The Structure of the thesis

1 Motivation

Every time requires it's own heroes. Nowadays, with a massive digitalization of the world, statistical analysis is not only used in spheres like classical production and distribution (logistics), but also everywhere on the web. In order to create a nice web-site, that will attract buyers and more important will make them purchase things from a particular on-line shop using this exact site, on-line shops require an analysis of a customer's behavior, specific demands and preferences. Another example where data analysis is essential is video games industry. The AI, which is built in an immense data source and is constantly analyzing the incoming data from the players, servers and third party sources (if needed), is used in many game-projects from a simple browser-strategy to "the most expected game of the year". The game play, the enemies, the load distribution on the servers and many other things are based on analysis and forecast. The are many-many other examples where data analysis is used.

Since the computer technologies are way more advanced, than they were several decades ago, the computations can be automatized, the data collection can be automatized. For every purposes certain tools, languages and libraries should be chosen carefully among the great choice. In this work we will concentrate our attention on a statistic-econometric task. On the whole, almost every language can be used to build a program for collecting, processing and visualizing data: SCALA, C, C++, .NET, Java, Python, JavaScript, R, etc.. However, some of the languages are more suitable for this purposes.

In this work we will concentrate on "easy" languages. We will compare Python 3 (Python 2 in form of some libraries) language with R language. Both of them are relatively new (Python is about 25 years old and R is about 22 years old) to the wide publicity. Both are currently used for data analysis (for Python it is not the only use, but as we said before, we will concentrate on this aspect). Both languages are widely used in open source projects and have a large community behind them. Also, very important that both of the languages are relatively easy to begin with also for non-programmers. Python has a gradual learning curve due to its simplicity, clear and intuitive syntax. Although R has a steep learning curve it is still pretty easy to use for small and not too complex projects (which is the case for this thesis).

1.1 Python language overview

The Python language is pretty young. It has first appeared about 25 years ago. As described on the official page in Wikipedia:"Python is a general-purpose, high-level programming language"[1]. That means that Python can be used for many tasks like back end development, front end development, data analysis and even for processor simulation (the python code will generate you VHDL code eventually). The language supports multiple programming paradigms: Object Oriented, Imperative and Functional programming.
Another big advantage of the language is being a cross-platform language executable via interpreter that can be installed on every operation system. Another possibility to run python programs without installing an interpreter is to package the program into stand-alone executable.
The language has an automatic garbage collector, so you don't have to worry about memory leaks by small to medium programs.
Enormous community stays behind Python and its variations. The standard library has more than enough functions for easy start. The language's syntax is very clear and intuitive in comparison to C++, C or even Java. I would say the language is similar to normal human speech but in more strict and logical form.
Another good thing about Python is build-in test - doctest module. This test is very easy to use to check and debug your code on the fly. The use of the module is intuitively clear even to beginners. You also don't keep the test's text in other file, which gives you certain level of an overview of the program. Unit test are also widely used among the programmers using Python language. There also other possibilities to debug and benchmark the code written in python, but we will not discuss it here.
The disadvantage of the language is its speed in comparison with C, C++ and Java. Also the visualization tools could be better, but since the language is a multipurpose language, it is normal, that some things are not top among the class.

1.2 R language overview

According to Wikipedia:"R is a programming language and software environment for statistical computing and graphics"[5]. R is a successor of the S language, purposed about 39 years ago. R is distributed under GNU General Public license, which makes this language a nice choice for an open source analytical project.
Same as Python, R also uses interpreter to execute the code. R language is pretty much single-purpose language. The main use of R is statistical analysis and visualization. Since R is an open source project, there is a huge community of real life practice and scientists behind it.
The disadvantage of the language is its one-purpose nature. Analytical project built with R will work with no doubt very fast and good, presenting good re-

sults. But the use of such project will be reduced, since there are very few quick and comfortable ways to integrate the program into the system.

1.3 Python vs R wars

Since both languages offer tons of packages and libraries for analyzing and visualization, people argue about what language is a better fit. You can find a lot of discussions about this topic on stackoverflow [6], stackoverflow analog for data science [7] and many other resources on internet [8],[9],[10]. So far nobody has managed to give a satisfactory final answer, since the languages indeed are different. However for every concrete task and scale we can give a list of requirements and compare the languages objectively leaving the subjective comparison aside.
Tho formal criteria to compare Python and R are:

1. The amount of useful resources for the task
2. Clear documentation with examples
3. Performance
4. Memory use
5. Appropriate data structures
6. Possibility to work with Big Data
7. Visualization tools
8. Hard or soft limitations
9. Need of workarounds

Using this list both languages can be objectively evaluated. These points are appropriate for all small-, middle- and big-projects. In next section we will describe the test-task to compare Python and R languages and define the points-system for objective evaluation.

1.4 Thesis description

The idea of this bachelor thesis is to test which language suits better specific statistic task. The program that will be evaluated consists of four parts:

1. Getting and formatting data - the program should be able to read a prepared file in csv format. Afterwards, the program should format the data into needed structure for further use.

