveni a údržba a zdokonalování aplikace nebude představovat žádný problém.



Obrázek 2.11: Volba účtu k publikaci

Kapitola 3

Aplikace proteoME

První kapitola se věnovala především práci s proteomickými daty, ta druhá zase tvorbě Shiny aplikace ve formě balíčku v softwaru R. Tato kapitola obě předchozí části spojuje a je zároveň průvodcem realizací hlavního cíle práce. Tím je totiž nová Shiny aplikace *proteoME* ze stejnojmenného R balíčku, který autor tohoto textu vytvořil právě za účelem usnadnění celého procesu zpracování a analýzy proteomických dat pro ty, kteří neovládají programovací jazyky, případně si jen chtějí ušetřit čas a využít nabízeného uživatelského prostředí.

Uživatelé R mohou využít možnost instalace balíčku a spouštění aplikace přímo v tomto softwaru (resp. v rozšířeném RStudiu). Instalaci balíčku lze provést příkazem

```
devtools::install_github("TomChupan/proteoME")
```

a aplikaci lze posléze spustit pomocí příkazů

```
library(proteoME)
proteoME::run_app()
```

Možnost využívat aplikaci lze však i mimo prostředí R (bez nutnosti cokoliv instalovat) – od června 2024 bude v repozitáři balíčku na GitHubu

zveřejněn i odkaz na *ShinyApps.io*, na kterém si každý (bez ohledu na to, zda má nainstalované R či nikoliv) může aplikaci vyzkoušet a rozhodnout se o její případné budoucí instalaci[1]. Tato dostupnost rozšíří množinu možných uživatelů a představuje další usnadnění práce s tímto typem dat. Dalším faktorem umožňujícím používání aplikace širokému spektru uživatelů je její jazykové provedení – veškerý obsah i dokumentace jsou v angličtině.

Uživatelské prostředí aplikace je rozděleno na postranní panel a tělo (obsahující hlavní výstupy ve formě tabulek, grafů apod.), v rámci těla aplikace navíc uživatel přepíná mezi jednotlivými záložkami, přičemž postranní panel se při přepínání dynamicky mění. Postupně si představíme jednotlivé záložky, nabízené nástroje a funkce – to vše na souboru syntetických datových sad (inspirovaných reálnými daty), které jsou rovněž součástí balíčku *proteoME* (vizte dále). Tato kapitola tak zároveň slouží jako návod k použití aplikace s praktickou ukázkou analýzy proteomického datasetu. Názvy podkapitol budou kopírovat názvy záložek v aplikaci – budou tedy psány anglicky, což bude vhodné i pro odlišení od částí první kapitoly.

3.1 Data import

V této části si nejprve představíme datové sady, které jsou součástí balíčku *proteoME*. Jednak půjde o popis těchto souborů, jednak jde o ukázku formátu, který je nutné dodržovat při nahrávání vlastní datové sady. Následně si ukážeme první záložku aplikace sloužící k nahrání souborů s daty a jejich základní zobrazení a úpravu.

V podkapitole 1.1.2 byl uveden příklad datové sady s abundancemi pro-

[1] Používat aplikaci dlouhodobě jen na tomto odkaze v blízké budoucnosti nebude možné kvůli omezení počtu aktivních hodin danému serverem *ShinyApps.io* při využívání plánu, který je zdarma.

teinů. Ve stejném formátu je první z ukázkových souborů, který po načtení balíčku můžete „zavolat“ příkazem `data_example`. Tato datová tabulka má celkem 1018 řádků, které představují jednotlivé proteiny, a 101 sloupců. První sloupec[2] je tvořen unikátními identifikátory proteinů (*accession number*), zbylé sloupce obsahují abundance proteinů pro jednotlivé soubory[3] – názvy sloupců odpovídají ID souborů. Abundance nejsou ani transformované či normalizované (jde o tzv. *raw*[4] hodnoty abundancí, nebyla s nimi provedena žádná úprava), ani imputované (v datech jsou tedy chybějící hodnoty). Část této datové sady můžete vidět v Tabulce 3.1.

Tabulka 3.1: Ukázka datového souboru `data_example`

Accession	F1	F2	F3	F4	F5	F6
P04745	36 164,74	20 378,79	19 579,20	*NA*	153,92	2 015,13
B7ZMD7	9 409,04	2 564,65	11 163,22	92 742,42	75 504,78	69 845,65
H6VRF8	597 164,63	53 611,28	39 914,64	37 792,98	9 397,29	12 515,90
P19961	4 115,53	7 102,86	8 042,43	11 195,39	12 463,95	16 082,41
P35527	144 533,14	110 654,87	73 610,83	3 804,32	4 834,82	329,12
P13645	62 661,85	85 856,07	84 322,87	23 734,81	28 264,32	34 264,96

Další dostupnou datovou sadou jsou anotace souborů. V části 1.2 u nich byl uveden anglický pojem *file annotations*, stejný význam má v kontextu proteomických dat termín *run annotations* (jak jsme si už uvedli – soubor je označení pro softwarový výstup běhu – vizte část 1.1.2). Tento soubor lze načíst příkazem `ann_run_example` a obsahuje 100 řádků (pro každý běh/soubor jeden) a 6 sloupců – vizte Tabulku 3.2.

První sloupec tvoří ID souborů (*runID*), musí tak být **shodný** s názvy druhého až posledního sloupce datové sady s abundancemi. Sloupec *sampleID* pak každý soubor přiřazuje k vzorku (subjektu), z kterého pochází. Ve třetím

[2]Název sloupce *Accession* je povinný.
[3]Tato terminologie je vysvětlena v první kapitole.
[4]V češtině lze použít ekvivalent „surové“ hodnoty.

Tabulka 3.2: Ukázka datového souboru `ann_run_example`

runID	sampleID	rep	batchNo	batchOrder	dateTime
F1	A1	1	1	1	07/03/2023 05:23
F2	A1	2	1	2	07/03/2023 10:56
F3	A2	1	1	3	07/04/2023 09:06
F4	A2	2	1	4	07/04/2023 16:37
F5	A3	1	1	5	07/05/2023 17:48
F6	A3	2	1	6	07/06/2023 00:23

sloupci (*rep*) je informace o tom, kolikátý replikát daného vzorku tento soubor představuje. Sloupce *batchNo* a *batchOrder* obsahují údaje o označení dávky, resp. pořadí replikátu v rámci dávky. V posledním sloupci je datum a čas měření – v R použijte pro převod na tento formát řetězec `%m/%d/%Y %R`. První dva sloupce jsou v současné verzi aplikace povinné, zbylé nemusí být přítomny[5]. Opět je však povinné držet se předepsaných názvů sloupců.

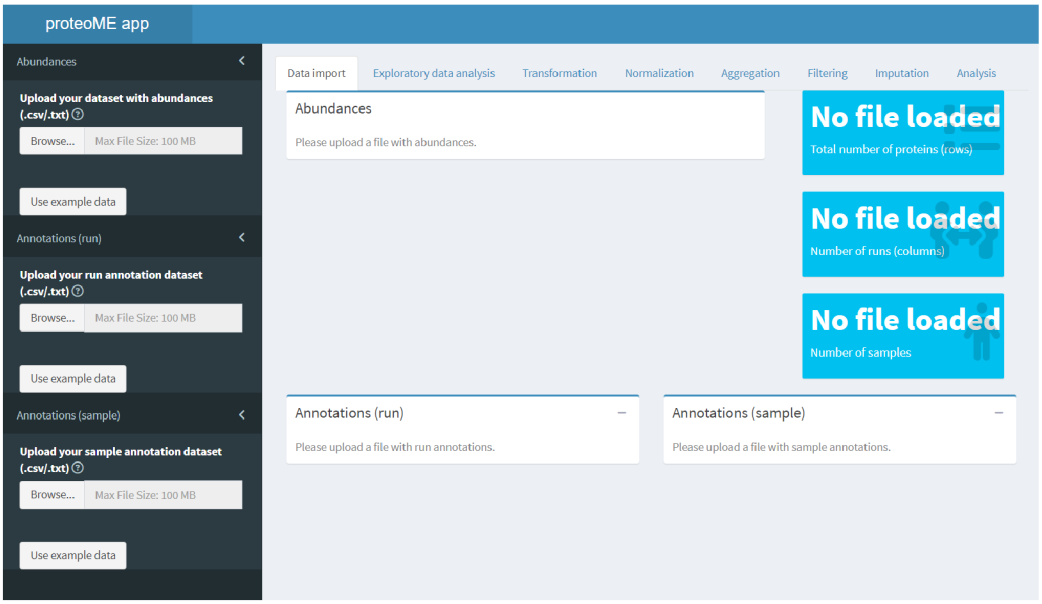
Posledním souborem, který balíček *proteoME* zahrnuje, jsou anotace vzorků. Tento dataset lze načíst příkazem `ann_sample_example` a obsahuje 50 řádků (pro každý vzorek jeden) a 5 sloupců. Jak už bylo zmíněno v první kapitole, tento typ datové sady obsahuje údaje o jednotlivých subjektech (pacientech). Kromě identifikátoru jde především o skupinu léčby (*treatment*), dále však mohou být dostupné další libovolné údaje (např. pohlaví). První sloupec s identifikátorem vzorku (*sampleID*) musí odpovídat unikátním hodnotám ze stejnojmenného sloupce datové sady s anotacemi souborů. Názvy prvních dvou sloupců jsou povinně v této formě (kterou můžete vidět i v Tabulce 3.3), na zbylé sloupce se žádné požadavky nekladou.

Pojďme se nyní podívat na záložku pro import dat, kterou uživatel uvidí při spuštění aplikace – vizte Obrázek 3.1. Na postranním panelu jsou 3 po-

[5]Jde o přípravu na analýzu *batch efektu*, která je plánovaná až v pozdějších verzích aplikace.

Tabulka 3.3: Ukázka datového souboru `ann_sample_example`

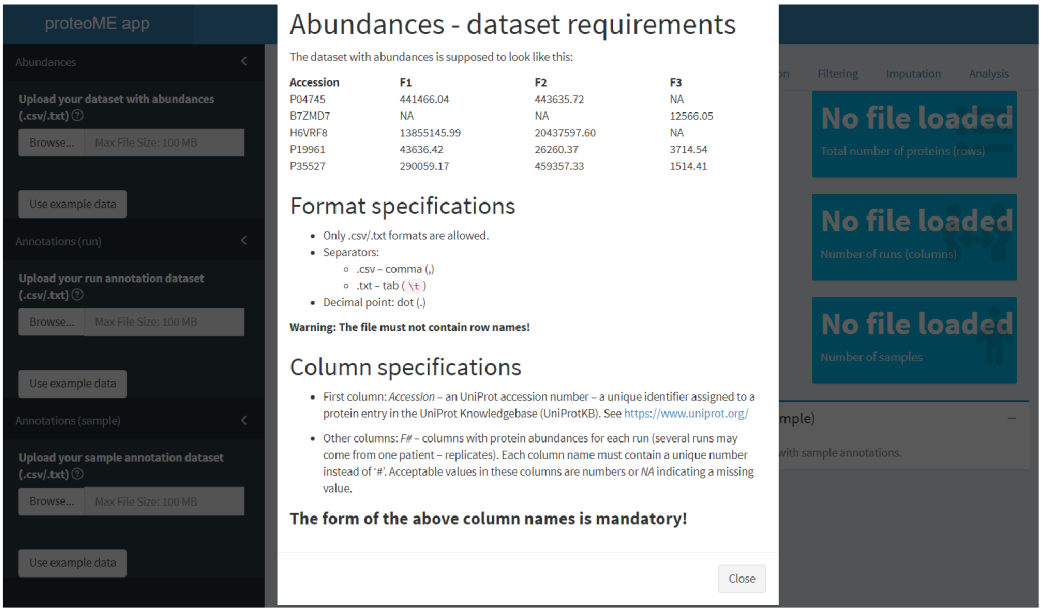
sampleID	treatment	sex	age	weight_kg
A1	A	M	76	61
A2	A	M	46	65
A3	A	M	70	79
A4	A	F	67	69
A5	A	F	56	59
A6	A	F	69	72



Obrázek 3.1: Screenshot aplikace *proteoME* (Data import) – úvodní obrazovka

dobné položky menu – pro každý z výše uvedených typů datové sady jedna – s nahrávacím vstupem a tlačítkem *Use example data*. Uživatel tak může buď nahrát svůj vlastní dataset, příp. může pomocí zmíněného tlačítka načíst do aplikace výše popsané datové sady integrované do balíčku *proteoME*. Nad nahrávacím vstupem je za pokynem k nahrání malá ikona otazníku v kroužku. Po kliknutí na tuto ikonu se otevře okno s nápovědou, jak má příslušná da-

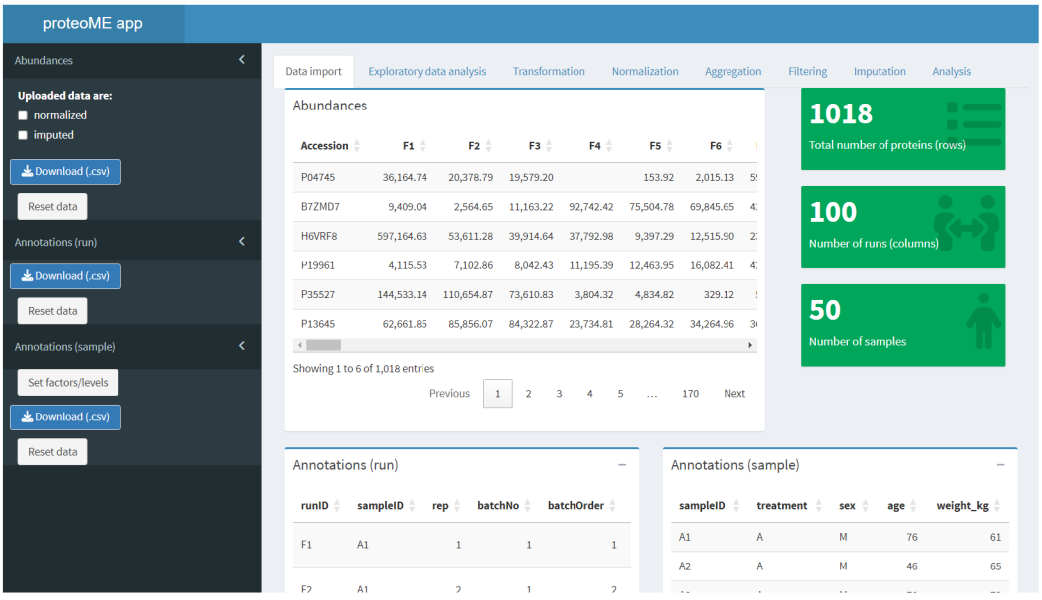
tová sada vypadat a jaké jsou na ni kladeny požadavky. Zde kromě výše uvedeného zdůrazněme formát souboru (přípustné jsou soubory formátu *.csv* a *.txt*), znak pro oddělení desetinného místa (akceptuje se pouze desetinná tečka) či nutnou absenci názvů řádků. Ukázka jedné z nápověd je na Obrázku 3.2.



