PALACKÝ UNIVERSITY OLOMOUC
FACULTY OF SCIENCE

BACHELOR THESIS

Automatic recognition of an audit data structure

Department of Mathematical Analysis and Application of Mathematics
Supervisor: **Mgr. Iveta Bebčáková, Ph.D.**
Student: **Bohdana Nazarchuk**
Study program: B1103 Applied Mathematics
Specialisation: Mathematics-Economics of Banking / Insurance Systems
Form of studies: full-time
Submission year: 2018

BIBLIOGRAFICKÁ IDENTIFIKACE

Autor: Bohdana Nazarchuk

Název práce: Automatické rozpoznání struktury auditorských dat

Typ práce: Bakalářská práce

Pracoviště: Katedra matematické analýzy a aplikací matematiky

Vedoucí práce: Mgr. Iveta Bebčáková, Ph.D.

Rok obhajoby práce: 2018

Abstrakt: Každý podnikový informační systém generuje registr majetků v nějakou strukturu (názvy sloupců, pořadí sloupců, formát buněk atd.). Na základě analýzy hodnot a vztahů mezi sloupky se však dá zpětně určit, jakou informace sloupec obsahuje. Cílem této práce je naprogramovat pomocí programovacího jazyka Python chytrý nástroj pro automatické rozpoznání struktury dát (na základě reálných dát poskytnutých společností *Pricewaterhouse Coopers*). Takový nástroj pomáhá zvýšit produktivitu práce a uvolnit čas pro řešení komplikovaných problémů.

Klíčová slova: Audit, Python, struktura dát, automatizace, algoritmy, statistika

Počet stran: 43

Počet příloh: 1

Jazyk: anglický

BIBLIOGRAPHICAL IDENTIFICATION

Author: Bohdana Nazarchuk

Title: Automatic recognition of an audit data structure

Type of thesis: Bachelor's

Department: Department of Mathematical Analysis and Application of Mathematics

Supervisor: Mgr. Iveta Bebčáková, Ph.D.

The year of presentation: 2018

Abstract: Every enterprise resource planning system generates a register of assets in a given structure (the names of the columns, the order of the columns, the format of the cells etc.). Based on analysis of the values and relations between the columns it is possible to find out what information each column possesses. The aim of this bachelor's thesis is to design a smart tool with the help of programming language Python which automatically recognizes the structure of the data and rebuilds it in the way appropriate for further usage (using the real data provided by the *PricewaterhouseCoopers* company). Such a tool helps to increase productivity and release resources for solving more sophisticated tasks. This thesis creates a real value for the global leading company while solving a set of interdisciplinary tasks, including audit, programming, statistics and theory of algorithms. As being a part of the collaboration with the *PricewaterhouseCoopers* company this work implies continuation with possible implementation of sophisticated mathematical algorithms for so-called Machine Learning.

Key words: Audit, python, data structure, automation, algorithms, statistics

Number of pages: 43

Number of appendices: 1

Language: English

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracovala samostatně pod vedením paní Mgr. Ivety Bebčákové, Ph.D. a všechny použité zdroje jsem uvedla v seznamu literatury.

V Olomouci dne ....

podpis

Contents

List of Figures

Listings

Abbreviations

AD Accumulated depreciation

BI Business Intelligence

CAATs Computer Assisted Audit Techniques

csv comma-separated values format

GBV Gross book value

IT Information Technologies

NBV Net book value

PwC PricewaterhouseCoopers

SIMD single instruction multiple data

xls Microsoft Excel file format

Poděkování

Ráda bych poděkovala společnosti PwC Czech Republic a to oddělení řízení IT rizik za ochotu a čas věnovaný při psaní této bakalářské práce. Také bych ráda poděkovala paní Mgr. Ivetě Bebčákové, Ph.D. za odborné vedení práce a panu Mgr. Ondřeji Vencálkovi, Ph.D. za cenné rady a připomínky.

Introduction

Our century is the century of Information Technologies (IT) and the main trend today is the automation of all possible repetitive and manual processes. This transition is a key to development of new technologies, it moves the global progress forward. We see how automation changes industries. McKinsey Global Institute's report [1], assesses that by 2030 year, 75 million to 375 million workers (3 to 14 percent of the global workforce) will have to change occupation because of global automation and consequently, extinction of some professions. Moreover, all workers will need to adapt, as machines become more powerful and occupations develop as fast as new technologies do. The sphere of audit is not an exception. That is why one of the world's biggest auditing companies PricewaterhouseCoopers (PwC) follows these trends by launching IT departments for tasks automation, forecasting and developing smart solutions. PwC [2] highlights the following eight technologies as the most disruptive and promising to business at the same time: Artificial intelligence, Augmented reality/virtual reality, Blockchain, Drones, Internet of things, Robots, 3D printing, Autonomous vehicles. PwC's IT departments are ready to help companies to face difficult challenges and exciting opportunities, which technology progress brings to the world of business. This bachelor thesis is written during the close collaboration with PwC Czech Republic. We use real data provided by the company to perform a project focused on automation and acceleration of audit results analysis. There are a lot of Business Intelligence (BI) tools [3], such as Power BI and Qlik Sense, which can provide us with data visualisation and data analysis. But before data can be used in the purpose of analysis, it should have particular structure

to load it in BI tool and perform any further operations with it. Data cleansing is an important process that helps companies to increase their efficiency and decrease time spent on manual routine work. The idea of this thesis is close to the idea of data mining [4], automatic process of discovering certain patterns in data. It is not completely new, people have been seeking patterns in data since life began and engineers, economists and statisticians have pushed ahead an idea that patterns in data can be identified automatically and used for analysis and prediction. But the new is a huge and fast increase in opportunities for working with data. It is concerned with an amazing growth of databases in recent years, which makes data mining one of the most significant business technologies. The aim of this bachelor thesis is to create a smart tool with the help of programming language Python which automatically recognizes the structure of the data based on the relations between values, statistical analysis and text recognition. This tool then transforms the data to the form suitable for subsequent visualisation and analysis. This smart solution is vital for the company as it increases the quality of services provided, it also reduces the amount of manual time-consuming work and consequently allows to focus on assignments which require more intellectual resources. Meanwhile, this is not a trivial task to put on the automation as it implies the combination of interdisciplinary approaches among which are mathematical statistics, programming, theory of algorithms and audit itself.