2. Analyzing data - the program should be able to run statistical test to figure out data characteristics. This step is needed to determine what regression models are allowed for this data set. Following test will be presented: stationary test, build cdf and kde, find moments, distribution test (goodness of fit tests).

3. Building a model - the program should be able to create a proper model using a step-forward algorithm, set the limits on the number of the predictors and return the names of the parameters, regression parameters and some useful statistics.

4. Visualization - the program should be able to present the results and steps between if needed.

The program will build a simple regression if possible. This task is considered to be middle-size task, not too easy but at the same time not too time and knowledge consuming. The data will be prepared and provided as 2 files (training set and test set) in .csv format.
According to the list for objective evaluation, each point will give either 0 or 1 score to language. The 0 score will be given in negative case and score 1 will be given in positive case.

Useful resources - at least 2 different libraries for one task gives 1 point.

Documentation - examples, clean source code and clear structured APIs give 1 point.

Performance - fastest language gets 1 point

Memory - the program with the smallest memory use gets 1 point.

Data structures - no additional formatting needed gives 1 point.

Big Data - libraries, plugins and frameworks for big data give 1 point.

Visualization - easiness in use and customization gives 1 point.

Limitations - if there are any limitations, the language gets 0 points.

Workarounds - additional time to solve the task gives 0 points (deviation from the time for the naive implementation).

The results of the programs will be also compared. If they differ, the models will be cross-compared wit the other program or manually.
The task is to build a best matching linear regression (if possible) for a company (always the first column in the file) using certain limitations on the number of the predictors.

1.5 Data preparation

As was mentioned before the data set for this task will be prepared in advance. For our purpose we decided to take Intel as a dependent variable in the regression. Predictors will be chosen among the companies from the same market of micro controllers, supplier market and customers market.
The Intel's competitors can be found on Wikipedia listed in several tables for years from 1998 to 2013. We used a parser to get following list of manufacturers for the years 2000-2013 for semiconductors market:

```
['AMD', 'Qualcomm', 'Micron Technology','Hynix',
'Infineon Technologies', 'Intel Corporation',
'STMicroelectronics', 'Texas Instruments']
```

The semiconductors are the basic component for different devices, so the potential consumer-markets may vary. In this Bachelor we will concentrate on microchips (CPUs) consumers, following markets are: Tablets, Smart-phones, personal computers, automobiles, video game consoles, medical technologies, Engineering technologies, Aviation. Tablets and personal computers are united into one market-group.
Automobile market:

```
['Toyota', 'GM', 'Volkswagen', 'Ford', 'Nissan',
'Fiat Chrysler Automobiles', 'Honda', 'PSA', 'BMW',
'Daimler AG', 'Mitsubishi', 'Tata', 'Fuji']
```

The only significant players (manufacturers) on the game console market are: Microsoft, Sony and Nintendo.
The aviation market is presented by following companies:

```
['Boeing', 'United Technologies', 'Lockheed Martin',
'Honeywell International', 'General Dynamics',
'BAE Systems', 'Northrop Grumman', 'Raytheon',
'Rolls Royce', 'Textron', 'Embraer', 'Spirit AeroSystems Holdings Inc.']
```

The next market consuming microchips and other Intel-production is Smart-phones, Tablets and PCs. These gadgets are not substitutes, but they are all very similar in production-process, end-consumer and functions. Since many companies are presented on the markets mentioned above and have further production markets, we will put them into "Diversified" category. The diversified companies that may influence Intel are:

```
['Samsung', 'Apple', 'Microsoft', 'Nokia', 'Sony', 'LG',
'Motorola',  'Lenovo',  'BlackBerry', 'Alcatel', 'Vodafone']
```

Another huge consumer market for Intel is medical equipment market. The following companies present this sphere:

```
['Johnson & Johnson', 'General Electric Co.', 'Medtronic Inc.',
'Siemens AG', 'Baxter International Inc.',
'Fresenius Medical Care AG & Co.', 'Koninklijke Philips',
'Cardinal Health Inc.', 'Novartis AG', 'Stryker Corp.',
'Becton, Dickinson and Co.', 'Boston Scientific Corp.',
'Allergan Inc.', 'St. Jude Medical Inc.', '3M Co.',
'Abbott Laboratories', 'Zimmer Holdings Inc.',
'Smith & Nephew plc', 'Olympus Corp.', 'Bayer AG',
'CR Bard Inc.', 'Varian Medical Systems Inc.',
'DENTSPLY International Inc.', 'Hologic Inc.',
'Danaher Corp.', 'Edwards Lifesciences', 'Intuitive Surgical Inc.']
```

Additional to above mentioned companies several big players from the industrial equipment market will be added. These are following companies:

```
['ABB Robotics', 'Adept Technology', 'Bosch', 'Caterpillar',
'Denso Robotics', 'Google', 'Universal Electronics']
```

The list of the potential predictors is not full, because some pretty big players on the markets are left due to the lack of information. The reason of the information-lack is recent enter to the international stock exchange (since 2010 earliest). Some companies are still closed for the foreign investors (which is the case for such giants as Samsung, Honda and other Asian companies). The last limit on chosen companies is the trading-volume. We simply can not use the company's data for composing a regression, if the last year or two the trade volumes for the shares was 0.
After the companies were chosen, the whole data table will be split into 2 data sets: learning set and validation set. We have obtained daily prices from 2006 to 2015 (31.12.2014 is the last date for all indexes). The training or learning set will be approximately 70% from the whole data volume: from 2006 to 2010. The validation-set is about 30%: from 2011 to 2012. For this work we use daily frequency (only opening prices).