Obrázek 3.2: Screenshot aplikace *proteoME* (Data import) – nápověda k datové sadě

V hlavní části aplikace jsou připravená tři okna a tři infoboxy čekající na import datové sady. Pracujme tedy dále s ukázkovými soubory. Po stisknutí tlačítka *Use example data* ve všech třech položkách menu se v oknech zobrazí tabulky s načtenými daty a do infoboxů se vyplní příslušné hodnoty udávající počet proteinů (řádků dat s abundancemi), počet běhů (sloupců dat s abundancemi při vynechání prvního sloupce s identifikátory proteinů) a počet vzorků. Na postranním panelu zmizí dosavadní možnosti importu dat a pro každý typ datové sady se objeví tlačítko pro stažení aktuální po-

doby dat (to bude důležité dále po transformaci a normalizaci) ve formátu *.csv*, u datasetu s abundancemi má navíc uživatel možnost označit, zda jsou jeho data již normalizovaná či imputovaná. Tato volba je motivována softwary, které zpracovávají výsledky měření hmotnostního spektrometru a tyto možnosti úpravy dat nabízí – proto je možné, že už jsou data uživatele po těchto úpravách. Kromě toho lze každou datovou sadu resetovat (tlačítkem *Reset data*), čímž se uživateli zpřístupní původní podoba položky menu před nahráním dat. Tento reset má samozřejmě dopad na případné výsledky získané v dalších záložkách. Resetování anotací nemá vliv pouze na transformaci a normalizaci dat, resetem abundancí se vynulují všechny dosažené výsledky. Stav obrazovky po načtení ukázkových datových sad můžete vidět na Obrázku 3.3. Ihned máme informaci, že zahrnují 1018 proteinů, 100 běhů a 50 vzorků, také můžeme prohlédnout a zkontrolovat všechny nahrané datové sady.

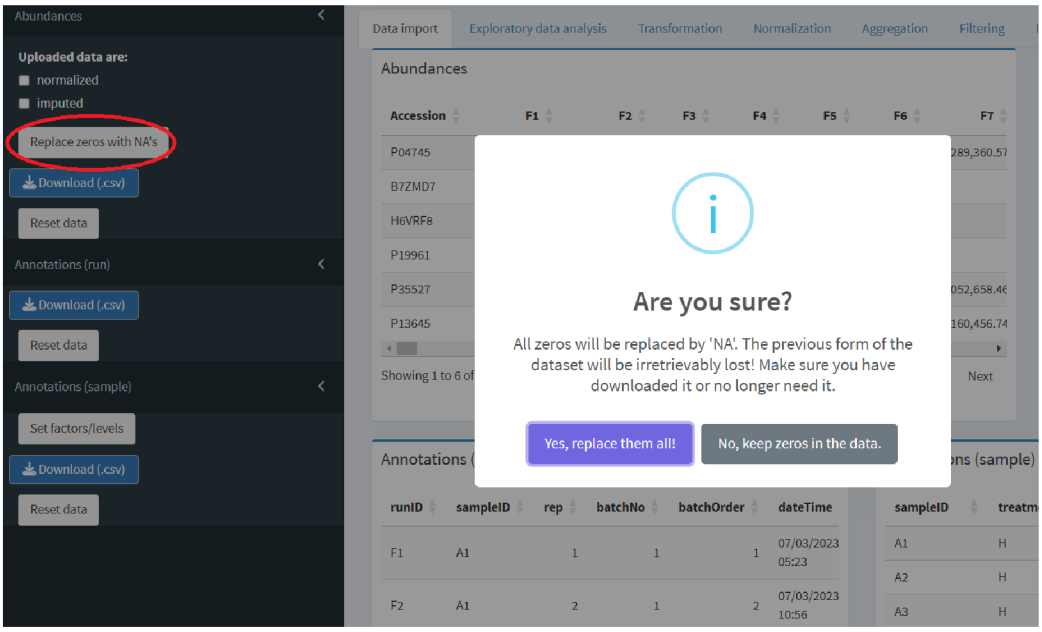


Obrázek 3.3: Screenshot aplikace *proteoME* (Data import) – stav po importu datové sady

Předpokládá se, že datová sada s abundancemi bude v případě surových údajů (bez předchozích úprav) obsahovat pouze nezáporné hodnoty. Se zápornými hodnotami lze pracovat, ale těch lze dosáhnout jen tehdy, pokud už jsou data transformovaná či normalizovaná – uživatel tedy musí spolu s importem datasetu se zápornými hodnotami zaškrtnout i políčko *normalized*. Pokud má uživatel jako výstup měření normalizovaná data, která neobsahují záporné hodnoty a potřebují ještě transformovat, může v tomto případě aplikaci nezaškrtnutím checkboxu tuto informaci „zatajit“, data na příslušné záložce transformovat a proces normalizace přeskočit. V případě, že uživatel nahraje vstupní data, která obsahují nuly, objeví se na postranním panelu další tlačítko – *Replace zeros with NA's*. Přítomnost nuly v buňce tabulky totiž zpravidla znamená absenci daného proteinu v dotyčném souboru (či jeho výskyt pod detekčním limitem). Jde samozřejmě o informačně bohatší hodnotu než *NA* (u které nevíme, zda je MAR či MNAR – vizte část 1.2.6), avšak při agregaci se nula bere jako každá jiná kvantifikovaná hodnota abundance a ovlivní tak výslednou podobu datové sady. Uživatel má proto prostřednictvím tohoto tlačítka možnost nahradit nuly chybějícími hodnotami – sám musí zvážit, co je v jeho situaci lepší. Dialogové okno s potvrzením této volby můžete vidět na Obrázku 3.4. V ukázkových datech se však nuly nevyskytují.

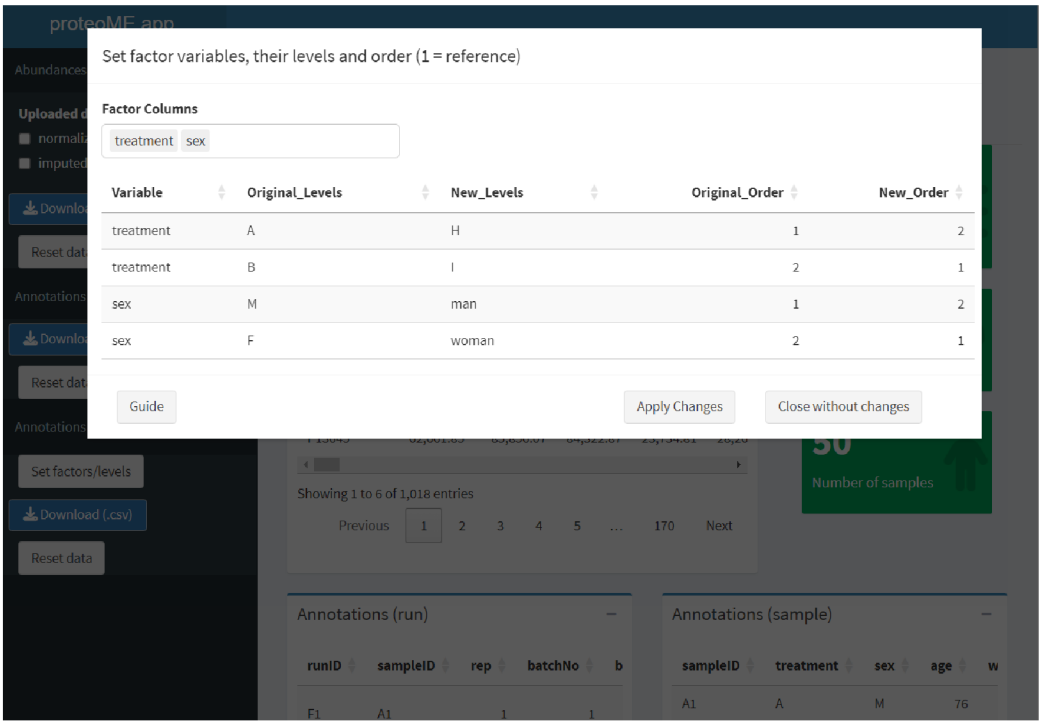
U anotací vzorků je navíc k dispozici tlačítko *Set factors/levels*, kterým uživatel otevře modální okno, ve kterém si může zvolit, které sloupce[6] datasetu s anotacemi vzorků chce označit za faktory. Navíc může změnit hodnoty jednotlivých úrovní (což se propíše i do příslušné datové tabulky) či pořadí úrovní faktorů (úroveň faktoru na pořadí 1 je brána jako referenční) – automaticky sestavená tabulka má totiž editovatelné sloupce *New levels* a *New*

[6] Zde může zvolit jakýkoliv sloupec kromě prvního – zásah do identifikátoru vzorků není povolen.



Obrázek 3.4: Screenshot aplikace *proteoME* (Data import) – nahrazení nul chybějícími hodnotami

Order, které může uživatel po dvojkliku na příslušnou buňku přepsat. K dispozici je také podrobný návod, jak s tímto nástrojem pracovat – návod si uživatel zobrazí kliknutím na tlačítko *Guide* (vizte Obrázek 3.5). Na tomto obrázku je také vidět nastavení sloupců *treatment* a *sex* jako faktorů, navíc s novým názvem úrovní (např. H a I místo A a B) a změnou pořadí (u faktoru *treatment* bude I referenční kategorií). Veškeré změny se potvrdí tlačítkem *Apply changes*. Jakmile máme do aplikace nahrané všechny tři datasety (příp. jsme provedli základní úpravy jako označení abundancí za normalizované/imputované či stanovení faktorových proměnných a úprava jejich úrovní), můžeme se přesunout na další záložku.



Obrázek 3.5: Screenshot aplikace *proteoME* (Data import) – nastavení faktorových proměnných

3.2 Exploratory data analysis

Tato záložka slouží pro základní vizualizaci importovaných dat a jako jeden z nástrojů pro rozhodnutí, co s daty provést v následujících krocích. K dispozici jsou dva grafy, o nichž už byla řeč v části 1.2.1: boxplot abundancí pro jednotlivé soubory a histogram počtu detekovaných proteinů – pro přiblížení těchto vizualizací se tedy odkazuji na zmíněnou pasáž tohoto textu. Každý[7] z grafů má svoji nabídku menu na postranním panelu a své okno v těle aplikace, ve kterém se daný graf vykreslí po stisknutí tlačítka *Render plot*. Na prvním řádku této nabídky je i nápověda ke grafu – opět stačí stisk-

[7]Nejen na této záložce, avšak i u některých dalších vizualizací – jde o použití jednoho shiny modulu na několika místech.

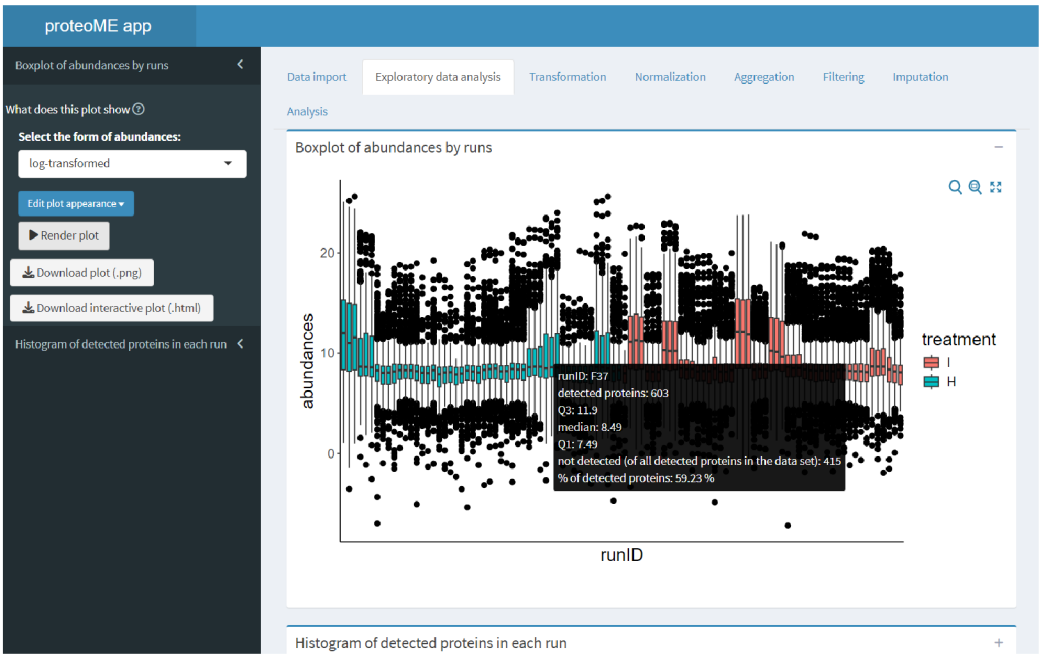
nout ikonu otazníku v kroužku, načež se objeví modální okno s nápovědou, co daný graf ukazuje. Po vykreslení grafu se navíc objeví dvě tlačítka pro stažení – jedno pro statickou verzi grafu (ve formátu *.png*), druhé pro interaktivní verzi (ve formátu *.html*). Všechny grafy produkované tímto typem menu totiž obsahují interaktivní prvky. Některé grafy mají navíc možnost výběru určitého parametru, který závisí na typu grafu – ty si rozebereme u každé vizualizace zvlášť.

Posledním společným prvkem menu pro grafy je tlačítko *Edit plot appearance*. Kliknutím na toto tlačítko uživatel rozbalí dodatečnou nabídku možností pro úpravu grafických parametrů. Okno s touto nabídkou je rozděleno na dvě části – obecnou (aplikovatelnou na libovolný typ grafu) a specifickou pro daný typ grafu. V obecné části lze upravit:

- výšku a šířku grafu
- velikost popisků a názvů os
- velikost textu a nadpisu legendy
- názvy os

Specifickou část této nabídky rozebereme u příslušných typů vizualizací. Jakékoliv změny v tomto rozbalovacím menu potvrzujeme tlačítkem *Apply changes*, čímž se změny projeví ve vykresleném grafu. Ukázku rozbalené nabídky pro úpravu vzhledu grafu (konkrétně pro boxplot) můžete vidět na obrázku 3.7.