1. The Impact of IT on the Audit Process

Audit is very ancient phenomenon, there are some confirmations of its existence in ancient cultures such as China, Rome, Egypt and Greece [5] [6]. There are also some records of auditing activity that were found by anthropologists in early Babylonian times (around 3000 BC) [5]. "To hear" is the meaning of the Latin word "audire", which is the basis of the well-known term "audit". There are several versions of the parentage of this word. For example, according to [5], in medieval times in Rome auditors used to listen to the taxpayers, such as farmers, telling about the progress and results of their business and the tax duty due. But Derek Matthews [7] mentioned that this term appeared because in that time all book-keeping was manual and accounts were read out to auditors in Britain so they could check its correctness. Nowadays audit is defined in the following way: *"An audit is the examination of the financial report of an organisation - as presented in the annual report - by someone independent of that organisation. The purpose of an audit is to form a view on whether the information presented in the financial report, taken as a whole, reflects the financial position of the organisation at a given date"* [8].

The sphere of audit is continuously developing with the times and considerable changes had started in 1950 – 1960s with the technologies and computer boom in business area. At that time auditors mostly continued their business on paper, only a few industrial companies used new technologies for data processing operations. But computers are becoming more multifunctional and new technologies are so progressive, that it is impossible not to follow them. In the 21st century all

business transactions are conducted through information technologies and here appears a term Computer Assisted Audit Techniques (CAATs). CAATs are a set of tools, which help to automate and speed up the audit procedure and to achieve the goals of auditing [9]. The main reasons to incorporate information technologies into auditing process can be summarised as follows [10]:

- Ability to cope with difficult tasks
  For example, tasks which require working with huge amount of data and which is almost impossible to complete manually.

- Increased productivity
  Auditors can focus on more significant issues, while all the routine work will be done automatically.

- Competitive advantage gained
  Using CAATs improves client's perception of the quality of services provided by the company.

CAATs include: data analysis software, BI tools, network security evaluation software, software and code testing tools, generalized audit software etc.

2. Data description

Every enterprise resource planning system generates a register of assets in some structure (the names of the columns, the order of the columns, the format of the cells etc.). For statistical analysis and processing, data is usually stored in comma-separated values (csv) or Microsoft Excel file (xls) formats. In our case, we have data stored in xls format. We have 35 testing sets of data with different number of rows and columns. This are only a training data samples, our tool is created to work on any other data that will be loaded.

In Fig. 1 one of the data sample is represented as a snapshot from the command line output of the Python script. There are three first rows of the data, that were anonymised because this data is sensitive. You can see there a lot of different metrics, but we are interested only in few of them: *Item Number, Item Description, Gross Book Value, Accounting Accumulated Depreciation and Net Book Value*. Our aim is to program a tool which will automatically recognise columns containing these metrics and will create a new xls file with these columns only. Columns like *Item Description, Accounting Gross Book Value, Accounting Accumulated Depreciation and Accounting Net Book Value* can be recognized using formulas, relations between these columns and other characteristics of the figures in columns. There is also a column which can be recognised only by name: *Item Number*. Our script is divided into several parts, where each part recognizes different columns.

```
  Majetek -item number Inventární číslo Popis - item description    SKP/CZ-CC  \
0                10001           K9977j                      AUTO  00.92.22.66
1                10002            NP341                  TISKARNA  67.00.01.70
2                10003            NI72K                   TELEFON  03.77.24.09

  Datum uvedení do užívání - activation date  \
0                                 1996-05-19
1                                 2001-11-09
2                                 1993-03-02

   ÚČETNÍ Pořizovací cena - gross book value  DAŇOVÁ Pořizovací cena  \
0                                        0.0                     0.0
1                                   353999.0                353999.0
2                                   198708.0                198708.0

   ÚČETNÍ Odpisy - depreciation  DAŇOVÁ Odpisy  \
0                           NaN            NaN
1                           0.0            0.0
2                           0.0            0.0

   ÚČETNÍ Oprávky - accumulated depreciation           ...            \
0                                       NaN          ...
1                                  353999.0          ...
2                                  198708.0          ...

   DAŇOVÁ Odpisová metoda  ÚČETNÍ Životnost  DAŇOVÁ Životnost Účet pořízení  \
0                   ZRY94               0.0              50.0     9999990.0
1                   ZRY99              80.0              92.0     8833500.0
2                   ZRY67              30.0              86.0     4427299.0

  Účet odpisů Účet oprávek ASSET_ID ASSET_CATEGORY_ID - asset class  \
0  55145638.0   43430100.0  10001.0                            347.0
1  84756666.0   48375230.0  10002.0                            645.0
2  55734832.0   39833333.0  10003.0                            957.0

   BOOK_TYPE_CODE_UCETNI  BOOK_TYPE_CODE_DANOVA
0                 UCETNI                 DANOVA
1                 UCETNI                 DANOVA
2                 UCETNI                 DANOVA
```

Figure 1: Data structure representation

3. Motivation for choosing Python

To perform this automation, it is possible to use different programming languages. In this chapter, some general information about programming language Python and reasons for choosing exactly this option are represented.

In [11] we can find the following definition: *"Python is a general-purpose, object-oriented, and open source computer programming language."* Python was released in year 1991 and now it is one of the most popular programming languages. Python is widely used both among non-technical users and among leading IT companies. For example, many components of the Google search engine are written in Python, Johnson Space Centre uses Python in its Integrated Planning System as the standard scripting language, famous computer game Battlefield 2 uses Python to implement core elements of the gameplay and there are lots of other interesting implementations of Python [12].

Here are the main advantages of programming language Python, which are clearly described in [13]:

- Libraries supporting

  Standard library is a large collection of prebuilt functionality, which allows users not to create their own complicated scripts every time, but to use some already invented tools. There are a lot of different libraries [14] made specially for working with data, for example Pandas. This is the reason why Python is so popular among data scientists [15] and this is also the main reason why we use it in this coursework.

- Software quality

  Python's readability and software quality in general make it dominant over other languages. Thanks to its uniformity Python code is easy to read and understand, even if it was written by someone else.

- User-friendliness

  Compared to programming languages like C++ and Java, Python is easy to run, without any intermediate compile and link steps. Also Python code is usually one-third to one-fifth the size of the same code written in C++ or Java.

- Program portability

  Python programs run on every major computer platform. To port Python code from Windows to Linux we need only to copy script's code between machines.

4. Recognition of column by name: *Item Number*

This chapter contains information concerning the first part of our task, recognition of column *Item number* only by the name. This metric has no specific features, which would enable us to involve more elegant and reliable solution. Lis.1 represents our script. In this code, we used regular expressions and library Pandas, which allow us to work with data.

4.1 Regular expressions

The term *regular expression* was introduced after American mathematician Stephen Cole Kleene formalized the concept of regular languages and *finite automata* [16] which was based on a pioneering work of McCulloch and Pitts [17] where they studied the behaviour of nervous systems. In theoretical computer science, the *regular expression* is a sequence of characters defining the search pattern. Then this pattern is usually used by searching algorithms. The syntax of *regular expressions* is standardised by the POSIX and Perl standards (the latter is the mostly used one).