2 Python

2.1 The structure of the program

The program consists of three main classes, connected with each other in main class. The first class is called DataFormating.py and is responsible for getting data out of the csv file and writing it into an instance of a proper format for other two classes, also the dependent variable will be extracted from the whole data base (the dependent variable is always the first column in the csv table). The second class is called StatisticTests.py, it runs several test to find out main static characteristics in order to choose an appropriate model class for the data. The third class is called BuildModel.py. This class builds the multiple linear regression (according to the results of the StatisticTests class) using step-forward approach:

1. The amount of companies taken into account is reduced using correlation: all companies with correlation less than 30% are left out.
2. A limit on the maximum number of the parameters is set using following rule: 1 company out of 10 if number of company exceeds 5.
3. The first step is to find the best one parameter company using the highest correlation coefficient from the first step.
4. All possible combinations for fixed company from the previous step and left companies are created. For all these combinations the linear regression using Generalized Least Squares approach [`https://en.wikipedia.org/wiki/Generalized_least_squares`].
5. The best model among the class is chosen using AIC criterion.
6. Best models among the classes are compared using likelihood ratio test.
7. If the bigger model is better and the limit of the predictors in regression is not achieved, the new small model is equal to old big model and repeat from step 4. The final result is the last big model.
8. If the smaller model is better or/and the maximum number of parameters in the final regression is achieved, than the final result is either the small model or the last big model.

At the end the full information for the best model is returned.

2.2 The process description

In this section the process of the coding is described: what difficulties and problems are encountered and how they were solved.

2.2.1 Used Libraries

In the course of writing the program several steps were implemented. As was mentioned above in the "Program structure" section, these steps are: extracting and preparing the data for further use, check statistic characteristics and build the model based on the results of the statistics. In order to build the program three main libraries were used:

1. Base Python library[12].
2. SciPy resource (unites six different libraries, in our case three were used explicitly)[13].
3. Statmodels[14].

The functions and data structures defined in the base library was used for every simple task, except simple mathematical computations, since there were faster implementations.
The second most used library in this work is NumPy,this library is the part of the SciPy source. NumPy offers many fast multi dimensional computations and assosiated multi dimensional structures. In our work we used this library to compute several statistic and build data for displaying graphs. As we mentioned before, the built-in library was not fast enough for our computation (for bigger data arrays numpy is almost not an arguable choice):

```
>>> x=[random.randint(0,10) for i in range(0,1500)]
>>> len(x)
1500

>>> mean_base = statistics.mean(x)
>>> mean_base
4.966
>>> cProfile.run('statistics.mean(x)')
         4531 function calls in 0.006 seconds

>>> mean_numpy = numpy.average(x)
>>> mean_numpy
4.9660000000000002
>>> cProfile.run('numpy.average(x)')
         17 function calls in 0.001 seconds
```

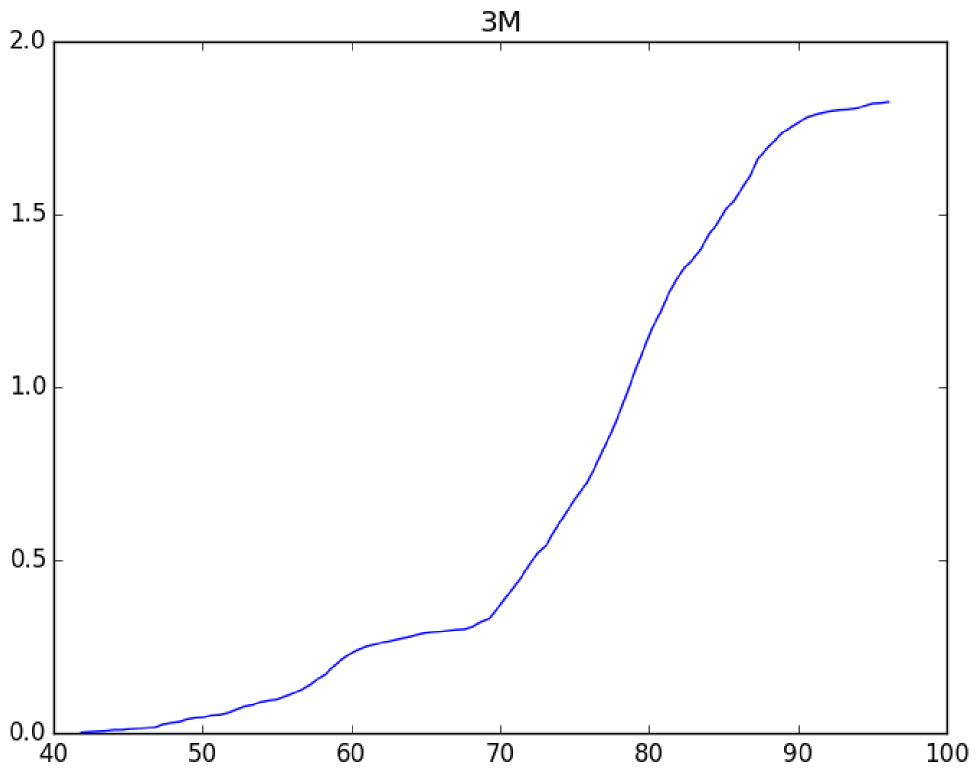
As you can see, the NumPy implementation of the function "average" is more accurate and efficient (4531 intern calls vs 17 intern calls). The run time difference can be already shown on the list containing 1500 elements.
The third library with fundamental use is Pandas, this library is also a part of the SciPy source. This library provides high-performance data structures and associated functions for data analysis. This library can be used for generating, accessing and formatting data. In this work we used to get the data from our