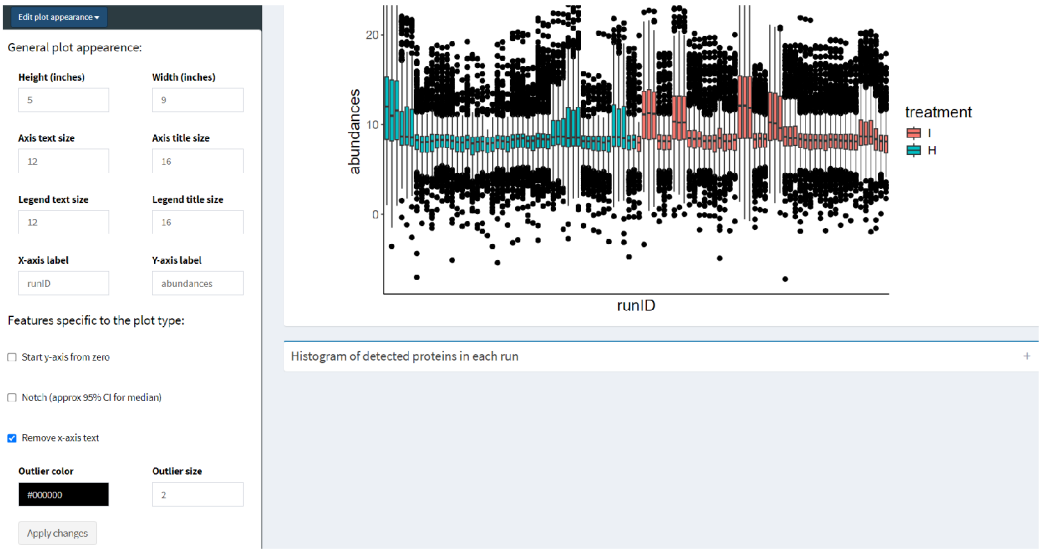
Podívejme se tedy blíže na grafy dostupné na záložce *Exploratory data analysis*. Začněme boxplotem (vizte Obrázek 3.6) – u něj si uživatel volí, zda chce vykreslit přímo hodnoty abundancí (tak, jak je nahrál), nebo je chce v rámci zobrazení grafu zlogaritmovat či odmocnit (kvůli typickému zešikmení – vizte Obrázek 1.1). Pro případ výskytu nul v datech je kromě pouhé



Obrázek 3.6: Screenshot aplikace *proteoME* (Exploratory data analysis) – boxplot abundancí pro jednotlivé soubory

logaritmické transformace dostupná i možnost transformování pomocí funkce $y = log(x + 1)$. Po transformaci či normalizaci dat na dalších záložkách a návratu zpět na záložku *Exploratory data analysis* lze však vykreslit pouze nijak neupravené abundance (*just the values*), neboť nemá smysl data vykreslovat např. dvakrát zlogaritmovaná. Interaktivní infobox zobrazující se při najetí myši na libovolnou „krabici“ obsahuje informace o ID daného souboru, počtu detekovaných proteinů v tomto souboru, dolním a horním kvartilu a mediánu abundancí proteinů v daném souboru, z celkového počtu proteinů v datech je také dopočítaný údaj o počtu nedetekovaných proteinů a o tom, kolik procent z celkového počtu proteinů se v tomto souboru podařilo detekovat. Jednotlivé krabice mají variabilní šířku dle počtu detekovaných proteinů.

Co se úpravy vzhledu grafu týče, u boxplotu má uživatel možnost za-

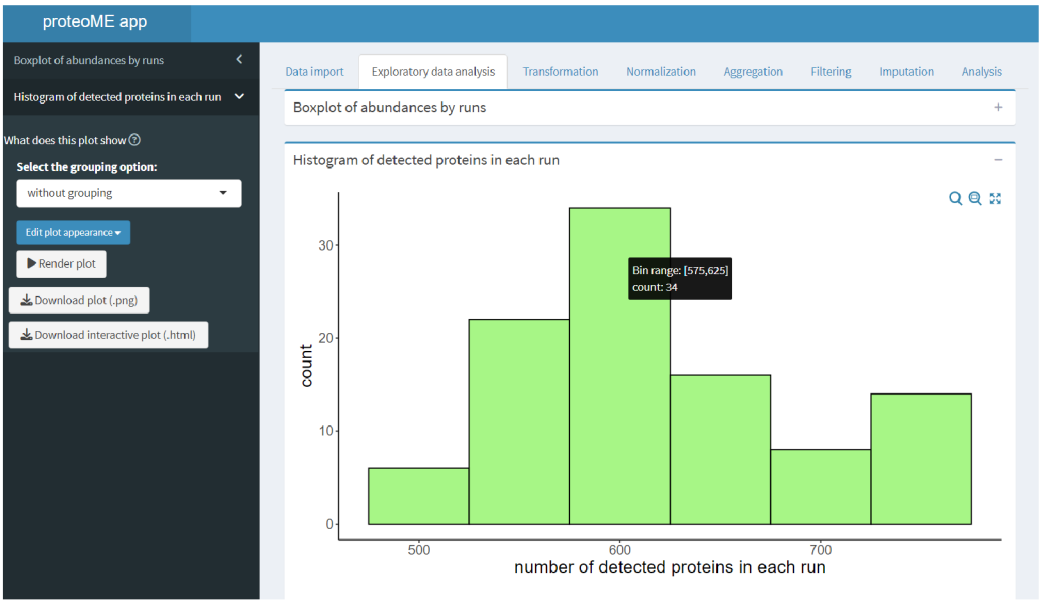


Obrázek 3.7: Screenshot aplikace *proteoME* (Exploratory data analysis) – ukázka rozbalovací nabídky pro úpravu vzhledu grafu (boxplot)

čínat s osou y od nuly, zobrazit konfidenční interval pro medián (resp. jej „vykrojit“ do každé krabice), odebrat popisky[8] osy x, příp. ovlivnit vzhled outlierů (jejich barvu a velikost). Informace, které lze z tohoto typu grafu získat (a které nám mohou usnadnit rozhodování při dalších krocích) jsou rozebrány v části 1.2.1.1.

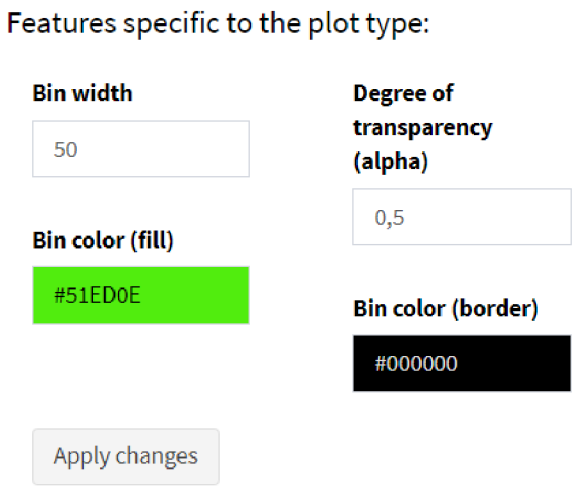
Druhým grafem na této záložce je histogram počtu detekcí v rámci souboru, kterému je věnována část 1.2.1.2 – proto se na tomto místě zaměříme místo vysvětlování toho, co nám ukazuje, na praktickou stránku věci při jeho vykreslování v aplikaci. Rozbalovací seznam s volbami zde obsahuje dvě možnosti vztahující se k zahrnutí skupiny léčby. Na Obrázku 3.8 můžete vidět vykreslený histogram pro ukázková data bez zahrnutí faktoru *treatment*. Interaktivní infobox ukazuje rozsah sloupce, na který uživatel najel myší, a počet běhů, v nichž byl počet detekovaných proteinů v tomto roz-

[8] Při větším množství souborů totiž dochází k překrývání a graf nepůsobí příliš esteticky.

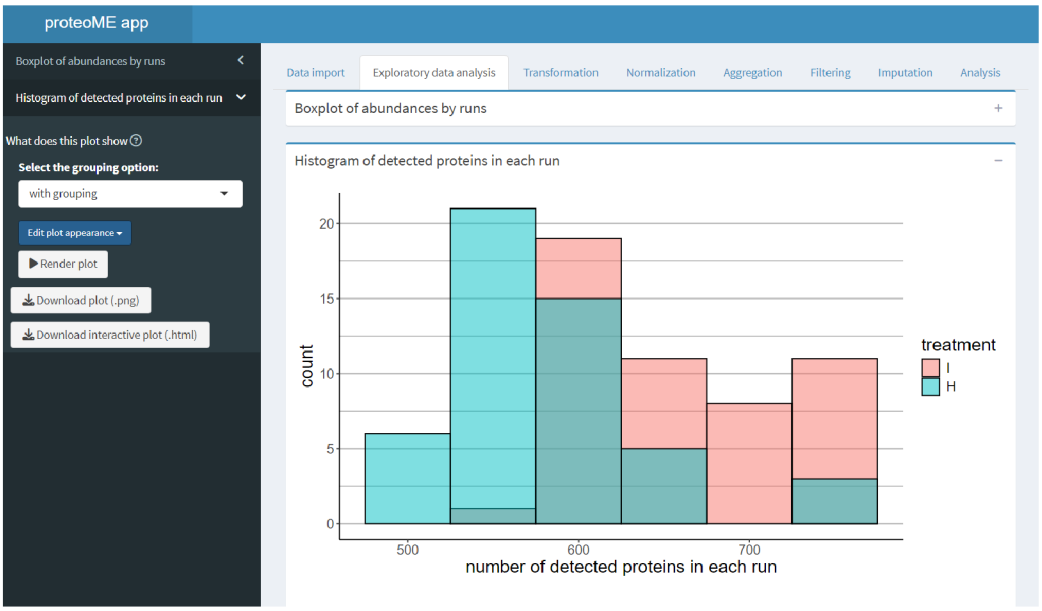


Obrázek 3.8: Screenshot aplikace *proteoME* (Exploratory data analysis) – histogram počtu detekovaných proteinů (bez zahrnutí skupin)

mezí. V rozbalovacím okně s úpravou vzhledu lze u histogramů upravit šířku sloupce (tzv. *bin width*), transparentnost sloupců (hodnota mezi 0 do 1, kde 0 značí absolutní průhlednost sloupců a 1 zcela neprůhledné sloupce) a barvu výplně a okraje sloupců – vizte Obrázek 3.9. Při zahrnutí informace o příslušnosti ke skupině léčby (volba *with grouping*) se pro každou skupinu vykreslí jeden histogram. Díky průhlednosti lze vidět i překrývající se sloupce, avšak interaktivní prvek neumožní reakci se „zakrytými“ sloupci. Proto je v této verzi grafu k dispozici pouze infobox s rozsahem sloupce a četnosti si uživatel musí odečíst z osy y – pro zjednodušení jsou přidány vodorovné čáry (Obrázek 3.10).



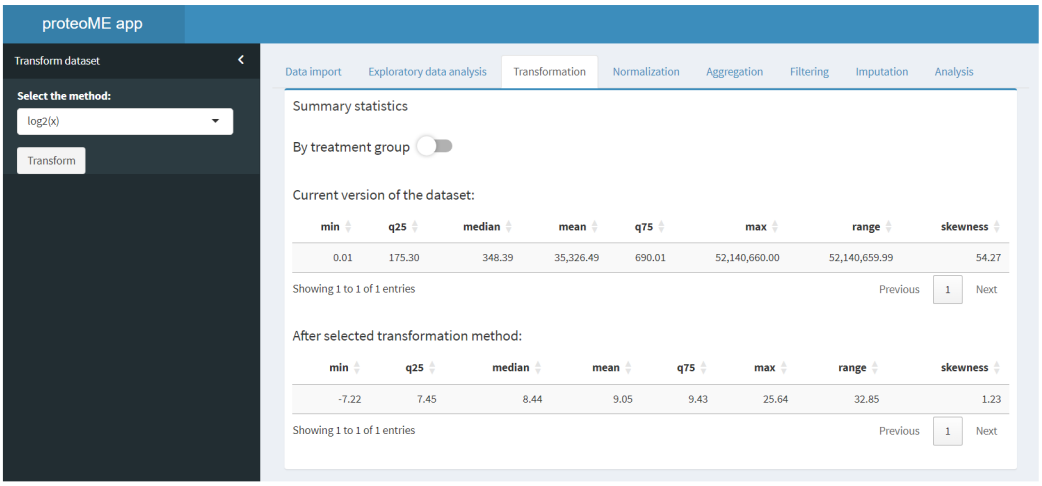
Obrázek 3.9: Screenshot aplikace *proteoME* (Exploratory data analysis) – ukázka rozbalovací nabídky pro úpravu vzhledu grafu (histogram)



Obrázek 3.10: Screenshot aplikace *proteoME* (Exploratory data analysis) – histogram počtu detekovaných proteinů (se zahrnutím skupin)

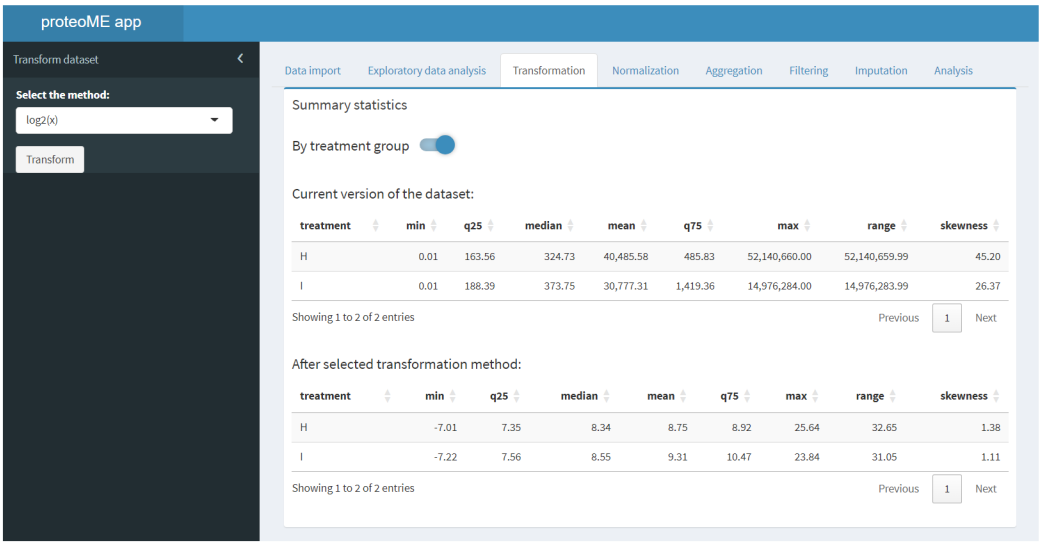
3.3 Transformation

Boxplot na předchozí záložce má uživateli napovědět, zda bude vhodné data transformovat. Záložka *Transformation* slouží přesně k provedení této akce. Volby transformace jsou totožné s možnostmi zobrazení dat v boxplotu (tedy $log(x)$, $log(x+1)$ a odmocnina), kromě této selekce je v postranním panelu už jen tlačítko *Transform*, které danou akci provede. V těle aplikace jsou na této záložce nejprve 2 tabulky se souhrnnými číselnými charakteristikami abundancí (vizte Obrázky 3.11 a 3.12), které odpovídají Tabulkám 1.4 a 1.5 uvedeným v první kapitole. Uživatel si sám přepínačem zvolí, zda se chce na tyto charakteristiky dívat celkově, nebo s rozlišením skupiny léčby.