The most common usage of regular expressions may be formulated by three examples:

- Search for a specific pattern appearance in text. For example, check if either the word "recognize" or the word "recognise" appears in a loaded text.
- Replace parts of text. For example, change all the words similar to "tree" with "spruce".

- Check if an input corresponds a predetermined pattern. For example, check if an information entered in a HTML formulary is a valid e-mail address.

In Python regular expressions are supported by the *re* module which provides matching operations similar to Perl.

4.2 Code

The logic of the script in Lis.1 is as follows:

1. Upload initial target xls file, choose particular sheet and save it in a data frame.

2. Create a list of all key words (all possible names of columns).

   As PwC Czech Republic is a part of network of firms all over the world, column name can be in different languages and it also can be in a form of different abbreviations. We have a set of testing files, so we went through each file and added all possible names of column, which occur over there. It is easy to append any other key words in our script, it can be modified according to the current needs.

3. Create a list of all column names from initial xls file.

4. Create a regular expression to search key words and make it ignore letter case.

5. Create an empty list for being used later to store new columns.

6. Create a loop for choosing columns, which contain key words from initial data frame.

7. Create a new data frame for storing chosen columns.

8. Rewrite chosen columns from initial data frame to new data frame.

9. Export new data frame to xls file.

```
import pandas as pd
import re

#upload Excel file
xl = pd.ExcelFile(".\\data\\01 Majetek FY17.xlsx")
#read particular excel sheet
df = xl.parse("2017_Majetek")

#list of all keywords
key_words1 = ['majetek', 'item number', 'i_cislo', 'Inventarne cislo
    ', 'Inv.majetek', 'Třída IM', 'M_CISLO', 'Číslo', 'INV_CISLO']

#list of all column names in initial excel file
col_list = list(df.columns)

#regular expression to search key words
words_re = re.compile("|".join(key_words1),re.IGNORECASE)

#list of new columns
new_col_list = []

#loop to choose right columns from old data frame
for i in col_list:
    if words_re.search(i):
        print(i)
        new_col_list.append(i)

#new dataframe
df_new = pd.DataFrame()

#rewriting new columns from old data frame to new data frame
df_new[new_col_list] = df[new_col_list]

#export to xls file
writer = ExcelWriter('new_data.xlsx')
df_new.to_excel(writer,'Sheet1')
writer.save()
```

Listing 1: Python script for recognition of columns by name

With the help of the Python script Lis.1, we exported recognised column *Item Number* to a new xls file.

5. Recognition of columns by their characteristics

In this chapter, the methods which require deeper analysis and more sophisticated approaches are presented. The goal is to use more reliable methods for the columns recognition as recognition by name may not always lead to desired results.

5.1 Columns *Accounting Gross Book Value, Accounting Net Book Value, Accounting Accumulated Depreciation*

In this section, we present the way to recognize the columns *Accounting Gross Book Value, Accounting Net Book Value* and *Accounting Accumulated Depreciation*. The definitions of these metrics are as follows:
Gross book value (GBV) – initial cost of an asset, including all the shipping, taxes and other fees related to this purchase [18].
Accumulated depreciation (AD) – the sum of depreciation for an asset that has been ascribed to expense since that asset was acquired and put into service [19].
Net book value (NBV) – the amount at which a company carries an asset on its accounting records. It is equal to the difference of gross book value and accumulated depreciation [19].
From this definitions there is an obvious relation between our values of interest:

$$\mathrm{GBV} = \mathrm{NVB} + \mathrm{AD} \tag{1}$$

Introduced formula is a key to recognition of this columns. We have to find three columns where one is the sum of the other two. There are several ways to do it,

in the following subsections, we represent two different approaches, which give us the same result but are characterized by different computational complexity and execution time.

5.1.1 Computational complexity

Firstly we introduce the term Computational Complexity which will allow us to characterize different algorithms we are going to use in the course of this work. In informatics and theory of algorithms, computational complexity is defined as the dependence of the amount of work performed by a given algorithm from the amount of input data $f(n)$ [20]. The amount of work is usually defined by abstract notions of time and space referred as computational resources. The former one is defined as a number of basic steps necessary to perform the computation and the latter one is usually related with the amount of memory used. According to the Church-Turing thesis [21], if the problem can be solved by an algorithm, there exists a Turing machine which can solve this problem. By Turing machine [22] we suppose a general mathematical model of any computing instance, whether it is a quantum computer or a simple calculator, and which may be represented as mechanism manipulating symbols on a string of tapes.

The main types of Turing machines [23] are deterministic and non-deterministic ones. Deterministic Turing machine is the mechanism using a fixed set of rules which determine its actions. The non-deterministic Turing machine is the one with extra random bits. This allows the latter to be more efficient in some computations.

Based on this definition there are different complexity classes [23] [24]. If we talk about the time complexity the three main classes are DTIME (deterministic Turing machine within time $f(n)$), P (deterministic Turing machine within polynomial time pol(n)) and EXPTIME (deterministic Turing machine within exponential time exp(n)). The analogous classes exist for the non-deterministic machines and they are referred as NTIME, NP and NEXPTIME correspondingly.

Usually the explicit form of time complexity $f(n)$ is very hard to derive

and hence some estimations should be used instead. The consideration of large amounts of input data and estimations of increase of computational time lead to the notion of asymptotical complexity of the algorithm. The following notation is used to determine asymptotical behaviour of the function [25]:

$$f(n) \in O(g(n)). \tag{2}$$

It (2) means that function f has the asymptotical upper threshold g:

$$\exists C > 0, n_0 : \forall n > n_0 \; |f(n)| \leq C|g(n)|,$$

The complexity of an algorithm is usually determined as the worst-case complexity.

Computational complexity is closely related with the *Kolmogorov complexity*, which was firstly introduced by Andrey Kolmogorov in year 1963 [26]. In the algorithmic information theory, the Kolmogorov complexity is the length of the shortest program needed to produce this object as the output of the program's execution. Suppose there are two string of the same length of 24 symbols:

String 1: wklwklwklwklwklwklwklwkl
String 2: hkwzb34skwl1zj9sdnsdkoao

String 1 can be described by 11 characters: "wkl 8 times", whereas string 2 has no simple description except of the whole string itself, which consists of 24 characters. The length of the minimal description is the *Kolmogorov complexity* of the string. The same idea may be generalized to more complex objects.