.csv file in a intuitive manner and to avoid unnecessary formatting steps. Also Pandas offers you a functionality to get the data directly from web sites like: Yahoo Finance, Google Finance, Google Analytics, etc.., although we didn't used it in our program. Fo example the method we used to read the data and returned it in Data.frame format, which can be easily sliced according to the needs of the program or formatted.

```
data = pd.read_csv(str(self.file_name))
cProfile.run('data = pd.read_csv("../data/LearningSet.csv")'
    3316 function calls (3238 primitive calls) in 0.024 seconds
```

The LearningSet file is a table containing 78 companies with 1240 prices for each company, 477,6KiB.
The library used for drawing and saving graphs is very popular and for many other libraries is a default use. Matplotlib is also a part of the SciPy source. It offers you different types of graphs and formatting tools.

```
plt.title(3M)
plt.plot(bin_edges[1:], cdf)
plt.clf()
```



The last but the fundamental library is Statmodels. This library offers a wide variety of tools and models for statistic purpose. Two main functions of the library used in our program are: Augmented Dickey-Fuller test for unit root and building a multiple linear regression with generalized least squares approach.

```
statistic = statmodel.adfuller(self.dict_data[key], 250,
                         'ctt', 't-stat', False, False)[0] - stationary test

model_null = sm.GLS(y, smaller_model_list)
info_small_model = model_null.fit().summary()
```

2.2.2 Documentation quality and quantity

The quality and the quantity of the documentation plays a crucial role for the project of every complexity, but especially for beginner levels. In this work we will evaluate the documentation of the languages and connected libraries, packages, frameworks using score system. The languages can get points for each position criterion given in the table:

1. The documentation is always available.
2. The documentation is easy to navigate.
3. The documentation offers a sufficient amount of examples.
4. The examples given in the documentation cover at least two complexity levels (a very simple use and use in an advance context).
5. The documentation provide a link to the source code (mainly for non-base packages and libraries).
6. For scientific libraries the theoretical background is provided.
7. The documentation is easy to read (formatting, colors, description of the parameters and output).

Originally, Python language was used a lot by the scientific society and many libraries were also implemented for scientific purpose by people with a certain background. In this work we have used base Python 3 library [12], SciPy[14] and Statmodels[15]. SciPy unites several packages, only three from the whole list were used in our Python program: NumPy, Matplotlib and Pandas.
The Python program was written in Python 3 version, in this work we did not encountered any problems with the version conflicts. But it is possible that some external libraries are build on Python 2 and thus can cause some issues. In this case you can use Six library[16] to guarantee the compatibility of the versions. As we said, our program did not require use of this library, but it is safe to keep it in mind.
According to the provided evaluation table all four libraries will get points and the weighted (approximate weights are based on appearance frequency and

importance of the library in the program) score will be added to the python total score.
The base Python library:

1. 1 Point (the documentation was always available, no delays over period of 6 month were noticed).
2. 1 Point (the web site is modern and have a very clear structure).
3. 1 Point (each function has one simple example, some functions are explained by "identical function" example, at the end of the class description composite example is given).
4. 1 Point (see above).
5. 0 Point (the code is mostly not in a free access, mostly normal for base libraries).
6. 1 Point (functions are either explained in the documentation, or provide a link to an external source, for example for math.gamma(x) the description link sends you to the Wikipedia page for gamma function).
7. 1 Point (the Structure of the documentation for every class is the same and mostly intuitive. You can navigate through the documentation using different methods: global index, glossary, etc.).

From the SciPy source three libraries were used explicitly: NumPy, Matplotlib and Pandas. The documentation of the libraries contain more detailed information and some additional examples, but the structure and the formatting are the same according to python styling trend. So the SciPy documentation will be evaluated.

1. 1 Point.
2. 1 Point.
3. 1 Point.
4. 1 Point.
5. 1 Point.
6. 1 Point.
7. 1 Point (similar styling pattern used for all libraries).