Obrázek 3.11: Screenshot aplikace *proteoME* (Transformation) – tabulky s číselnými charakteristikami (bez zahrnutí skupin)

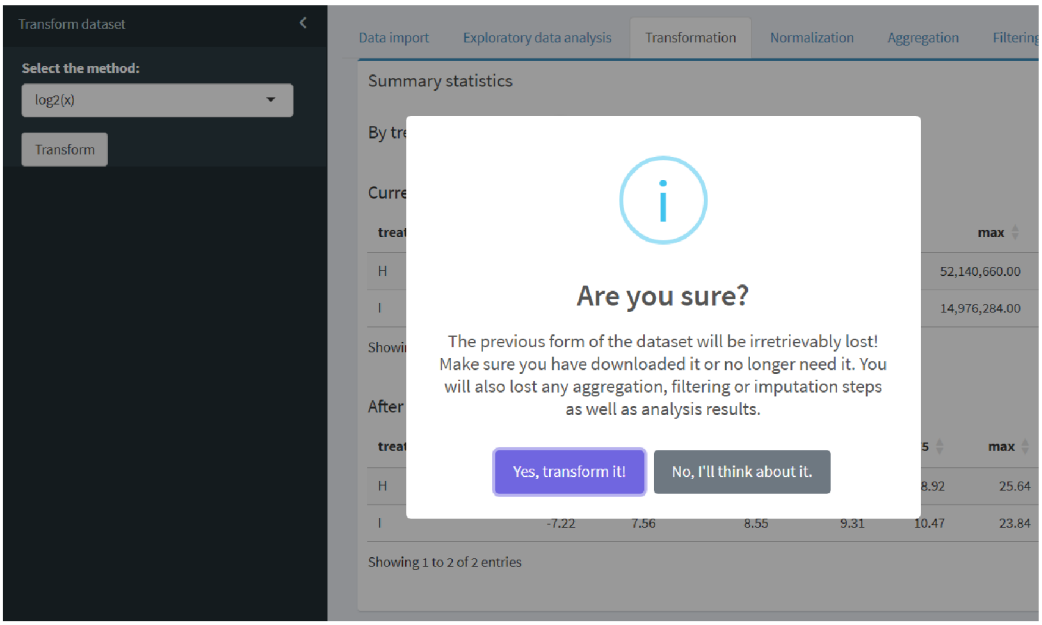
V horní tabulce jsou tyto údaje spočítány z aktuální verze datové sady s abundancemi, v té dolní je ukázán efekt transformace zvolené v nabídce menu – druhá tabulka se tak interaktivně mění v závislosti na tomto vstupu. Uživatel má tedy možnost si i na základě těchto hodnot rozmyslet, kterou transformaci (pokud nějakou) chce použít. Jakmile se rozhodne, může stisk-



Obrázek 3.12: Screenshot aplikace *proteoME* (Transformation) – tabulky s číselnými charakteristikami (se zahrnutím skupin)

nout tlačítko *Transform*. Objeví se vyskakovací okno, ve kterém je požádán o potvrzení a seznámen s následky této akce (Obrázek 3.13). Tento typ potvrzení před akcí zasahující do dat je přítomen i v následujících záložkách u všech momentů, kdy může uživatel nějaká data ztratit. Zde může např. dojít k resetu agregované datové sady (v případě, že by uživatel nejprve data agregoval na úroveň vzorků a až následně je chtěl na úrovni souborů transformovat) – toto „dvoufázové“ potvrzení akce by mělo zamezit nechtěným ztrátám.

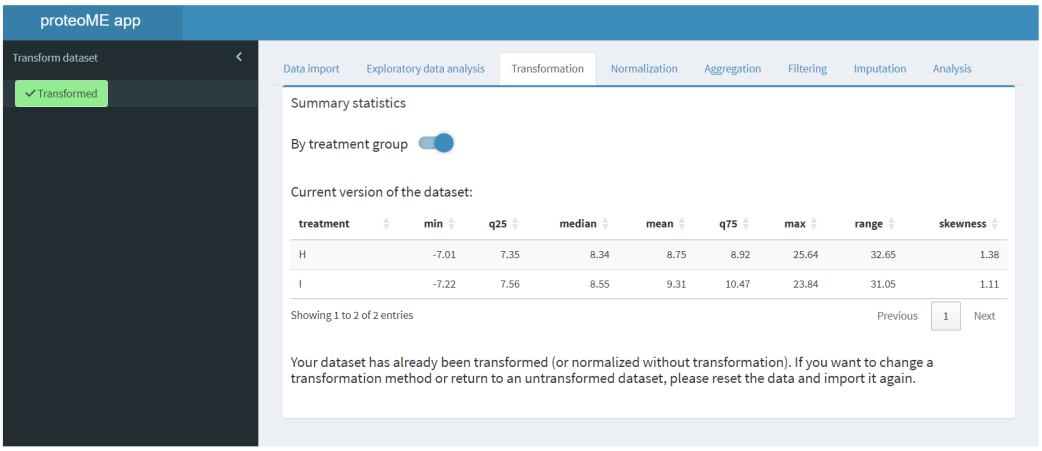
Pokud si uživatel akci rozmyslí, může kliknout na tlačítko *No, I'll think about it.*, vrátí se zpět a datovou sadu si může na záložce *Data import* stáhnout. Pokud je o transformování přesvědčen, klikne na tlačítko *Yes, transform it!*, čímž se celá datová matice s abundancemi transformuje dle zvolené funkce a následné vyskakovací okno potvrdí uživateli, že akce úspěšně proběhla. Pozná to však i přímo na aktuální záložce. Postranní panel se změní,



Obrázek 3.13: Screenshot aplikace *proteoME* (Transformation) – potvrzovací okno

po transformaci obsahuje pouze zelené tlačítko ✓ *Transformed*[9]. V těle aplikace je pouze horní tabulka číselných charakteristik počítaná z aktuální (nyní již transformované) datové sady, pod ní je informace, proč se již nezobrazuje dolní tabulka a jak případně postupovat, kdyby chtěl uživatel transformovat data jinak – vizte Obrázek 3.14. Přetrvávající tabulka se spočítanými hodnotami stále reaguje na aktuální podobu datové sady, lze se k ní tedy vrátit např. i po normalizaci a zkontrolovat, jak se číselné charakteristiky abundancí změnily. V našem ukázkovém příkladu máme nyní datovou matici s abundancemi transformovanou pomocí funkce $log(x)$.

[9]Na to sice lze kliknout, ale pouze uživateli oznámí, že již data transformoval, a uvede, jak postupovat, pokud by chtěl např. změnit metodu transformace – cestou je resetovat data na záložce *Data import* a nahrát je znovu.



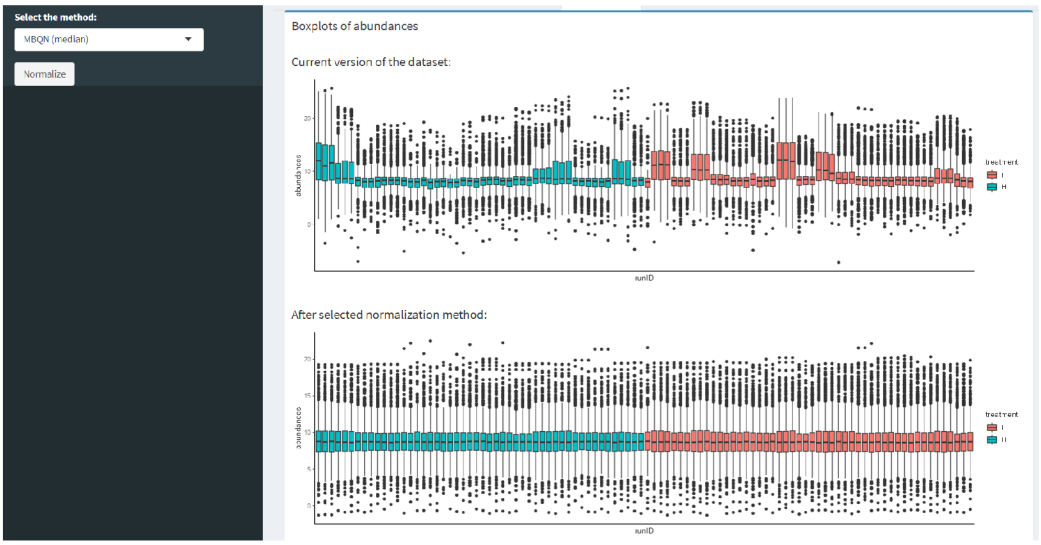
Obrázek 3.14: Screenshot aplikace *proteoME* (Transformation) – stav po transformaci dat

3.4 Normalization

Následující záložka určená k normalizaci dat je v používání a funkcích velmi podobná té předchozí. Postranní menu obsahuje rozbalovací seznam metod normalizace, které má uživatel k dispozici. Dostupné metody odpovídají těm popsaným v části 1.2.3 – tedy mediánová normalizace, kvantilová normalizace a metoda MBQN v obou verzích (tedy *median-balanced* i *mean-balanced*). Pod touto volbou je tlačítko *Normalize*, které otevře potvrzovací okno podobné tomu, které bylo popsáno u transformace.

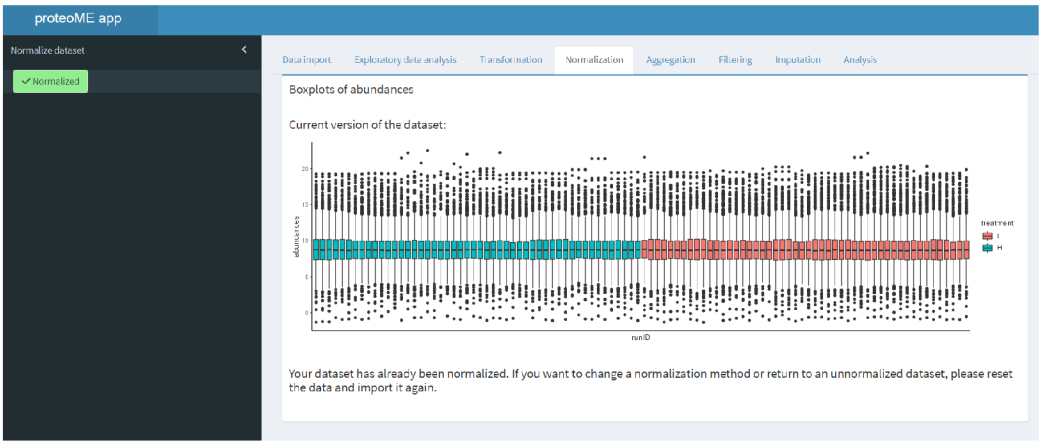
A na základě čeho se rozhodovat, kterou metodu použít a zda normalizovat? V těle aplikace je tentokrát dvojice boxplotů, které nesou stejnou informaci jako boxplot na záložce *Exploratory data analysis*[10]. Ten horní je vykreslován na základě aktuální verze datové sady, dolní boxplot reaguje na volbu metody normalizace a pomáhá tak uživateli alespoň vizuálně zhodnotit, která z metod bude pro jeho datovou sadu nejvhodnější. Situaci se

[10] Tyto boxploty však nedisponují interaktivními infoboxy a není u nich možnost stáhnutí – jejich účelem je pouze vizuální porovnání podoby dat před a po normalizaci – vizte dále.



Obrázek 3.15: Screenshot aplikace *proteoME* (Normalization) – srovnání boxplotů před a po normalizaci

zvolenou metodou MBQN (median) lze vidět na Obrázku 3.15. Po stisknutí tlačítka *Normalize* při zvolené metodě a potvrzení akce vidí uživatel potvrzovací okno s informací o úspěšné normalizaci dat – ta jsou tedy nyní v rámci našeho příkladu i normalizovaná pomocí zvolené metody. Na postranním panelu je zelené tlačítko ✓*Normalized* se stejnou funkcí jako u předchozí záložky, v těle aplikace zůstává pouze jeden graf s boxploty pro aktuální (již normalizovanou) datovou sadu a pod ním instrukce pro návrat před tento proces a volbu jiné metody normalizace – vizte Obrázek 3.16. Tento boxplot si uživatel může vykreslit (a příslušně přizpůsobit) na záložce *Exploratory data analysis*. Normalizovanou datovou sadu s abundancemi (podobně jako po transformaci) lze stáhnout na záložce *Data import*.



Obrázek 3.16: Screenshot aplikace *proteoME* (Normalization) – stav po normalizaci dat

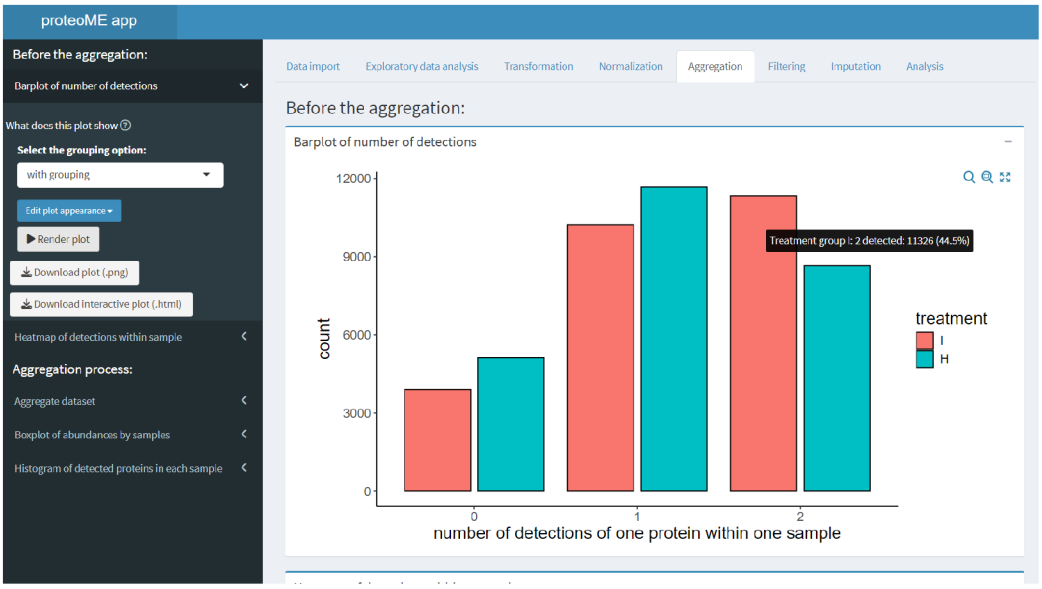
3.5 Aggregation

Pokud uživatel prošel předchozími záložkami, má data na úrovni jednotlivých běhů připravena. Analýza však probíhá na úrovni vzorků – musíme je tedy agregovat[11]. Právě k tomu slouží záložka *Aggregation*. Podrobně je proces agregace popsán v části 1.2.4, aplikace *proteoME* agregaci provádí totožně (a používá i stejné vizualizace pro zjednodušení volby parametrů agregace). Pro hlubší pochopení celého procesu i toho, co grafy ukazují a k čemu slouží, doporučuji vrátit se ke zmíněné pasáži textu. Zde se podíváme na praktické používání této záložky v aplikaci.