5.1.2 Brute force method

Let us consider the first possible approach to identify columns of interest. As it was mentioned earlier this method is based on the known relation (1). It means that the problem may be reduced to the so-called *"2 -SUM"* which may be formulated in the following way. The one is given an array of n numbers and

a target sum S. The goal is to determine whether or not there are two numbers from the array which sum is equal to S. In our case, the array is the set of columns of the dataset and moreover, we have an additional dimension — rows of the dataset. Otherwise, our problem is very similar to the described one. The idea of algorithm we are implementing is that it iterates over all the columns in a dataset and for every column considered as possible sum S, the remaining ones are summed in pairs and compared with S. As there is also additional dimension (rows of the dataset), the operation of addition is applied on every row and the number of successful coincidences is calculated. If it is equal to 100%, then these three columns are the target ones. The time complexity of this algorithm may be estimated as

$$O(N \times n(n-1)^2), \tag{3}$$

where N is the number of rows in a dataset and n is a number of columns. This formula maybe explained in the following way. For every chosen element n_i from the array there are $(n-1)$ elements to calculate sum of pairs and compare with n_i. To calculate all the pairs one needs $(n-1)^2$ steps. This should be multiplied by the number of elements in the initial array n, as the same operation will be repeated for every single one, and by the number of rows N, which are supposed to be iterated through.

The logic of the Python script itself is presented below:

1. Firstly we need to keep only numerical data as the other types are out of our interest. In Python we check if the columns contain the *float64* datatype which basically stores decimal numbers.

```
#keep only columns with data type being float64
df = df1.loc[:, df1.dtypes == 'float64']
```

Listing 2: Script for keeping numerical data only

2. After we got rid of all unnecessary information we need to deal with the missing data. In our case, the easiest way is to fill the missing records with zeros as it's not going to ruin relations between columns.

```
#fill NaNs with zeros
df = df.fillna(0)
```

Listing 3: Filling all missing data with zeros

3. Next, we need to create a function to find all possible pairs of columns. This one will be used later while iterating through all the columns. It returns an array of all possible pairs when the list of items is passed to the input.

```
def pairs(source):
        result = []
        for p1 in range(len(source)):
                for p2 in range(p1+1,len(source)):
                        result.append([source[p1],source[p2]])
        return result
```

Listing 4: Defining a function to find all pairs of columns

4. Implementation of the main logic. The realisation of the algorithm is performed through couple of loops, one of them iterating through columns and creating list of pairs and another one iterating through rows and checking the coincidences. The percentage of coincidences is counted the following way: the number of rows where the sum is equal divided by the number of all rows and multiplied by 100.

```
#list of new columns without the current one (to find pairs)
    new_col = [x for x in columns if x != i]
    #create a list of all possible pairs from the new list
    pairings = pairs(new_col)
    #create a count of coincidences
    count = 0
    #create a count of rows
    count_r = 0
    #loop to iterate through all the pairs
    for j in pairings:
        #loop to go through all the rows in a paired columns
        for z, row in df.iterrows():
            #if not zero rows
            if(df.loc[z, i] != 0):
                #count of rows increased
                count_r = count_r + 1
                #check if the sum of these columns equal to the target column
                if(round(df.loc[z, j[0]] + df.loc[z, j[1]],2)==round(df.loc[z, i],2)):
                    #count of coincidences increased
```

```
                    count = count + 1
            #calculate the percentage of coincidences
            perc = (count/count_r)*100
            #set all the counts to zero for the next iteration
            count = 0
            count_r = 0
```

Listing 5: Creating list of pairs and checking the coincidences

After the script has been executed and in the case it has found the columns of interest, two situations are possible:

1. The percentage of coincidences is 100% within only one combination of columns. It means that this set of columns is the target one: *Accounting Gross Book Value, Accounting Net Book Value, Accounting Accumulated Depreciation*. In this case the script uploads them to a new xls file.

2. The percentage of coincidences is 100% within two or more combinations. This can happen because in the initial files usually there is not only accounting data, but also a tax data. This means that Tax Gross Book Value, Tax Net Book Value and Tax Accumulated Depreciation are also present and they have the same relation between them. In this case we have no other option but to check the names of the columns either manually or programmatically.

5.1.3 Hash Tables method

The previous method demonstrated quite a poor performance so the ways for it's enhancement should be found. In this purpose, we consider another algorithm which is known as a *Hash Table* method [27].

The following example represents an idea of this algorithm. Let assume that one has an array $A = \{1, 5, 0, 2, 7, 3, 4, 6\}$ and $S = 6$. The idea of the approach is to build a table containing keys and differences between keys and target value S. Once the table is build, one needs to iterate through the elements of the array, find them in the table and check if the stored result of difference between S and key refers to any other element in the array. This method implies much

less calculations than the brute force one. Tabl.1 is the example of a hash table for array A and $S = 6$.

Keys	Values	Pairs
1	$6 - 1 = \mathbf{5}$	(5; 1)
5	$6 - 5 = \mathbf{1}$	(1; 5)
0	$6 - 0 = \mathbf{6}$	(6; 0)
2	$6 - 2 = \mathbf{4}$	(4; 2)
7	$6 - 7 = -1$	
3	$6 - 3 = \mathbf{3}$	(3; 3)
4	$6 - 4 = \mathbf{2}$	(2; 4)
6	$6 - 6 = \mathbf{0}$	(0; 6)

Table 1: Example hash table

We can generalize this method to our case which has, as it was mentioned earlier, one additional dimension — rows. The problem of subtraction of two elements now is the problem of subtraction of two vectors v_1 and v_2:

$$v_1 = (a_1, a_2, \ldots, a_n), v_2 = (b_1, b_2, \ldots, b_n)$$

The result of this operation is new vector with the following elements:

$$v_r = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n),$$

which imply n operations to be performed. It is possible to do this by looping through all the elements of v_1 and v_2, what has been done in the brute force method, or we can go further in performance enhancement by implementing *vectorized operations*. Vectorized operations in contrast to scalar ones allow to apply the function on the object (array, vector) as a whole instead of being applied to every element individually. In computer science, this is called SIMD (single

instruction, multiple data) operation. The idea is that one command performs operation on several components at the same time. For hash table there is a need to subtract vector of size N from another vector of size N. Let us suppose that M is the number of components which can be operated concurrently and which is defined by the properties of the computational system. Using vectorization, the number of operations needed to execute is approximately N/M, compared to N for conventional looping.

The time complexity of the hash table method may be estimated as follows:

$$O(N \times n(n-1)), \tag{4}$$

where n is the number of columns and N is the number of rows. This formula may be explained in the following way. Hash table method implies n steps to be performed when applied to the array of n elements. As we have to iterate through all the columns and for each consider the separate *2-SUM* problem, we apply $n(n-1)$ steps. This should be multiplied by the number of rows N as for every one we are performing the same operations. In case of vectorized operations applied, (4) transforms to

$$O\left(N \times n(n-1)/M\right). \tag{5}$$

Lis.6 represents the implementation of hash tables method in the Python script:

1. First steps are similar to step 1 and 2 of brute force method. We need to keep only columns, which contain the *float64* data type that stores decimal numbers and fill missing data with zeros.