The most important library for our python program was Statmodels, since we used it to build the models. Although the library itself is very powerful, it is not easy to use and to understand.

1. 0 Point (the documentation is hosted on Sourceforge and the server was down for about a week during the program development).

2. 0 Point (too many sub classes and sub methods, which importance you can not evaluate from the first view).

3. 1 Point.

4. 0 Point (basic examples are not always provided by the documentation).

5. 1 Point (source code is hosted on github, thus no issues with availability, but very hard to read).

6. 1 Point.

7. 0 Point (the explanation order of the input parameters, output parameters and methods is alphabetical and not logical, the explanation is sometimes not sufficient).

So the final score for the python documentation is:

$$\frac{(\frac{2}{5}(6) + \frac{1}{5}(7) + \frac{2}{5}(3))}{7} = 0.71$$

2.2.3 Advantages and disadvantages of the program

There are several advantages and disadvantages of the Python language for statistic tasks.
The first advantage is small time investment needed up front. The Python language is intuitive and does not require deep understanding of the programming paradigm for simple programs, thus Python is great for prototyping and checking small hypothesis. However this is a temporary advantage, since for more complex tasks you will have to use complex libraries and structures, which require deeper understanding of the language.
The second really important advantage of the python language is testing. Python has built-in doc-tests [22], that are very easy to use. Another test-possibility are Unit Tests [23]. Python language gives the programmer different ways to test and to debug the program in order to avoid possible computational and logic errors. We will not discuss this theme in details, since the tutorials and the documentation are clear and offer sufficient amount of examples.
Another advantage of the python is its multipurpose nature, although we didn't use it for our program it is still important for the world outside of the analysis field.
There are several disadvantages of the statistic program written in python.
The first and the biggest disadvantage is the performance. This disadvantage appears only for beginners by implementing mid-difficult task with mid-size data structures. For large data sets python will be not the best option.
The main time consumption comes from StatisticTests.py, specifically from stationarity test.

```
>>> cProfile.run("statistics.stationarity()")
        2898213 function calls (2887173 primitive calls) in 5408.838 seconds
```

other run time profiles

Another small disadvantage of the program written in python is the time you have to invest into the language investigation for more complex tasks. In order to save memory or make the performance better we had to drop naive implementation and read the documentation more carefully (although the language is intuitive on the surface it is not that simple). For example:

```
>>> a = {"a":[1,2,3], "b":[1,2,3], "c":[1,4,9]}
>>> b = a
>>> b["a"] = [1,4,9]
>>> b
{'b': [1, 2, 3], 'a': [1, 4, 9], 'c': [1, 4, 9]}
>>> a
{'b': [1, 2, 3], 'a': [1, 4, 9], 'c': [1, 4, 9]}
```

Although we haven't expected dictionary to change. Python does not implicitly creates copies. The example above shows how two objects are referred to the same object. Thus during mutating one of the objects, all references will change in order to keep referring to the object in its current state. This can be crucial for the program, since we have used one originally formatted dictionary for different purposes. In order to avoid this error you need to do following thing:

```
>>> a = {"a":[1,2,3], "b":[1,2,3], "c":[1,4,9]}
>>> b = a.copy()
>>> a
{'b': [1, 2, 3], 'a': [1, 2, 3], 'c': [1, 4, 9]}
>>> b
{'c': [1, 4, 9], 'b': [1, 2, 3], 'a': [1, 2, 3]}
>>> b["a"] = [0]
>>> b
{'c': [1, 4, 9], 'b': [1, 2, 3], 'a': [0]}
>>> a
{'b': [1, 2, 3], 'a': [1, 2, 3], 'c': [1, 4, 9]}
```

or

```
>>> a = {"a":[1,2,3], "b":[1,2,3], "c":[1,4,9]}
>>> b = dict(a)
>>> b
{'c': [1, 4, 9], 'b': [1, 2, 3], 'a': [1, 2, 3]}
>>> b["a"]=[0]
>>> b
{'c': [1, 4, 9], 'b': [1, 2, 3], 'a': [0]}
>>> a
{'b': [1, 2, 3], 'a': [1, 2, 3], 'c': [1, 4, 9]}
```

Such thing can confuse not only the very beginners, but also advanced programmers, because you need to keep every thing in mind.

One more small disadvantage is the unsorted nature of the dictionary. After we have read the csv table and formatted it into dictionary, where keys are the companies names and values are the associated prices lists. The keys appear in dictionary in random order, so we can't iterate over the keys directly and the first key will not be the first company from the table. Additional commands will be needed. But this disadvantage will mostly appear for the very beginners and the very beginning of the program, since the workaround is intuitive.