Postranní panel je rozdělen na část *Before the aggregation* (tedy před agregací) a *Aggregation process* (tedy proces agregace). V první části může uživatel vykreslit barplot počtu detekcí proteinu v rámci vzorku (odpovídající Obrázku 1.9 – volí si zohlednění skupiny léčby) a heatmapu s počty detekcí v rámci vzorku spolu se souhrnnými anotačními sloupci pro jednotlivé počty detekcí a tabulku s mediánem počtu vzorků s danou četností detekce (odpoví-

[11] Pokud uživatel data neagreguje, nemůže provést na dalších záložkách žádnou akci.

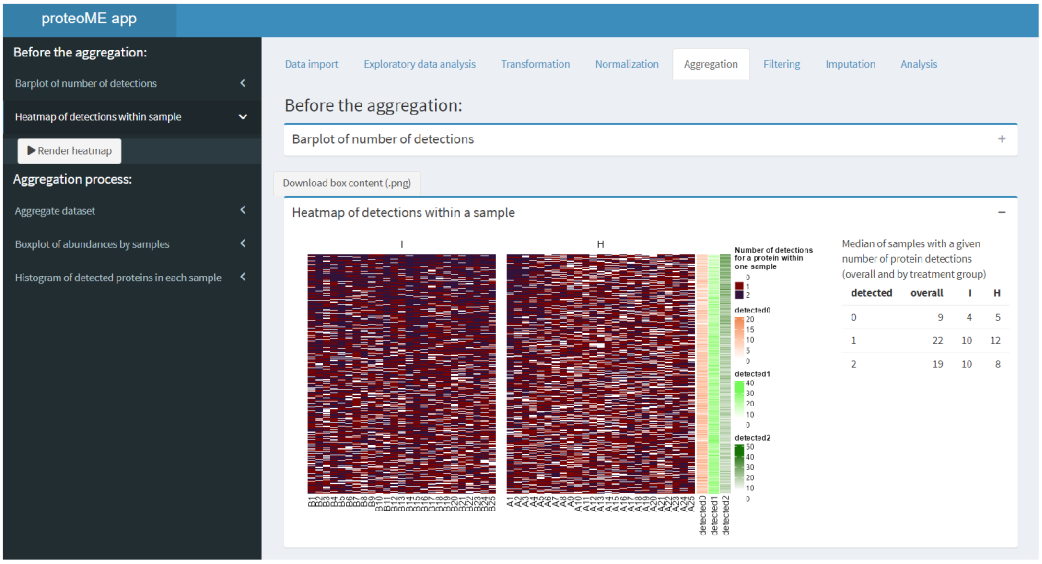
dající Obrázku 1.10). U barplotu je k dispozici úprava vzhledu představená již u předchozích grafů, v části specifické pro daný typ grafu lze upravit barvu výplně sloupců (pouze u varianty bez zohlednění skupiny) a barvu okraje sloupců. Interaktivní infobox nese informaci o absolutním i relativním počtu výskytu jednotlivých počtů detekcí (dostupné jak bez zohlednění skupiny, tak při zahrnutí tohoto faktoru, přičemž relativní počet se počítá v rámci každé skupiny zvlášť). Ukázku obrazovky s tímto grafem lze vidět na Obrázku 3.17. Opět je k dispozici stáhnutí statické i interaktivní verze vizualizace.



Obrázek 3.17: Screenshot aplikace *proteoME* (Aggregation) – barplot počtu detekcí proteinu v rámci vzorku

Heatmapa detekcí s tabulkou mediánů je mnohem komplexnější vizualizací, která v aplikaci nemá žádné možnosti úprav – v nabídce menu je pouze tlačítko *Render heatmap*, které spustí proces vykreslování. Heatmapu lze následně zobrazit v těle aplikace v okně *Heatmap of detections within a sample* (vizte Obrázek 3.18). Barevné škály anotačních sloupců jsou generované au-

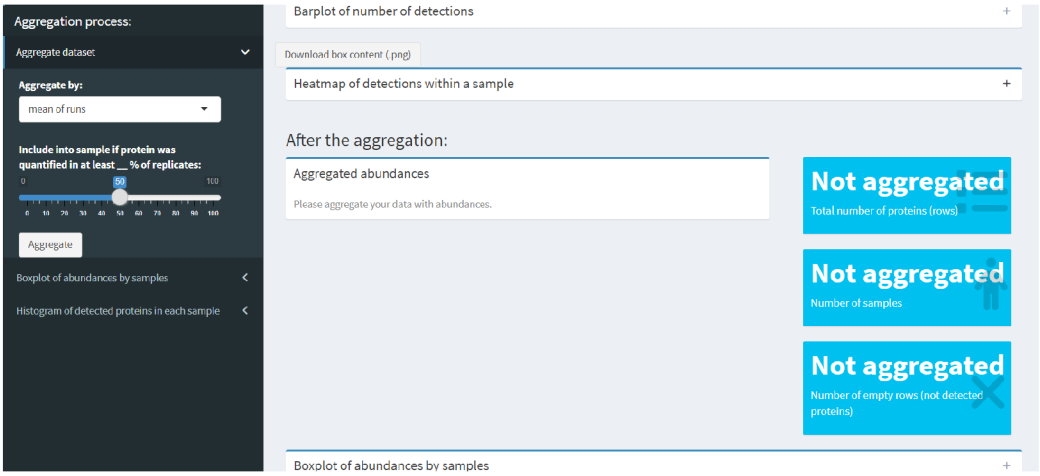
tomaticky a diskrétní barvy heatmapy nastaveny fixně – bez možnosti volby ze strany uživatele. Pokud uživatel není spokojen s barevnými škálami anotačních sloupců, může opakovaně generovat heatmapu až do nalezení uspokojivé kombinace. Jedinou dostupnou funkcí je stáhnutí obrázku, ve kterém jsou oba vytvořené objekty – tedy heatmapa i tabulka s mediány.



Obrázek 3.18: Screenshot aplikace *proteoME* (Aggregation) – heatmapa detekcí v rámci vzorku

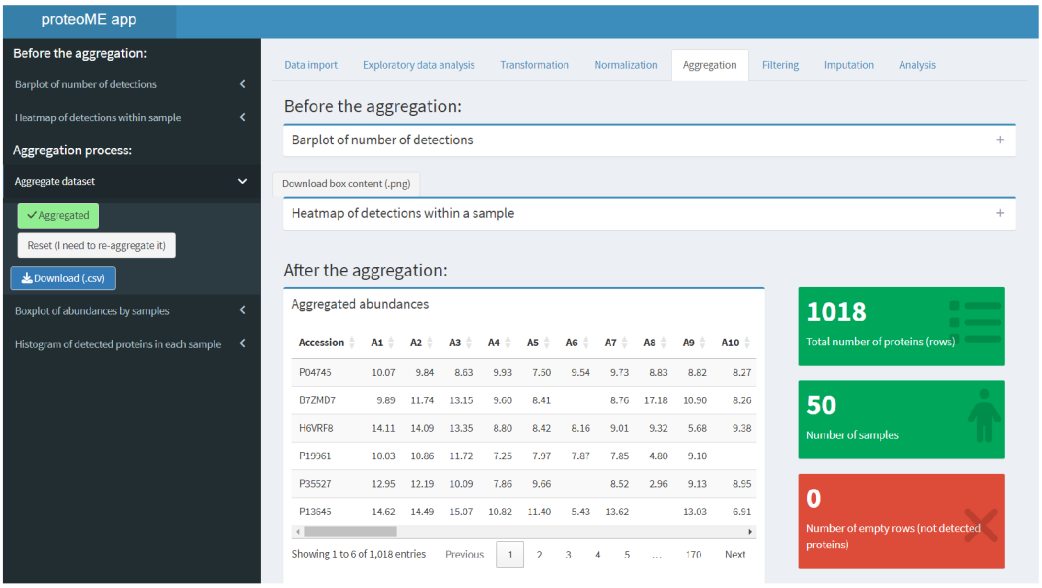
Tyto grafy tedy mají usnadnit rozhodování, jak nastavit parametr k agregování – číslo n, které stanoví hranici, v kolika procentech souborů (replikátů) daného vzorku musel být protein kvantifikován, aby byly jeho abundance zahrnuty do agregačního procesu. Pokud tuto podmínku protein v daném vzorku splňuje, jeho výsledná abundance ve vzorku se spočítá jako průměr či medián abundancí kvantifikovaných replikátů – o tom rozhoduje uživatel v postranním menu v části *Aggregate dataset*, kde si volí i hodnotu parametru n – vizte Obrázek 3.19, kde chceme ukázková data agregovat pomocí průměru (což u dvou replikátů pro každý vzorek vychází stejně jako medián),

a to v případě, že byl protein kvantifikován alespoň v 50 % případů (tedy alespoň v jednom ze dvou replikátů). Pak už jen uživateli stačí kliknout na



Obrázek 3.19: Screenshot aplikace *proteoME* (Aggregation) – volba parametrů agregace

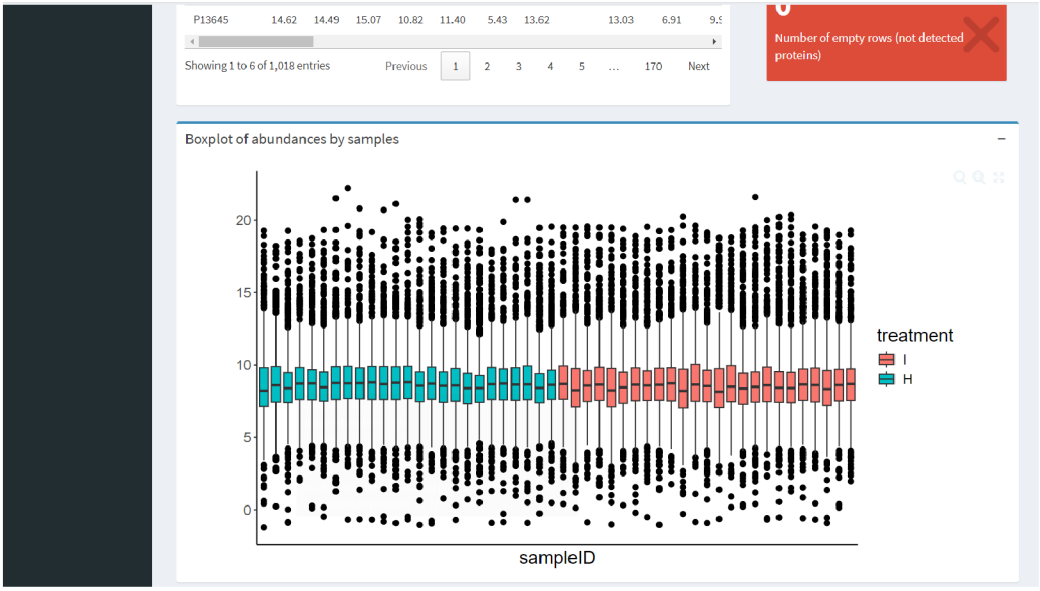
tlačítko *Aggregate* a ve vyskakovacím okně tuto volbu potvrdit. Agregace dat nijak neovlivní datové sady na úrovni souborů, s kterými jsme pracovali na předchozích záložkách. Po úspěšné agregaci se uživateli v těle aplikace zobrazí tabulka s abundancemi podobná té, která je k dispozici na úvodní záložce – tentokrát už s přepočítanými daty na úroveň vzorků. Stejně tak se vyplní údaje do připravených infoboxů. Kromě počtu proteinů a počtu vzorků představených již na záložce *Data import* je zde box s počtem prázdných řádků – tedy kolik proteinů nebylo kvantifikováno v žádném vzorku (ať už od úvodního importu dat, nebo až vlivem agregace). Tabulka i infoboxy také reagují na všechny další akce (filtrace, imputace), lze se k nim tedy i nadále vracet. Stav obrazovky po agregaci lze vidět na Obrázku 3.20. Agregovanou datovou sadu lze stáhnout tlačítkem *Download (.csv)*, které se objevilo v menu místo volby parametrů agregace. Tamtéž je nyní i tlačítko potvrzující proběhlou agregaci, navíc je k dispozici tlačítko pro reset procesu agregace. To je



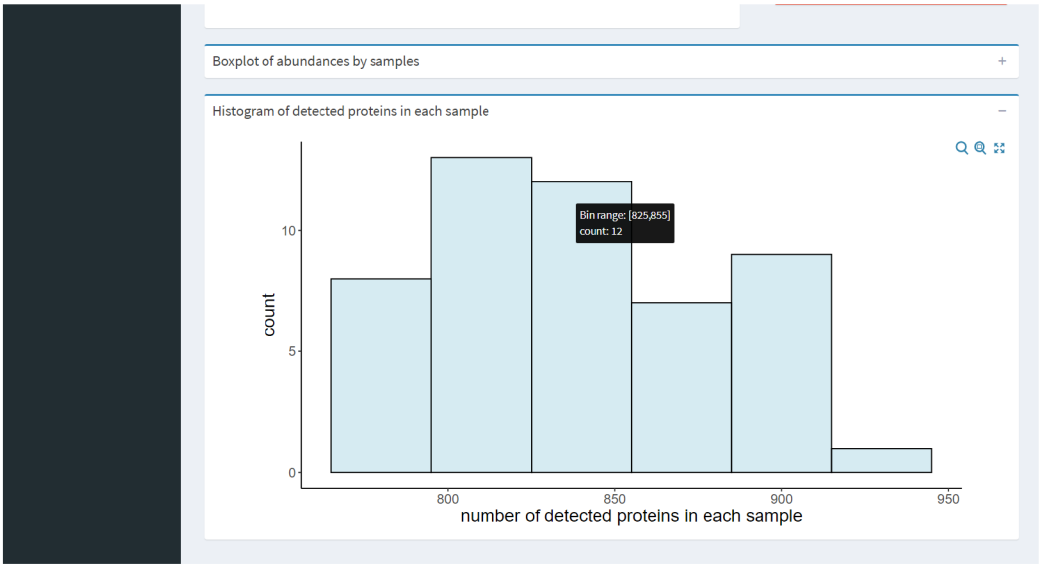
Obrázek 3.20: Screenshot aplikace *proteoME* (Aggregation) – stav po agregaci

velmi důležité pro další užívání aplikace. Pokud totiž bude chtít uživatel po provedení filtrace či imputace změnit použitou metodu či parametry[12], musí použít toto tlačítko a data reagregovat. Po agregaci lze na data na úrovni vzorků nahlédnout pomocí již „známé“ dvojice grafů, která byla představena v rámci záložky *Exploratory data analysis* – boxploty abundancí a histogram počtu detekovaných proteinů. Jedinou funkční změnou je nedostupná volba transformace zobrazovaných dat u boxplotů – po agregaci totiž uživatel již nemůže data nijak transformovat. Zbylé možnosti úprav vzhledu i funkce u obou grafů zůstávají i na úrovni vzorků. Vykreslené v aplikaci je můžete vidět na Obrázku 3.21, resp. 3.22. Znovu vygenerovat i stáhnout lze tyto grafy i po případné filtraci a imputaci na dalších záložkách.