2. Create a list of all column names and an empty list, where output column names will be written.

3. Choose all the columns except of the current one.

4. Create a data frame, which we use in hash table (data frame in our case), with the differences between the current column and all other columns.

5. Go through initial list of all column names and compare them to the elements of hash table.

```
#keep only columns with data type being float64
df = df1.loc[:, df1.dtypes == 'float64']

#fill NaNs with zeros
df = df.fillna(0)

#create list of columns
columns = list(df)
new_col_list = []

for i in columns:
    #choose all the columns except of the current one
    new_col = [x for x in columns if x != i]
    #create data frame, which we use in the "hash table"
    df_keys = pd.DataFrame()
    for j in new_col:
        df_keys[j] = df[i] - df[j]
        df_keys[j] = df_keys[j].apply(lambda x: round(x,2))
    #go through initial list of columns and compare them to the
    elements of hash table
    for j in new_col:
        df[j] = df[j].apply(lambda x: round(x,2))
        for m in df_keys:
            if((df[j] == df_keys[m]).all()):
                print('Column' + i + 'is a sum of' + j + 'and' + m)
```

Listing 6: Hash table method

After the script has been executed and in the case it has found the columns of interest, the same as in brute force method two situations are possible:

1. The script has found only one combination of columns. It means that this set of columns is the target one: *Accounting Gross Book Value, Accounting Net Book Value, Accounting Accumulated Depreciation.* In this case the script uploads them to a new xls file.

2. The script has found two or more combinations. This can happen because in the initial files usually there is not only accounting data, but also a tax data. This means that Tax Gross Book Value, Tax Net Book Value and Tax Accumulated Depreciation are also present and they have the same relation between them. In this case we have no other option but to check the names of the columns either manually or programmatically.

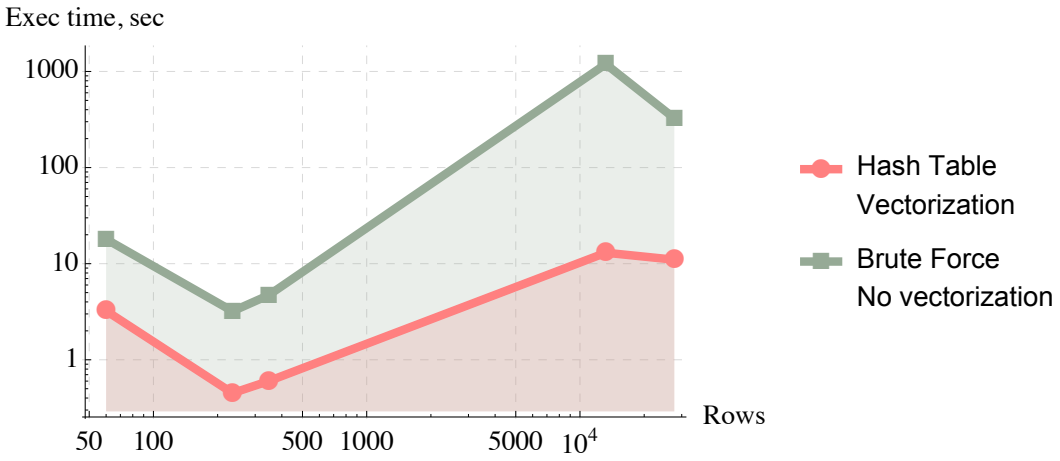

Figure 2: Dependence of execution time from the amount of data for five data sets. The figure is plotted in the logarithmic scale for both axes. The decreasing of the execution time for higher number of rows for some points is due to the drastically lower number of columns for these data samples.

After both methods have been applied we did a comparison between them. In this purpose, the execution time was calculated as the function of the input data amount (represented as a number of rows). The results are presented in Fig. 2. It is clearly seen that the hash table method together with vectorization shows much better performance and doesn't exceed 10 seconds even for 10^4 rows for which brute force method without the vectorization is executed in about 1000 seconds. Decreasing of the execution time with increased number of rows is related with the fact that for these particular points the number of columns was very low, enabling the script to finish the calculations quite fast.

5.2 Column *Item Description*

The next column which should be identified is the *Item Description* one and contains the description of the asset. The sample is presented in the Fig.3.

Based on the information obtained by the observation of many datasets, we know that this column should contain much more spaces than other columns with the symbolic data, as this column represents the syntactic structure. The idea behind the approach which we are going to apply is based on the statistics

Popis - item description
TERMOTISKARNA
TISKARNA TERMO PRINTER
NOTEBOOK LATITUDE DELL
PROGRAM FIREWALL
TERMOTISKARNA ALLEGRO XL
NOTEBOOK TWINHEAD TH VX23
TISKARNA
DELL OPTIPLEX
Licence Passport
INSTALACE PROGRAMU PASSPORT
PROGRAM NA MZDY
FAX CANON
AUTO OPEL VECTRA
AUTO RENAULT LAGUNA
BIONAIRE\CISTIC VZDUCHU
VYSOKOZDVIZNY VOZIK
WIN SYSTEM

Figure 3: Data sample of Item Description column

collected throughout multiple testing datasets provided by the *Pricewaterhouse-Coopers* company. For our disposition, there were 35 datasets and we divided them in two subsets — learning one (20 datasets) and testing one (15 datasets). For every learning dataset we calculated the distribution of the spaces for Item Description column. The sample distributions are presented in Fig.4. It is clearly seen that they look similar and resemble the *Poisson distribution*. Based on this fact we can calculate the properties of the distribution of spaces of a given column and compare them with the ones from the learning datasets.

To do so we firstly calculate the mean values for every distribution which are defined in the following way:

$$\mu = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right), \tag{6}$$

with n being the number of rows. But the mean value alone doesn't tell too much about the distribution itself. Another characteristic which we calculate is