2.3 Results

The end result is currently printed out to a console. The final message contains the names of the companies used in the final regression, main characteristics of the regression (AIC, BIC, Loglikelihood, coefficients and errors, etc.) and the total run time of the program. The results of the program for the given data, described in the data preparation section are:

```
The best model contains  6  parameters. And the model is:
['STMElectro', 'Olympus', 'St Jude', 'Lenovo', 'MicronTech', 'Google']
                            GLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.998
Model:                            GLS   Adj. R-squared:                  0.998
Method:                 Least Squares   F-statistic:                 8.701e+04
Date:                Do, 24 Sep 2015   Prob (F-statistic):               0.00
Time:                        20:20:45   Log-Likelihood:                -1753.3
No. Observations:                1239   AIC:                             3519.
Df Residuals:                    1233   BIC:                             3549.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
x1             0.1254      0.016      7.933      0.000         0.094     0.156
x2             0.0609      0.012      5.065      0.000         0.037     0.085
x3             0.3261      0.016     21.029      0.000         0.296     0.357
x4             0.2126      0.004     47.272      0.000         0.204     0.221
x5             0.0083      0.001     12.929      0.000         0.007     0.010
x6             0.1291      0.012     10.536      0.000         0.105     0.153
==============================================================================
Omnibus:                       10.831   Durbin-Watson:                   0.120
Prob(Omnibus):                  0.004   Jarque-Bera (JB):               10.859
Skew:                           0.212   Prob(JB):                      0.00439
Kurtosis:                       2.827   Cond. No.                         357.
==============================================================================

[Finished in 4173.2s]
```

To sum up the objective arguments of using python for writing a small program to compute a linear regression for a given data and restrictions, we will sum up the points from the table from the thesis description section. The points for the Python program are:

1. Useful sources: 1 Point.
2. Documentation: 0.71 Point.
3. Performance: 0 Points.
4. Memory: 0 Points (Since the data format for different packages is different, our program have used a combination of different data structures to store all formats).
5. Uniform Data structures: 0 Points (different libraries uses different shapes).
6. Big Data: 1 Point (Python have tools for Big Data analysis, or can be used with known tools like Hadoop, Cassandra, Pig, Hive, etc.).
7. Visualization: 1 Point (several libraries also for displaying the chart in web).
8. Limitations: 1 Point (accept time to actually learn the language there are no limitations).
9. Workarounds: 1 Point (for this particular task there have always been a method or several wrap functions).

The Python program gets 5.71 points in total.
The subjective arguments for and against are:

1. Code is easy to understand, since syntax is much like simple English.
2. The data structures are exactly same as you imagine
3. Slicing of the data arrays is normally intuitive (unless you have a multi-dimensional custom structure).
4. For small data arrays the language is fast enough even without tricks.
5. Debugging messages are easy to understand.

3 R

3.1 The structure of the program

The program written in R language has the same structure as the Python program. The R program consists of three classes: DataFormatting.r, StatisticTests.r and BiuldModel.r. Their functions are similar to the same functions from the first program. However there are slight differences in the inner functions of the classes. The main class is not implemented, since we wanted more control over the program flow and thus have called all functions and imported classes via console in R run time.
The R language is very convenient in terms of data formatting - you don't need it. The object called Data.frame is returned when using read.csv() from the base package. This object has methods that allow the programmer to use the data from different spots in the program and for different purposes directly without extracting it and saving in a proper format. For example the list of all companies from the table was extracted using the function "colnames(data.frame)" as an output we get the list object containing the elements from the first row of the table. So our R program does not have inner helpers - converter functions, thus is a bit easier to understand.
The The StatisticTests.r runs tests to detect statistic characteristics of the data: Kolmogorov-Smirnov goodness of fit distribution test, Augmented Dickey-Fuller test to detect unit root, build cumulative distribution function and kernel density function, find moments (mean, standard deviation, skew, kurtosis).
The third class, called BuildModel.r, creates the multiple linear regression for given data table following the same steps from the step-forward approach that was used in the Python program:

1. Cut off all companies with correlation less than 30%.
2. Set the limit for the maximum number of parameters in the final regression.
3. Choose the best one parameter model.
4. Build all possible tuple-combinations with fixed parameter.
5. Choose the best model among the two parameter class.
6. Compare the best models from two different classes via Likelihood Ratio Test.
7. Repeat until the smaller will be better or/and the maximum number of parameters is reached.

3.2 The process description

In this section the process of the program writing is described in more details: what packages were used, what difficulties appeared, etc..

3.2.1 Used Libraries

For this R program following packages were used:

1. Base package[16].
2. Stats package[17].
3. Nlme package[18].
4. Tseries package[19].

The first base library was used for all basic computations like moments, correlation vector, etc.. This package provides many different functions for formatting the data. For example:

```
formula <- as.formula(paste(dep$name,"˜", as.character(names(small_model)), collapse=""))
```

This function converts the string object from the brackets into a formula object that is further used as an input parameter for building a linear regression using GLS method. The base package offers basic math operation as well.
The StatisticTests.r class uses both stats and tseries packages to build statistic tests on the data. The tseries package offers many different tests and methods used in computational finances and time series analysis. In this work we have only used Agumented Dickey-Fuller test to test the data for stationarity:

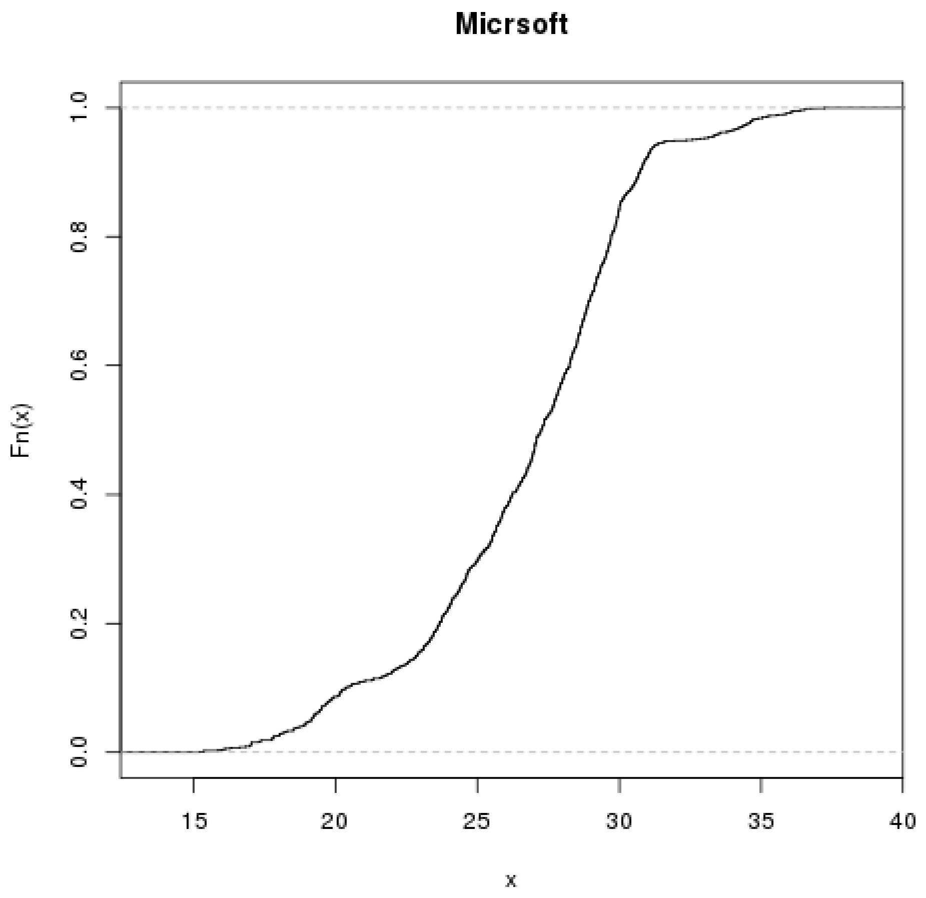
```
test <- adf.test(data[[company]])
```

The second package used in StatisticTests class offers functions to run simple statistic tests on the data and build simple characteristics. In our work we have used the stats package to build graphs for each company from the data table and test the data for its distribution. The program has yields the same results as the python program.
The fourth package used in this program offers the the possibility to build linear regression using given parameters. The use of the functions from the library are very intuitive:

```
small <- gls(as.formula(paste(dep$name,"˜", as.character(names(small_model)), collapse=""))
```

Where the first parameter is from the class formula looks like string object: "Intel Olympus + St.Jude". The second parameter is the data, where the variables from the formula match the names of the columns from the data frame object.
The graphics is provided by the graphics library, but it comes with the default R set up. The plotting in R is easy to use:

```
png(path_cdf)
plot(graph[[name]], verticals = TRUE, do.points = FALSE, main=name)
dev.off()
```



3.2.2 Documentation quality and quantity

In comparison to the Python language, R was build solely for analytical purposes. The most part of the contributors are scientific institutions and scientists, and thus the R language documentation is written in a scientific style. To evaluate the R language documentation we will use the points system described in Python Documentation section.
The R Base package:

1. 1 Point (the documentation is hosted by ETH Zürich and no server downtime was noticed).

2. 0 Points (if you don't know exactly for what function you are looking for, alphabetic search not going to be much of a help).

3. 0 Points (the native documentation does not provide sufficient amount of examples).
4. 0.5 Points (depends on the function).
5. 0 Points.
6. 1 Points (the literature source is given but not in form of the link).
7. 0 Points (no styling applied).

The Stats package is organized similar to the Base package and thus have similar points:

1. 1 Point (also hosted by ETH Zürich).
2. 0 Point (the alphabetical list with short description).
3. 0.5 Points (depends on function, demo files are offered only for some of the functions).
4. 0.5 Points (depends on the function).
5. 0 Point.
6. 1 Point.
7. 0 Points (no styling applied).

The Nlme package is hosted by the biggest R source called "CRAN". This source offers significant amount of simple and complex demos, test data sets and tutorials, etc.. CRAN is considered to be the central R source.