[12]Ostatně i při potřebě znovu agregovat data s jinými parametry.



Obrázek 3.21: Screenshot aplikace *proteoME* (Aggregation) – boxplot abundancí po agregaci



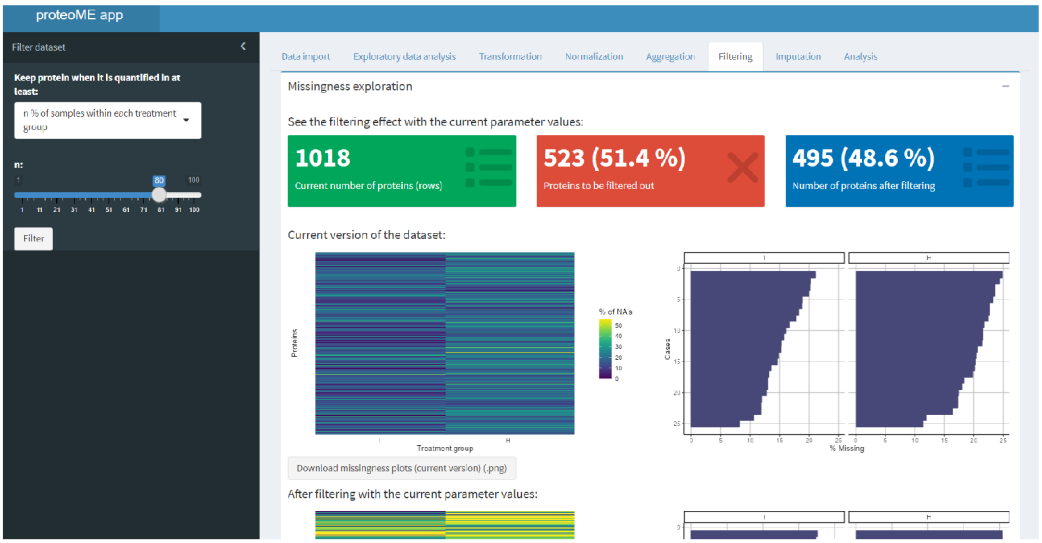
Obrázek 3.22: Screenshot aplikace *proteoME* (Aggregation) – histogram počtu detekovaných proteinů po agregaci

3.6 Filtering

Jakmile jsou data agregována, zpřístupní se uživateli nástroje filtrace a imputace. Redukce řádků s příliš mnoha chybějícími hodnotami na záložce *Filtering* nabízí stejné možnosti, které byly představeny v části 1.2.5. Nabídka menu tvoří dvě volby. První doplňuje metodický pokyn „Ponech v datech protein, pokud byl kvantifikován alespoň v:“ jednou z těchto možností:

- n % všech vzorků
- n % vzorků v rámci alespoň jedné skupiny léčby
- n % vzorků v rámci všech skupin léčby

Druhou volbou je právě hodnota parametru n. V hlavní části aplikace je na této záložce před provedením filtrace trojice infoboxů. První z nich obsahuje pouze aktuální počet řádků (proteinů) v datové matici abundancí, další dva jsou interaktivní a reagují na zvolenou metodu a parametr filtrace – to aby se uživatel mohl lépe rozhodnout, za jakých podmínek filtrovat. V těchto interaktivních boxech se totiž přepočítávají počty proteinů, které by za současného nastavení byly odfiltrovány z dat pryč, resp. počty proteinů, které v datech při těchto parametrech zůstanou (obě hodnoty jsou zde i v procentech). Pro ještě větší přehled uživatele jsou pod těmito infoboxy dva páry grafů, které odpovídají Obrázkům 1.11 a 1.12. Horní dvojice je statická, počítaná z aktuální verze datové sady. Spodní dvojice se však interaktivně mění dle navolených parametrů filtrace. Grafy lze takto po dvojicích stáhnout ve formátu *.png* stisknutím příslušného tlačítka umístěného pod nimi. Část obrazovky před filtrací si můžete prohlédnout na Obrázku 3.23, kde se chystáme ponechat v datech pouze ty řádky, v nichž byl protein kvantifikován alespoň v 80 % vzorků v rámci každé skupiny.



Obrázek 3.23: Screenshot aplikace *proteoME* (Filtering) – stav před filtrací

Obrázek 3.24: Screenshot aplikace *proteoME* (Filtering) – stav po filtraci

Stisknutím tlačítka *Filter* na postranním panelu (a potvrzením této volby v objevivším se okně) uživatel provede filtraci řádků při zvolených parametrech, v nabídce menu zůstane pouze tlačítko potvrzující, že filtrace proběhla, navíc se objeví tlačítko pro stáhnutí současné verze agregované a filtrované

datové sady. Totožná data by si uživatel mohl stáhnout i na předchozí záložce *Aggregation* (kam se proces filtrace samozřejmě propsal) – zde však jde spíše o komfortní prvek, který uživateli umožní nepřeklikávat na předchozí záložky a plynule pokračovat směrem vpřed. V těle aplikace zůstává pouze infobox se současným počtem řádků a dvojice vizualizací chybějících hodnot taktéž počítaná ze současných – již filtrovaných – dat.

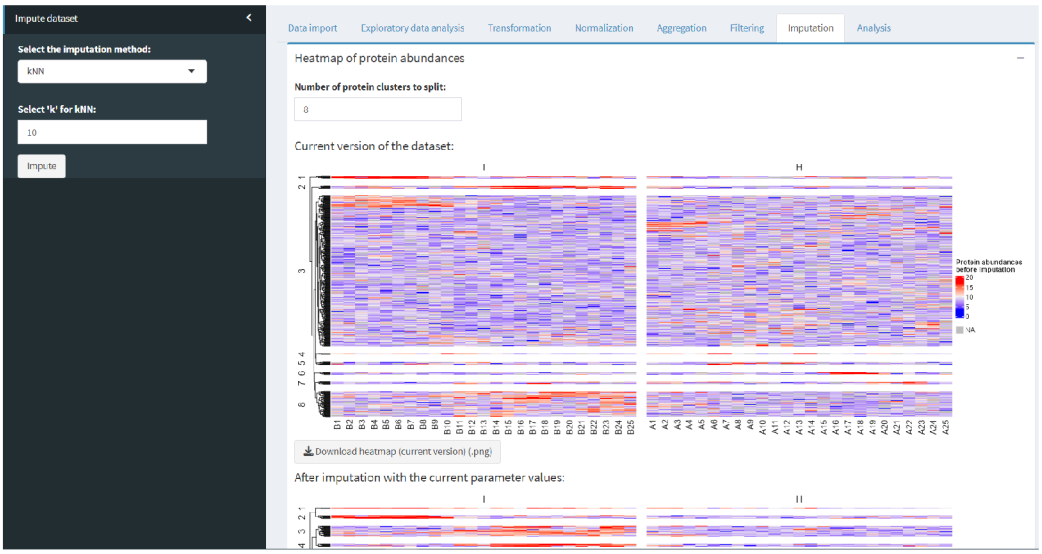
3.7 Imputation

Poslední záložka pro úpravu dat před samotnou analýzou slouží k nahrazení chybějících hodnot. Uživatel má k dispozici 3 metody imputace – jde o *SampMin*, *kNN* a *RF*, které byly představeny v části 1.2.6. Kromě volby metody se v postranním menu objevují možnosti další úpravy parametrů dynamicky se měnící dle zvolené metody. U *kNN* uživatel nastavuje hodnotu parametru k („počet sousedů“), u *RF* je zase možnost změnit počet rozhodovacích stromů vytvořených v rámci každé iterace procesu náhodných lesů. U této metody je navíc k dispozici možnost „zasadit semínko“[13] pro zajištění dosažení stejných výsledků, pokud by se uživatel o imputaci touto metodou pokoušel opakovaně.

V těle aplikace je dvojice heatmap s hodnotami abundancí – horní (statická) vykreslovaná na základě aktuální podoby agregované (příp. i filtrované) datové sady, spodní (dynamicky se měnící) ukazující případnou podobu datové sady při imputaci aktuálně zvolenou metodou a se zadanými parametry. Nad heatmapami je ještě číselný vstup pro volbu počtu shluků – obě[14] he-

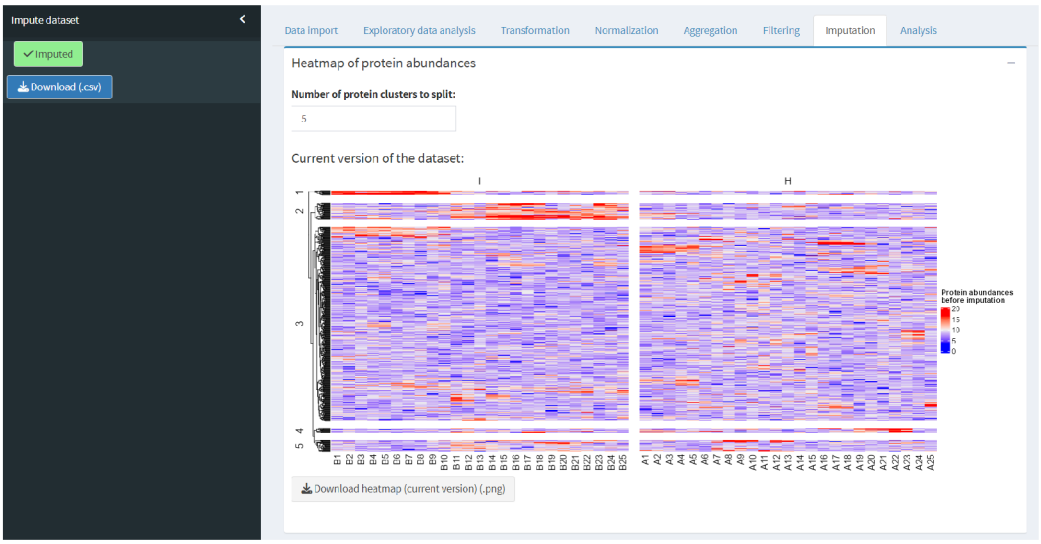
[13] Tímto pojmem se myslí nastartování procesu generování pseudonáhodných čísel fixním způsobem – pro dané „semínko“ získáme v R při zopakování stejné sady úkonů závislých na náhodných číslech totožné výsledky. Bez zasazení semínka bychom vždy dostali výsledky odlišné.

[14] První heatmapa obsahující i chybějící hodnoty provádí shlukování řádků pouze tehdy, je-li k dispozici dostatečné množství dat.



Obrázek 3.25: Screenshot aplikace *proteoME* (Imputation) – stav před imputací

atmapy totiž provádí shlukování řádků tak, jak bylo popsáno v části 1.2.6. Počet shluků představuje jen vizuální prvek a nemá žádný vliv na doplnění chybějících hodnot. V případě zájmu lze heatmapy stáhnout příslušným tlačítkem umístěným pod každým z těchto grafů. Náhled obrazovky před imputací se zvolenou metodou *kNN* a defaultním počtem sousedů (10) můžete vidět na Obrázku 3.25 – tímto způsobem imputujeme data v naší ukázkové analýze, a to stisknutím tlačítka *Impute* a následným potvrzením ve vyskakovacím okně. Po nahrazení *NA* hodnot je postranní menu prakticky totožné s předchozí záložkou (co do vzhledu a funkce), v těle aplikace zbyla pouze heatmapa s aktuální verzí datové sady, která je již imputovaná. Stále lze upravovat počet shluků pro případný export grafu – vizte Obrázek 3.26.



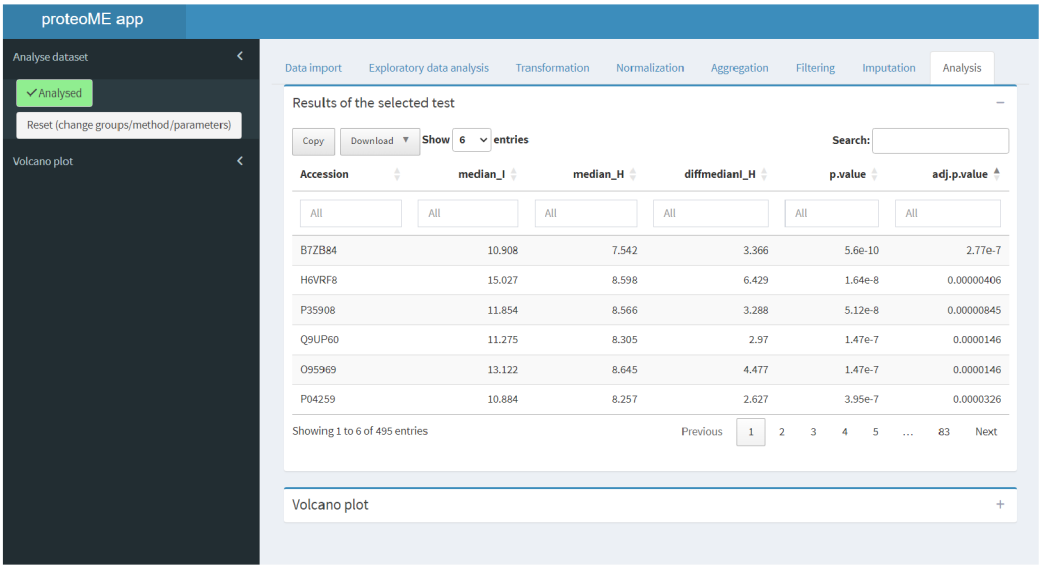
Obrázek 3.26: Screenshot aplikace *proteoME* (Imputation) – stav po imputaci

3.8 Analysis

Nyní jsme tedy připraveni na to hlavní – můžeme se přesunout k poslední záložce aplikace, která slouží k analýze datové sady. Obsah záložky znovu kopíruje popsaný proces analýzy z části 1.2.7 (včetně tabelárního výstupu a volcano plotu). Pojďme se tedy podívat na její praktické používání. Menu na postranním panelu je rozděleno na dvě části – první slouží samotnému procesu analýzy, druhá zase pro vykreslení a úpravu parametrů volcano plotu. U části kontrolující analýzu dat má uživatel nejprve k dispozici numerický vstup udávající počet skupin léčby, které chce porovnávat. Minimální počet skupin je dva, maximální je dán celkovým počtem skupin v datech. Pokud se zvolený a maximální počet nerovnají (situace, kdy chceme do analýzy zahrnout pouze některé skupiny léčby), objeví se pod tímto vstupem okno pro výběr skupin[15]. Dle celkového počtu porovnávaných skupin se také dynamicky mění nabídka testů k provedení analýzy – pro dvouvýběrové srovnání

[15]Počet zvolených skupin musí odpovídat číslu na vstupu výše.