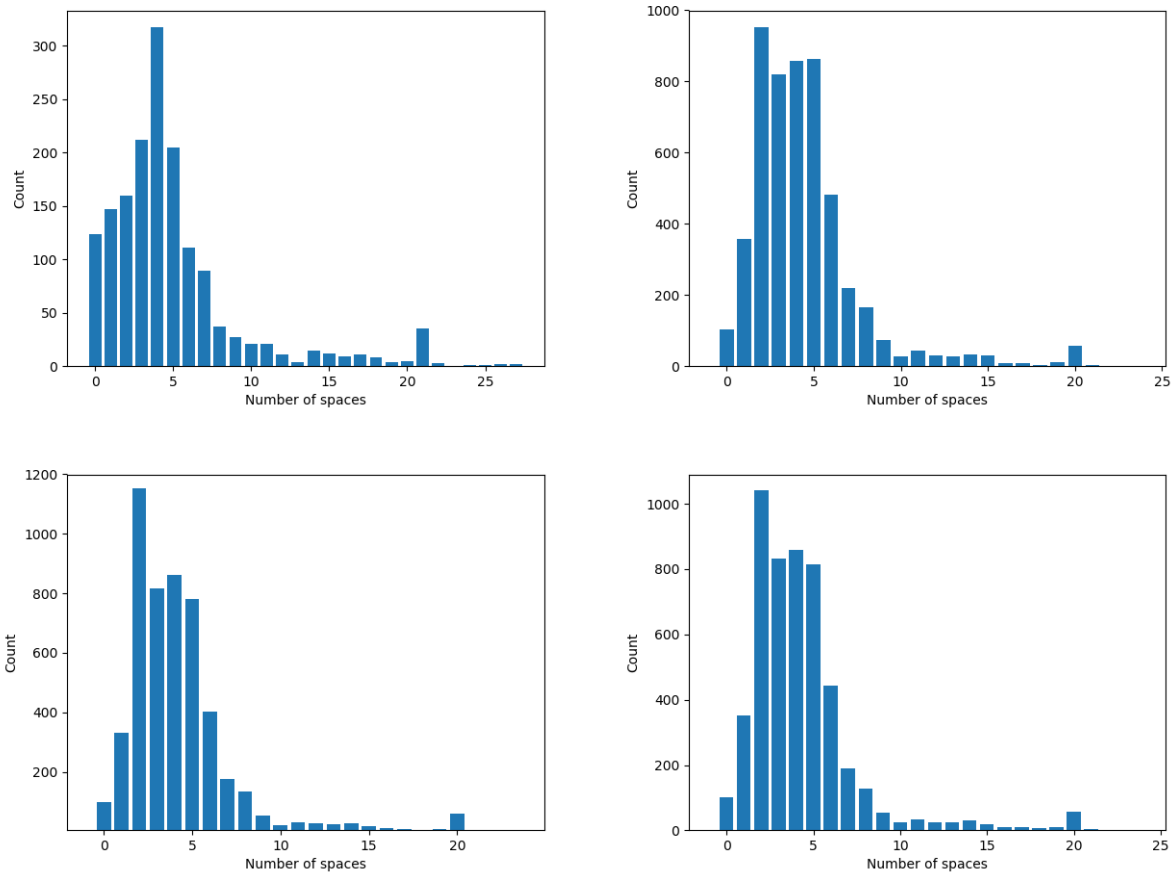

Figure 4: The sample of the distribution of spaces throughout the dataset for four different datasets

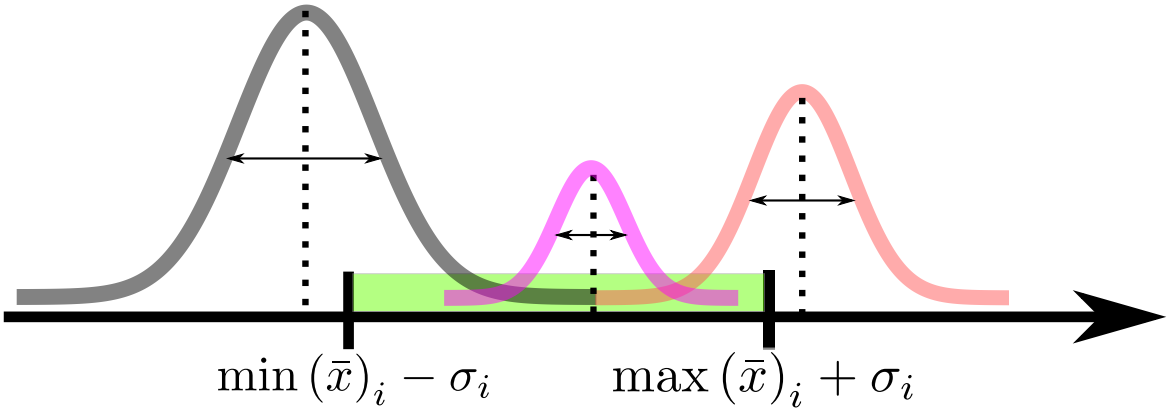

Figure 5: Schematic representation of the selection criteria. Once the mean value plus minus standard deviation intersects the region of interest – the column is marked as the target one.

the standard deviation defined as follows:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}, \tag{7}$$

with μ defined in (6). The standard deviation is a measure used to characterize the amount of dispersion of the dataset. When this value is low, it indicates that the points of a dataset tend to be closer to the mean value. For high values of the standard deviation the dataset is widely spread around the mean value. We use this value instead of variance as it is measured in the same units as the data.

After the values (6) and (7) have been calculated the interval of interest is created as the range between minimal mean minus the deviation for this dataset (which provided the minimal average) and maximal average plus deviation for this dataset (which provided the maximal average). The idea of the approach is then to calculate mean and deviation for the tested dataset and check if it does intersect the calculated interval as presented in Fig. 5. Once this intersection is present it means that the distribution is quite close to the common one for the column *Item description*.

The realization of this logic is represented below by two listings (Lis. 7 and Lis. 8).

Lis. 7 contains the preparatory code creating the interval of interest. Its structure is the following:

1. Create two list to store the output of averages and standard deviations for every of 20 learning files.

2. Iterate through all 20 learning files in the directory.

3. Read each file and write it to the data frame.

4. Create two list to store the numbers of spaces and squared differences for every row in Item Description column.

5. Iterate through rows in Item Description column, count the number of spaces for every row and add this number to the list, which was created in the previous step.

6. Calculate the average number of spaces in Item Description column. The number of spaces for every row in the column has been written to one list, so to calculate the average we need to divide the sum of the numbers in this list by its length.

7. Again iterate through rows in Item Description column and count the number of spaces for every row.

8. For every row, calculate the squared difference between the average number of spaces in Item Description column from step 6 and current value of number of spaces. Add each calculated value to the list created in step 4.

9. Calculate the average of the squared differences from the previous step to get standard deviation.

10. Store calculated average number of spaces in Item Description column and standard deviation from the previous step to the global list created in step 1 to later compare between different data frames (files).

11. Print out the average number of spaces and standard deviation in column Item Description for every dataset

12. Find minimum and maximum of all average numbers of spaces in Item Description column from the global list (from all learning files) to create an interval of interest. This interval is the range between minimal average value minus the standard deviation for this particular dataset and the maximal average value plus standard deviation for this dataset. In our case the interval is (2.3, 4.9).

```
#creating lists to store averages and deviations for every file
AVG_TOTAL_SPACES = []
DEV_TOTAL_SPACES = []

#iterating through all files in the directory
for file in os.listdir('.\\for_popis_test\\'):
    file1 = '.\\for_popis_test\\' + file
    print(file)

    #read file and write it to the data frame
    xl = pd.ExcelFile(file1)
    df = xl.parse("Sheet1")

    #creating lists to store numbers of spaces and squared
    differences
    COUNT_SPACES_POPIS = []
    DIFF_SQ_POPIS_SPACES = []

    #iterating through rows in the data frame
    for j, row in df.iterrows():
        #count spaces
        count_spaces_popis = str(df.loc[j, 'popis']).count(' ')
        #add count of spaces for this row to the list
        COUNT_SPACES_POPIS.append(count_spaces_popis)

    #calculate average number of spaces in the data frame
    avg_popis_spaces = sum(COUNT_SPACES_POPIS)/len(
    COUNT_SPACES_POPIS)

    #iterate through all the rows of the same data frame
    for j, row in df.iterrows():
        #count the number of spaces
        count_spaces_popis = str(df.loc[j, 'popis']).count(' ')
        #count the squared difference between average and current
    value
        diff_sq_popis_spaces = (count_spaces_popis -
    avg_popis_spaces)**2
```

```
34          #add calculated above value to the list
35          DIFF_SQ_POPIS_SPACES.append(diff_sq_popis_spaces)
36
37      #calculate the average of the squared differences to get
        standard deviation
38      dev_popis_spaces = math.sqrt(sum(DIFF_SQ_POPIS_SPACES)/len(
        DIFF_SQ_POPIS_SPACES))
39
40      #store calculated values to the global list to later compare
        between different data frames
41      AVG_TOTAL_SPACES.append(avg_popis_spaces)
42      DEV_TOTAL_SPACES.append(dev_popis_spaces)
43
44      #printing the values out
45      print('Popis: avg spaces = ' + str(avg_popis_spaces), 'dev
        spaces = ' + str(dev_popis_spaces))
46
47 #find minimum and maximum of the averages
48 min_avg_spaces = min(AVG_TOTAL_SPACES)
49 max_avg_spaces = max(AVG_TOTAL_SPACES)
```

Listing 7: Creating an interval of interest schematically represented in Fig. 5

Lis. 8 represents the code to select target columns based on the previously calculated interval. Its logic is as follows:

1. Choose the directory with target files and iterate through them.
2. Read particular file and sheet.
3. We need to keep only columns, which contain the *object* data type (basically a string), which is the data type of our column of interest Item Description.
4. Create two lists, one to store the number of spaces for every row in a file and the second one to store squared difference between the average number of spaces in every column and current value of number of spaces for every row.
5. Iterate through all rows in every column and do same cleaning, replace two or more spaces in a row by one.
6. Count the amount of spaces in each row and add it to the list, which was created in step 4.

7. Calculate the average number of spaces in every column. The number of spaces for every row in the column has been written to one list, so to calculate the average we need to divide the sum of the numbers in this list by its length.

8. Iterate through every row again to calculate the squared difference between the average number of spaces in every column from and current value of number of spaces. Add each calculated value to the list created in step 4.

9. Calculate the average of the squared differences from the previous step to get standard deviation.

10. Check the conditions to select correct column. If an interval (average number of spaces in column minus standard deviation, average number of spaces in column plus standard deviation) belongs to the interval, which was calculated with the help of Lis.7, then this is the target column Item Description.

```
#choose the directory with files
directory = '.\\for_popis_weights\\'

#iterate through files in the given directory
for file in os.listdir(directory):
    #path to the actual file
    file1 = directory + file

    #read file
    xl = pd.ExcelFile(file1)
    #parse sheet
    df = xl.parse("Sheet1")

    #choose columns with the data type being object
    df = df.loc[:, df.dtypes == 'object']

    #calculation of spaces statistics
    for i in df:
        #creation of storage lists
        COUNT_SPACES_POPIS = []
        DIFF_SQ_POPIS_SPACES = []

        #iterate through rows
        for j, row in df.iterrows():
            #replace three spaces in a row by one space
            df.at[j,i] = str(df.loc[j, i]).replace('   ',' ')
```

```
27              #delete all spaces at the end of the line
28              df.at[j,i] = str(df.loc[j, i]).rstrip()
29              #obvious replacements
30              df.at[j,i] = str(df.loc[j, i]).replace('    ',' ')
31              df.at[j,i] = str(df.loc[j, i]).replace('   ',' ')
32              df.at[j,i] = str(df.loc[j, i]).replace('  ',' ')
33
34              #count spaces
35              count_spaces_popis = str(df.loc[j, i]).count(' ')
36              #add to the storage
37              COUNT_SPACES_POPIS.append(count_spaces_popis)
38
39          #calculating average spaces
40          avg_popis_spaces = sum(COUNT_SPACES_POPIS)/len(
    COUNT_SPACES_POPIS)
41
42          #iterate through all the rows again to calculate the squared
     differences
43          for j, row in df.iterrows():
44              df.at[j,i] = str(df.loc[j, i]).replace('   ',' ')
45              df.at[j,i] = str(df.loc[j, i]).rstrip()
46              df.at[j,i] = str(df.loc[j, i]).replace('    ',' ')
47              df.at[j,i] = str(df.loc[j, i]).replace('   ',' ')
48              df.at[j,i] = str(df.loc[j, i]).replace('  ',' ')
49              count_spaces_popis = str(df.loc[j, i]).count(' ')
50              #calculate the squared differences
51              diff_sq_popis_spaces = (count_spaces_popis -
    avg_popis_spaces)**2
52              DIFF_SQ_POPIS_SPACES.append(diff_sq_popis_spaces)
53
54          #calculate standard deviation as the squared root from the
    averaged squared difference
55          dev_popis_spaces = math.sqrt(sum(DIFF_SQ_POPIS_SPACES)/len(
    DIFF_SQ_POPIS_SPACES))
56
57          #checking the conditions to select correct column
58          if(((avg_popis_spaces + dev_popis_spaces) > 2.3 and (
    avg_popis_spaces + dev_popis_spaces) < 4.9 ) or ((
    avg_popis_spaces - dev_popis_spaces) > 2.3 and (avg_popis_spaces
    - dev_popis_spaces) < 4.9)):
59              print('guessed column by spaces is ' + str(i))
60          else:
61              print('no columns have been found')
```

Listing 8: Implementation of the main logic for recognition of Item Description column

After the script has been executed and if there is some output, two situations are possible:

1. The script has found only one column. It means that this column is the

target one: *Item Description*.

2. The script has found two or more columns. This can happen because data is not clean enough or some other column has the same distribution of spaces. In this situation we have to check the names of the columns.