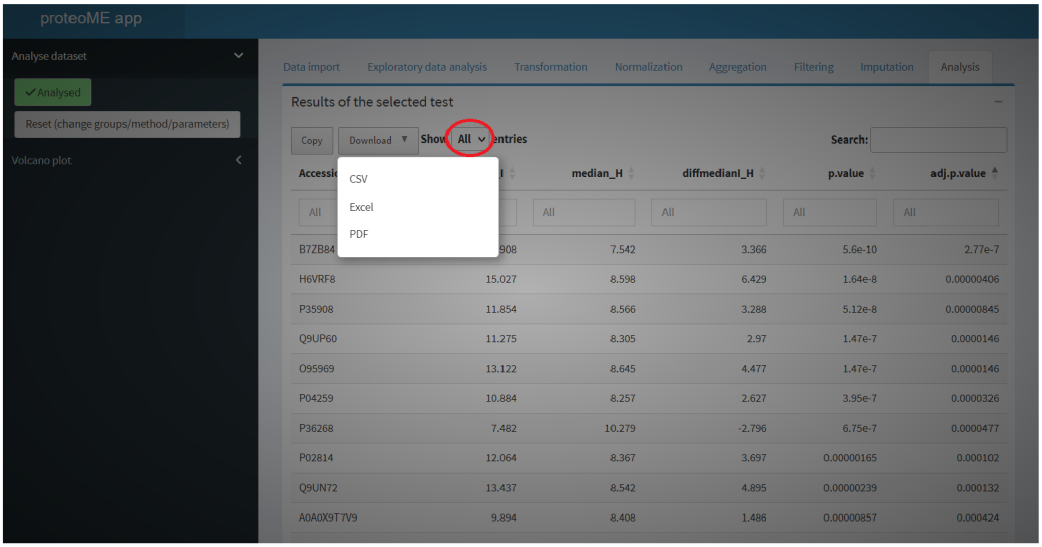
jsou k dispozici t-test a Wilcoxonův test, pro testování více skupin volíme mezi ANOVA a Kruskalovým–Wallisovým testem. U těchto testů se navíc uživateli zpřístupní volba hraniční p-hodnoty pro označení dvojice skupin za významně se lišící v distribuci v rámci mnohonásobného porovnávání (Tukey, Dunn), které po těchto testech následuje. Po zvolení vhodného testu stačí stisknout tlačítko *Analyse* – v ukázkovém příkladu tak učiníme se dvěma skupinami (víc jich v datech nemáme) a Wilcoxonovým testem. V těle aplikace se objeví tabulka s výsledky – vizte Obrázek 3.27.



Obrázek 3.27: Screenshot aplikace *proteoME* (Analysis) – tabulka s výsledky analýzy

Pro každou testovanou skupinu se spočítají mediány abundancí každého proteinu, jejich rozdíl, p-hodnota testu a její adjustace kvůli množství testovaných hypotéz. U vícevýběrových testů přibudou ještě sloupce s významně se lišícími dvojicemi a p-hodnotami mnohonásobného porovnávání. Všechny sloupce lze nejen řadit (např. na Obrázku 3.27 už je tabulka seřazená vzestupně dle adjustované p-hodnoty), ale i filtrovat (k tomu slouží okno pod

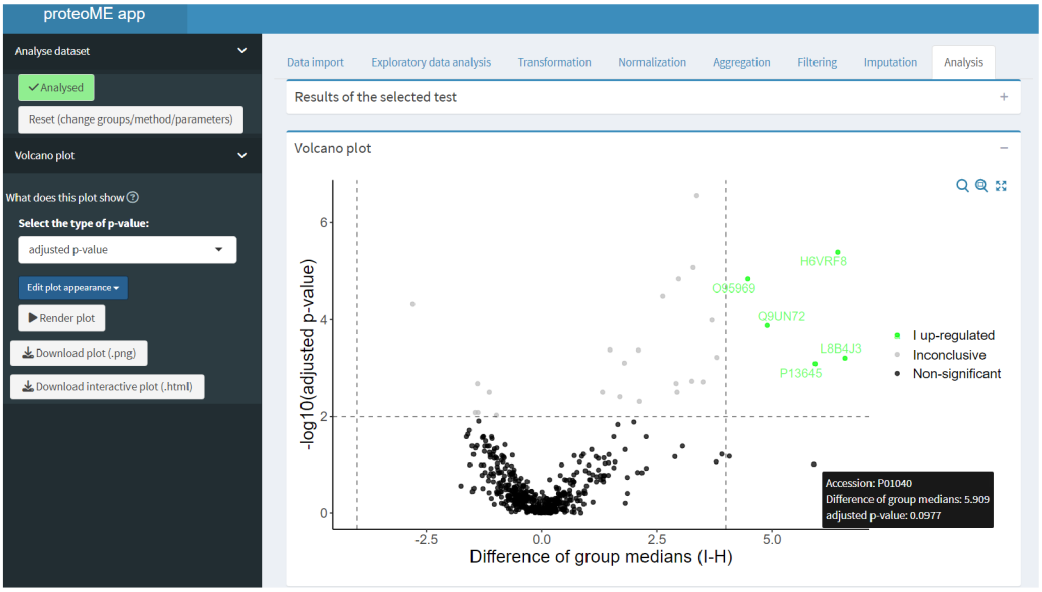
názvem sloupce). Uživatel si tak může v tabulce nechat např. pouze ty proteiny, u nichž p-hodnota vyšla menší než uživatelem stanovená hranice, příp. si najít konkrétní proteiny dle jejich idetifikátoru. Uživatele jistě potěší také široké spektrum možností exportu této tabulky. K dispozici je jednak prosté zkopírování do schránky (tlačítko *Copy* nad tabulkou) s tím, že si tabulku může uživatel vložit, kamkoliv potřebuje, jednak má možnost tabulku stáhnout v jednom ze tří formátů (*.csv*, *.xlsx*, *.pdf*) – ty jsou schovány v rozbalovací nabídce *Download* hned vedle tlačítka *Copy*. Jen je třeba dát pozor na to, že exportovány jsou vždy jen ty řádky, které jsou aktuálně v tabulce na dané stránce (exportní proces tedy ignoruje data na dalších stránkách tabulky). Pokud chce uživatel exportovat všechna data naráz, musí si je také všechna zobrazit nastavením *Show **All** entries* – vizte Obrázek 3.28.



Obrázek 3.28: Screenshot aplikace *proteoME* (Analysis) – možnosti exportu tabulky s výsledky

Přejděme nyní k druhému způsobu prezentace výsledků – volcano plotu. Jeho význam byl podrobně rozberán v části 1.2.7 a výstupy aplikace zde od-

povídají Obrázku 1.15. Zde si ukážeme, jak s grafem v aplikaci pracovat. Volcano plot lze vykreslit až po otestování datové sady (a získání výše představené tabulky). Postranní menu disponuje podobnými možnostmi, jako tomu je u dříve představených grafů. Rozbalovací seznam umožňuje uživateli po dvouvýběrovém testu zvolit mezi původní a adjustovanou p-hodnotou testu, která se vykresluje na osu y. Po testování pomocí ANOVA či Kruskalově-Wallisově testu je na ose y fixně p-hodnota odpovídající výsledku mnohonásobného porovnávání pro vybranou dvojici skupin, kterou uživatel volí z rozbalovacího seznamu namísto typu p-hodnoty. V obou případech pak stačí pro vykreslení grafu stisknout tlačítko *Render plot* – výsledek v rámci ukázkového příkladu lze vidět na Obrázku 3.29.



Obrázek 3.29: Screenshot aplikace *proteoME* (Analysis) – volcano plot

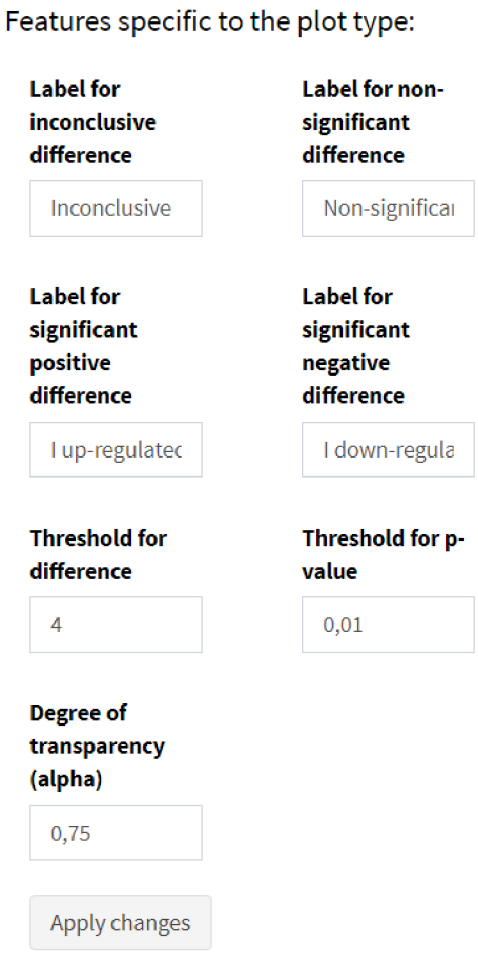
Plocha grafu je rozdělena na 4 části – výsledky nesignifikantní, nejednoznačné a významné s dominantními abundancemi jedné či druhé skupiny. Pro označení těchto skupin se defaultně použijí pojmy *Non-significant*, *In-*

conclusive a *'Group' up-regulated* (pravá horní část grafu) spolu s *'Group' down-regulated* (levá horní část grafu), kde místo *'Group'* figuruje označení referenční skupiny (resp. té, která je v popisku osy x u rozdílu mediánů zmíněná na prvním místě[16]). Body se následně rozřadí do kategorií podle toho, v které části grafu se nacházejí, barevně se odliší a u významných výsledků se k bodu připojí i identifikátor proteinu. V momentu vykreslení grafu se navíc k výsledkové tabulce připojí sloupec právě s označením této kategorie, uživatel si tak může pro export tabulky např. vyfiltrovat pouze ty proteiny, které se nacházejí v *up-regulated* či *down-regulated* kategorii. Body volcano plotu jsou navíc interaktivní – při přejetí kurzorem na daný bod se objeví infobox s ID proteinu, rozdílem skupinových mediánů a p-hodnotou z osy y (ať už se tam vykresluje kterákoliv z možných variant), ovšem transformovanou zpět na původní škálu[17].

Jak se ale nastavují parametry rozdělující volcano plot na tyto 4 části? K tomu slouží druhá polovina rozbalovacího okna pro úpravu vzhledu grafu (tlačítko *Edit plot appearance*), která má pro volcano plot podobu, kterou můžete vidět na Obrázku 3.30. Kromě názvů výše zmíněných čtyř kategorií lze přenastavit také výchozí hodnoty pro vymezení těchto kategorií – vstup *Threshold for difference* představuje absolutní hodnotu x-ových souřadnic svislých čar (tedy o kolik chceme, aby se skupinové mediány lišily, aby to pro nás bylo jednoznačné), *Threshold for p-value* zase vymezí výšku horizontální čáry pro vytyčení hranice významných výsledků testování. Uživatel navíc nemusí p-hodnotu transformovat, zadává skutečně požadovanou hranici významnosti (např. 0,01, která je výchozí) a vodorovná čára se v grafu sama vykreslí ve

[16] Toto pořadí lze ovlivnit na záložce *Data import* prostřednictvím tlačítka pro nastavení faktorů. Volcano plot se vždy vykresluje z pohledu té skupiny, která je v pořadí úrovní faktoru první/dříve než druhá ze zobrazovaných skupin.

[17] Připomeňme, že pro vykreslení osy y je p-hodnota p transformována pomocí funkce $p^\star = -\log_{10}(p)$, jak bylo zmíněno v části 1.2.7.



Obrázek 3.30: Screenshot aplikace *proteoME* (Analysis) – možnosti úpravy vzhledu a parametrů volcano plotu

výšce odpovídající transformované p-hodnotě (zde tedy $-\log_{10}(0,01)$). Dále lze upravit i průhlednost bodů. Graf lze stáhnout ve statické i interaktivní podobě.

Tímto tedy končíme ukázkovou analýzu proteomické datové sady, u které jsme zvládli projít všechny záložky a provést všechny kroky, které jsou v rámci aplikace *proteoME* dostupné. Pořadí záložek představuje doporučené pořadí kroků během analýzy – uživatel se jej sice držet nemusí, avšak provádění

kroků v jiném pořadí nedává v kontextu aplikace příliš smysl (kromě návratů na předchozí záložky kvůli exportu dat, grafů apod.). Aplikace je v současné verzi plně funkčním nástrojem, který již teď může velmi usnadnit datovou analýzu v tomto oboru, avšak zároveň jde o platformu připravenou pro snadnou implementaci rozšiřujících funkcí a vylepšení, které může přijít jak ze strany autora, tak ze strany široké veřejnosti (díky umístění zdrojového kódu na GitHubu).

Závěr

Cílem práce bylo vytvořit aplikaci pro preprocessing, imputaci a analýzu proteomických dat. Nutnou podmínkou k používání takového nástroje je porozumění základním pojmům specifickým pro tento obor a pochopení jednotlivých kroků při práci s těmito daty. K tomu slouží první kapitola, která navíc zmiňuje konkrétní metody, které se při zpracování proteomických dat dají použít, a nabízí i ukázku možných grafických výstupů.