Conclusion

In this bachelor thesis, we showed the proof-of-the-principle automated preparation of the data for analysis. For solving this problem we used some basics of accountancy and audit to find out the relations between metrics in our data. Also, we performed statistical analysis of certain data by building distributions of the metric of interest for learning datasets. Our main tool was programming language Python, especially library Pandas, which allows working with data and regular expressions, which are used by searching algorithms. The reader of this thesis can get acquainted with three different approaches to data recognition. The first one is recognition of columns by name with the help of key words using regular expressions. The second one is recognition of columns by known relations between them. Here we had to solve so-called *"2 -SUM"* problem and at this point we showed the difference of two methods. For solving this problem we applied brute force method and Hash Tables method together with vectorization. Our experiment showed that Hash Tables method plus vectorization raises the performance for growing amount of data more than a hundred times comparing to brute force method, so the decrease of execution time is highly considerable. The last but not least approach is recognition of columns using statistics. We used a part of datasets for learning, building the distribution of our metric of interest for every learning file, and we found out that this metric has Poisson-like distribution. Based on this fact we calculated the mean values and the standard deviations for every distribution. After that we created an interval of interest as the range between minimal mean minus the deviation for this dataset and maximal mean plus deviation for this dataset. On testing datasets, we calcu-

lated mean values and standard deviations and checked if it does intersect the calculated interval of interest. This approach shows quite stable efficiency.

We believe that this bachelor thesis could be useful for anyone, who is interested in implementation of Python language while working with data, who strives for reducing time spent on routine manual work and who is looking for examples of application statistics and theory of algorithms in data science. Also, created tool makes a real value for one of the world's largest professional services firm *PricewaterhouseCoopers* by solving a set of interdisciplinary tasks. All Python scripts together with data for testing can be found as an appendix to this bachelor thesis.

As being a part of the collaboration with the *PricewaterhouseCoopers* company this work implies continuation with recognition of another columns and possible implementation of sophisticated mathematical algorithms for so-called Machine Learning.

Bibliography

[1] MANYIKA, James et. al.: *JOBS LOST, JOBS GAINED: WORKFORCE TRANSITIONS IN A TIME OF AUTOMATION* McKinsey Global Institute, 2017.

[2] PwC Global: *Technology* [online]. [cit. 10.04.2018]. Available from: https://www.pwc.com/gx/en/industries/technology.html

[3] Trust Radius: *Best Business Intelligence Tools* [online]. [cit. 10.04.2018]. Available from: https://www.trustradius.com/bi

[4] WITTEN, Ian H. et. al.: *Data Mining: Practical Machine Learning Tools and Techniques. 4th ed.* Amsterdam: Elsevier, 2017. Morgan Kaufmann is an imprint of Elsevier. ISBN 9780128042915.

[5] HAYES, Rick et. al.: *PRINCIPLES OF AUDITING. An Introduction to International Standards on Auditing. 2nd ed.* Edinburgh: Pearson Education Limited, 2005. ISBN 0273684108.

[6] BROWN, Richard et. al.: *A history of accounting and accountants.* Edinburgh: T.C. and E.C. Jack, 1905.

[7] MATTHEWS, Derek.: *A history of auditing: the changing audit process in Britain from the nineteenth century to the present day.* New York: Routledge, 2006. ISBN 041538169X.

[8] PwC Middle East: *What is an audit?* [online]. [cit. 13.11.2017]. Available from: https://www.pwc.com/m1/en/services/assurance/what-is-an-audit.html

[9] SAYANA, S. Anantha.: *Using CAATs to Support IS Audit.* Information Systems Control Journal, Volume 1, 2003. ISSN 15267407.

[10] SENFT, Sandra and GALLEGOS, Frederick.: *Information technology control and audit. 3rd ed.* Updated and rev. Boca Raton: CRC Press, c2009. ISBN 1420065505.

[11] LUTZ, Mark.: *Python: pocket reference. 4th ed.* Sebastopol, Calif: O'Reilly Media, 2010. p. 1. ISBN 9780596158088.

[12] Python Software Foundation Wiki Server: *Organizations Using Python - Python Wiki.* [online]. [cit. 10.01.2018]. Available from: https://wiki.python.org/moin/OrganizationsUsingPython

[13] LUTZ, Mark.: *Learning Python. 5th ed.* Sebastopol, Calif.: O'Reilly Media, 2013. ISBN 9781449355739.

[14] Medium: *Top 15 Python Libraries for Data Science in 2017.* [online]. [cit. 28.11.2017]. Available from: https://medium.com/activewizards-machine-learning-company/top-15-python-libraries-for-data-science-in-in-2017-ab61b4f9b4a7

[15] Computer Business Review: *How Python rose to the top of the data science world* [online]. [cit. 28.11.2017]. Available from: https://www.cbronline.com/big-data/analytics/python-rose-top-data-science-world/

[16] KLEENE, Stephen Cole.: *Representation of events in nerve nets and finite automata.* C. Shannon and J. McCarthy, (eds.) Automata Studies, Princeton University Press, NJ, 1956, 3–41.

[17] MCCULLOCH, W. S. and PITTS, W..: *A logical calculus of ideas immanent in nervous activity.* Bull. Math. BioPhys., 5, 1943, 115–133.

[18] The Law Dictionary: *What is GROSS BOOK VALUE?* [online]. [cit. 18.05.2018]. Available from: https://thelawdictionary.org/gross-book-value/

[19] DUCHAC, E. Jonathan et. al.: *Financial Accounting: An Integrated Statements Approach. 2nd ed.* Sebastopol, United States of America: Thomson South-Western, 2007. ISBN 0324312113.

[20] NEUMANN, Frank and WITT, Carsten .: *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity.* Springer-Verlag Berlin Heidelberg, 2010. ISBN 9783642165436.

[21] COPELAND, B. Jack.: *The Church-Turing Thesis* Stanford Encyclopedia of Philosophy, 2017. [online]. [cit. 20.05.2018]. Available from: https://plato.stanford.edu/entries/church-turing/

[22] TURING, Alan.: *ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM.* Proceedings of the London Mathematical Society, Volume s2-42, Issue 1, 1 January 1937, p. 230–265

[23] PAPADIMITRIOU, Christos.: *Computational Complexity.* Addison-Wesley Longman, 1994. ISBN 0201530821.

[24] HARTMANIS, Juris et. al.: *Sparse sets in NP-P: EXPTIME vs. NEXPTIME.* Information and Control, 65, pp. 158-181, 1985.

[25] CORMEN, Thomas H. et. al.: *Introduction to Algorithms. 2nd ed.* 3.1: Asymptotic notation. MIT Press and McGraw–Hill, 2001. ISBN 0262032937.

[26] KOLMOGOROV, Andrey.: *On Tables of Random Numbers.* Theoretical Computer Science, 1998: 207 (2): p. 387–395

[27] KodeKnight: *Hash Tables* [online]. [cit. 19.05.2018]. Available from: http://k2code.blogspot.cz/2013/08/hash-tables.html