Ve druhé kapitole byl představen princip tvorby Shiny aplikací ve formě R balíčku, a to především za pomoci balíčku `golem`. Uvedli jsme, proč je lepší aplikaci stavět z více menších celků namísto jednoho či dvou rozsáhlých souborů. Využívání těchto stavebních jednotek (shiny modulů) bylo v této kapitole rovněž podrobně rozebráno. Nebyla zanedbána ani specifika dokumentace takové aplikace a její publikace – v závěru druhé kapitoly jsou uvedeny dvě cesty, jak aplikaci zpřístupnit veřejnosti. Tato část textu nejen že představuje danou problematiku českým čtenářům (kteří se s ní kvůli malému počtu zdrojů dosud nemuseli setkat), ale využívá i autorovy zkušenosti nabyté při tvorbě aplikace jako R balíčku dle základního zdroje (od autorů balíčku `golem`) s cílem ještě více umést cestu případným zájemcům a ušetřit jim hodiny času díky praktickým radám, které by autor při práci na tomto projektu velice ocenil.

Pro tuto diplomovou práci je stěžejní kapitola třetí, která popisuje úspěšně

naplněný cíl – novou aplikaci *proteoME*, která je ve formě R balíčku dostupná k instalaci z platformy GitHub. Aplikace je zde podrobně popsána a kapitola tak provádí čtenáře celým procesem zpracování proteomické datové sady prostřednictvím *proteoME* od importu dat v potřebném formátu až po výstupy analýzy, které mají dále sloužit k lepšímu zacílení výzkumu léčby různých onemocnění. Autor textu pevně doufá, že *proteoME* se stane vyhledávaným nástrojem v komunitě vědců zabývajících se proteomikou, který usnadní jejich každodenní práci. Aplikace je navíc díky své formě (v souladu s druhou kapitolou) připravena na postupné rozšiřování a nabízení nových funkcí, které by ji mohly učinit ještě atraktivnější ve srovnání s podobně zacílenými nástroji.

Pro mnohé studenty znamená diplomová práce úspěšný závěr jejich studia s tím, že téma často opustí a dál se k němu nevrací. Tato práce by však naopak měla tvořit pevné základy pro budoucí rozvoj problematiky, ke kterému chce autor dále přispívat a vylepšovat aplikaci tak, aby vědcům ještě více usnadnila postupnou cestu za rozšiřováním lidského poznání.

Seznam literatury

[1] BENJAMINI, Y., HOCHBERG, Y.: *Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing*. Journal of the Royal Statistical Society, Series B (Methodological), 57, s. 289-300, 1995. Dostupné z: https://doi.org/10.1111/j.2517-6161.1995.tb02031.x.

[2] BOLSTAD, B. M., IRIZARRY, R. A., ÅSTRAND, M., SPEED, T. P.: *A comparison of normalization methods for high density oligonucleotide array data based on variance and bias*. Bioinformatics, 19(2), s. 185-193, 2003. Dostupné z: https://doi.org/10.1093/bioinformatics/19.2.185.

[3] BROMBACHER, E., SCHAD, A., KREUTZ, C.: *Tail-Robust Quantile Normalization*. Proteomics, 20: 2000068, 2020. Dostupné z: https://doi.org/10.1002/pmic.202000068.

[4] CHMELÍK, J.: *Proteomic Guide* [online]. Chemické listy, 99(12), s. 883–885, 2006 [cit. 2024-01-23]. Dostupné z: http://www.chemicke-listy.cz/ojs3/index.php/chemicke-listy/article/view/1963.

[5] CONOVER, W. J.: *Practical Nonparametric Statistics*. 3rd ed. New York: A John Wiley & Sons, Inc., 1999. ISBN 978-0471160687.

[6] DRESSLER, F. F., BRÄGELMANN, J., REISCHL, M., PERNER, S.: *Normics: Proteomic Normalization by Variance and Data-Inherent Correlation Structure*. Mol Cell Proteomics, 21(9): 100269, 2022. PMID 35853575. PMCID PMC9450154. Dostupné z: https://doi.org/10.1016/j.mcpro.2022.100269.

[7] DUNN, O. J.: *Multiple Comparisons Using Rank Sums*. Technometrics, 6(3), s. 241-252, 1964. Dostupné z: https://doi.org/10.1080/00401706.1964.10490181.

[8] FAY, C., ROCHETTE, S., GUYADER, V., GIRARD, C.: *Engineering Production-Grade Shiny Apps*. 1st ed. New York: Chapman and Hall/CRC, 2021. ISBN 9781003029878. Dostupné z: https://doi.org/10.1201/9781003029878.

[9] GARDNER, M. L., FREITAS, M. A.: *Multiple Imputation Approaches Applied to the Missing Value Problem in Bottom-Up Proteomics*. International Journal of Molecular Sciences, 22(17): 9650, 2021. Dostupné z: https://doi.org/10.3390/ijms22179650.

[10] GU, Z.: *Complex Heatmap Visualization*. iMeta, 1(3): e43, 2022. Dostupné z: https://doi.org/10.1002/imt2.43.

[11] HRON, K., KUNDEROVÁ, P., VENCÁLEK, O.: *Základy počtu pravděpodobnosti a metod matematické statistiky*. 3. přepracované vydání. Olomouc: Univerzita Palackého v Olomouci, 2018. ISBN 978-80-244-5398-9.

[12] JIN, L., BI, Y., HU, C. et al.: *A comparative study of evaluating missing value imputation methods in label-free proteomics*. Sci Rep, 11: 1760, 2021. Dostupné z: https://doi.org/10.1038/s41598-021-81279-4.

[13] KALABUSOVÁ, V.: *Zpracování eye-tracking dat v R* [online]. Olomouc, 2022 [cit. 2024-02-12]. Dostupné z: https://theses.cz/id/tttj25/. Diplomová práce. Univerzita Palackého v Olomouci, Přírodovědecká fakulta. Vedoucí práce Mgr. Kamila Fačevicová, Ph.D.

[14] KAMMERS, K., COLE, R. N., TIENGWE, C., RUCZINSKI, I.: *Detecting significant changes in protein abundance*. EuPA Open Proteomics, 7, s. 11-19, 2015. ISSN 2212-9685. Dostupné z: https://doi.org/10.1016/j.euprot.2015.02.002.

[15] KONG, W., HUI, H. W. H., PENG, H., GOH, W. W. B.: *Dealing with missing values in proteomics data*. PROTEOMICS, 22(23-24), 2022. Dostupné z: https://doi.org/10.1002/pmic.202200092.

[16] KOVÁŘOVÁ, H.: *Proteomics in Postgenome Era* [online]. Chemické listy, 99(12), s. 886–889, 2006 [cit. 2024-01-24]. Dostupné z: http://www.chemicke-listy.cz/ojs3/index.php/chemicke-listy/article/view/1964.

[17] KIM, G., DEBOIS, P., WILLIS, J., HUMBLE, J.: *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016. ISBN 1942788002

[18] LIU, M., DONGRE, A.: *Proper imputation of missing values in proteomics datasets for differential expression analysis*. Briefings in Bioinformatics, 22(3), 2021. Dostupné z: https://doi.org/10.1093/bib/bbaa112

[19] MERTENS, B. J. A.: *Transformation, Normalization, and Batch Effect in the Analysis of Mass Spectrometry Data for Omics Studies*. In: DATTA, S., MERTENS, B. J. A.: *Statistical Analysis of Proteomics, Metabolomics, and Lipidomics Data Using Mass Spectrometry. Frontiers in Probability and the Statistical Sciences*. Cham: Springer International Publishing, s. 1–21, 2017. ISBN 978-3-319-45809-0. Dostupné z: https://doi.org/10.1007/978-3-319-45809-0_1.

[20] PAULO, J. A., KADIYALA, V., BANKS, P. A., CONWELL, D. L., STEEN, H.: *Mass Spectrometry-based Quantitative Proteomic Profiling of Human Pancreatic and Hepatic Stellate Cell Lines*. Genomics, Proteomics & Bioinformatics, 11(2), s. 105-113, 2013. ISSN 1672-0229. Dostupné z: https://doi.org/10.1016/j.gpb.2013.01.009.

[21] STEKHOVEN, D. J., BÜHLMANN, P.: *MissForest—non-parametric missing value imputation for mixed-type data*. Bioinformatics, 28(1), s. 112-118, 2012. Dostupné z: https://doi.org/10.1093/bioinformatics/btr597.

[22] TROYANSKAYA, O., CANTOR, M., SHERLOCK, G., BROWN, P., HASTIE, T., TIBSHIRANI, R., BOTSTEIN, D., ALTMAN, R. B.: *Missing value estimation methods for DNA microarrays*. Bioinformatics, 17(6), s. 520-525, 2001. Dostupné z: https://doi.org/10.1093/bioinformatics/17.6.520.

[23] TYANOVA, S., TEMU, T., COX, J.: *The MaxQuant computational platform for mass spectrometry-based shotgun proteomics* [online]. Nat Protoc, 11, s. 2301–2319, 2016 [cit. 2024-02-07]. Dostupné z:https://doi.org/10.1038/nprot.2016.136.

[24] WICKHAM, H.: *Mastering shiny* [online]. O'Reilly Media, Inc., 2021 [cit. 2024- 02-12]. Dostupné z: https://mastering-shiny.org/.

[25] WICKHAM, H., BRYAN, J.: *R Packages*. 2nd Edition. O'Reilly Media, Inc., 2023. ISBN 9781098134945

[26] WILKINS, M. R. et al.: *Progress with Proteome Projects: Why all Proteins Expressed by a Genome Should be Identified and How To Do It*

[online]. Biotechnology and Genetic Engineering Reviews, 13(1), s. 19-50, 1996 [cit. 2024-01-24]. Dostupné z: https://doi.org/10.1080/02648725.1996.10647923.

[27] YANG, Y. et al.: *StatsPro: Systematic integration and evaluation of statistical approaches for detecting differential expression in label-free quantitative proteomics* [online]. Journal of Proteomics, 250, s. 104386, 2022 [cit. 2024-01-26]. ISSN 1874-3919. Dostupné z: https://doi.org/10.1016/j.jprot.2021.104386.

Internetové zdroje

[28] *Understanding Volcano Plots*. HTG Molecular [online]. 2022. [cit. 2024-04-15]. Dostupné z: https://www.htgmolecular.com/blog/2022-08-25/understanding-volcano-plots

[29] IRIZARRY, R. [@rafalab]. (2014, December 18). *.@ewanbirney Neither corresponds to the @Bioconductor implementation of quantile norm (est 2001 and explained below)* [Image attached] [Tweet]. X. https://twitter.com/rafalab/status/545586012219772928.

[30] PIPIS, G.: *The Benjamini-Hochberg procedure (FDR) and P-Value Adjusted Explained*. R-bloggers.com [online]. 2023. [cit. 2024-04-15]. Dostupné z: https://www.r-bloggers.com/2023/07/the-benjamini-hochberg-procedure-fdr-and-p-value-adjusted-explained/.

[31] *Proteome Discoverer Software* [online]. © Copyright 2006-2024 Thermo Fisher Scientific Inc. [cit. 2024-02-07]. Dostupné z: https://www.thermofisher.com/cz/en/home/industrial/mass-spectrometry/liquid-chromatography-mass-spectrometry-lc-ms/lc-ms-software/multi-omics-data-analysis/proteome-discoverer-software.html.

[32] WICKHAW, H, DANENBERG, P., CSÁRDI, G., EUGSTER, M.: *roxygen2: In-Line Documentation for R*. R package version 7.3.1, https://github.com/r-lib/roxygen2, 2024 [cit. 2024-03-10]. Dostupné z: https://roxygen2.r-lib.org/.

Balíčky softwaru R

[33] Attali D (2021). *shinyjs: Easily Improve the User Experience of Your Shiny Apps in Seconds*. R package version 2.1.0, https://CRAN.R-project.org/package=shinyjs.

[34] Attali D, Edwards T (2021). *shinyalert: Easily Create Pretty Popup Messages (Modals) in 'Shiny'*. R package version 3.0.0, https://CRAN.R-project.org/package=shinyalert.

[35] Bolstad B (2023). *preprocessCore: A collection of pre-processing functions*. doi:10.18129/B9.bioc.preprocessCore https://doi.org/10.18129/B9.bioc.preprocessCore, R package version 1.62.1, https://bioconductor.org/packages/preprocessCore.

[36] Chang W, Cheng J, Allaire J, Sievert C, Schloerke B, Xie Y, Allen J, McPherson J, Dipert A, Borges B (2023). *shiny: Web Application Framework for R*. R package version 1.8.0, https://CRAN.R-project.org/package=shiny.

[37] Coene J (2021). *sever: Customise 'Shiny' Disconnected Screens and Error Messages*. R package version 0.0.7, https://CRAN.R-project.org/package=sever.

[38] Csárdi G (2021). *rcmdcheck: Run 'R CMD check' from 'R' and Capture Results*. R package version 1.4.0, https://CRAN.R-project.org/package=rcmdcheck.

[39] Fay C, Guyader V, Rochette S, Girard C (2023). *golem: A Framework for Robust Shiny Applications*. R package version 0.4.1, https://CRAN.R-project.org/package=golem.

[40] Fay C, Rochette S (2020). *shinipsum: Lorem-Ipsum Helper Function for 'shiny' Prototyping*. R package version 0.1.0, https://CRAN.R-project.org/package=shinipsum.

[41] Hastie T, Tibshirani R, Narasimhan B, Chu G (2023). *impute: Imputation for microarray data*. doi:10.18129/B9.bioc.impute, R package version 1.76.0, https://bioconductor.org/packages/impute.

[42] Kassambara A (2023). rstatix: Pipe-Friendly Framework for Basic Statistical Tests. R package version 0.7.2, https://rpkgs.datanovia.com/rstatix/.

[43] Merlino A, Howard P (2021). *shinyFeedback: Display User Feedback in Shiny Apps*. R package version 0.4.0, https://CRAN.R-project.org/package=shinyFeedback.

[44] Rochette S, Guyader V, Delmotte M, Floc'hlay S (2023). *attachment: Deal with Dependencies*. R package version 0.4.0, https://CRAN.R-project.org/package=attachment.

[45] Wickham H, Bryan J, Barrett M, Teucher A (2023). *usethis: Automate Package and Project Setup*. R package version 2.2.2, https://CRAN.R-project.org/package=usethis.

[46] Wickham H, Danenberg P, Csárdi G, Eugster M (2022). *roxygen2: In-Line Documentation for R*. R package version 7.2.3, https://CRAN.R-project.org/package=roxygen2.