1. 1 Point (the CRAN servers are constantly monitored, no problems were noticed within a 2 year period).
2. 1 Point (the documentation is organized as a scientific paper with the content provided).
3. 1 Point.
4. 1 Point (examples are sometimes united for different functions).
5. 0 Points.
6. 1 Point (the theoretical background is provided in the form of the references after the function explanation).
7. 1 Point (scientific paper formatting).

The last package used in our program is Tseries:

1. 1 Point (hosted by CRAN).

2. 1 Point (scientific paper).
3. 1 Points.
4. 1 Point.
5. 0 Point.
6. 1 Point (in the form of references).
7. 1 Point (scientific paper formatting).

The quality of the R documentation is in average on the same level for all sources, but the convenience of the documentation (formatting, navigation) can differ a lot depending on the source. The total score for the R documentation is:

$$\frac{\frac{3}{10}(2.5) + \frac{1}{10}(3) + \frac{3}{10}(6) + \frac{3}{10}(6)}{7} = 0.66$$

3.2.3 Advantages and disadvantages of the program

Every programming language has its advantages and disadvantages. During writing our second program we have encountered several difficulties assosiated with the language features. Since we have already written our program, we decided to implement the same structure and thus have escaped difficulties choosing proper classes, structure of the program, functions, etc..
The first and a very important advantage of the R program is the data.frame object and associated functions. This object can be formatted or sliced within very small time from every function, the results of the of the slicing can be assigned needed format. One of the most useful functions during the work with R data objects was str() that shows you the structure of the object:

```
> str(data)
'data.frame':   1239 obs. of  69 variables:
 $ Intel               : num  26.6 26.6 26.4 25.8 25.9 ...
 $ AMD                 : num  27.5 28.4 27.8 27.9 29.2 ...
 $ Qualcomm            : num  46.5 45.6 45.4 43.8 44.2 ...
 $ MicronTech          : num  13.5 13.6 13.6 13.6 13.9 ...
 $ Infenion            : num  9.26 9.38 9.41 9.3 9.25 9.23 9.21 9.23 9.26 9.19 ...
 $ STMElectro          : num  18.5 18.7 18.6 18.1 18.2 ...
...
```

The slicing is very useful, when you know how exactly to handle it. For instance, to get all the companies names from the table you can use colnames():

```
> names<-colnames(data)
> class(names)
[1] "character"
> str(names)
 chr [1:69] "Intel" "AMD" "Qualcomm" "MicronTech" "Infenion" ...
```

In comparison to the Python program, where we have to store the data into dictionary with the names as the keys and the lists of share prices as the associated values, the names in the data.frame and other formats are coming in exact same order as they appear in the table. This feature make it very easy to extract the dependent variable, also you will have to know only the name of the variable for further computations.
The second advantage of the program is closely connected to the the function we have used to build the regression. As we described in the "Program Structure" and "Used Libraries" sections, for the regression we have used gls function from the nlme package. The use of the function is intuitive: you provide the formula for the regression and the data containing all given names from the formula (optional if the environment already have the data set loaded). Below two examples of the gls use are presented (these examples are not an optimal decision, but a show case):

```
> model1<-gls(Intel ~ Olympus + Google + AMD + St.Jude, data)
> model1
Generalized least squares fit by REML
  Model: Intel ~ Olympus + Google + AMD + St.Jude
  Data: data
  Log-restricted-likelihood: -1959.999

Coefficients:
(Intercept)     Olympus      Google         AMD     St.Jude
-1.38174717  0.18731627  0.01450565  0.05510854  0.21830586

Degrees of freedom: 1239 total; 1234 residual
Residual standard error: 1.158712
>
> model2<-gls(Intel ~ sin(pi*Olympus) + cos(2*pi*AMD) + St.Jude, data)
> model2
Generalized least squares fit by REML
  Model: Intel ~ sin(pi * Olympus) + cos(2 * pi * AMD) + St.Jude
  Data: data
  Log-restricted-likelihood: -2875.022

Coefficients:
      (Intercept) sin(pi * Olympus) cos(2 * pi * AMD)           St.Jude
       1.78264194        0.19693369        0.03505161        0.46976204

Degrees of freedom: 1239 total; 1235 residual
Residual standard error: 2.451844
```

Another positive moment about gls function is its data use. In comparison to python you don't have to format it and separate the data for chosen variables from the rest of the data object. You can use the data.frame loaded into the environment or use external data source, for example, for predictions. If y the

formula doesn't composed automatically, the only thing you have to care about is matching of the variable names from the formula and names in data.frame. The third advantage of the R program is its performance. Without explicit multiprocessing or JIT (just in time compiler is not used in this work), the program runs within a minute.total run time
The first and probably main disadvantage of the R language is its difficulty. The syntax is not intuitive, the data structures and their use are non trivial. In order to begin using R you will have to look through several tutorials and maybe full introduction course. The R background of the author of this work consisted only of the coursera "R programming" course [20], which didn't cover the whole program structure and the course was using RStudio [21]. RStudio is a good IDE for R to begin with, but if you need more control and understanding, you should switch to the console and a simple IDE to write the classes.
The first example of the R complexity is the absence of the classes in their normal understanding. In R there are three type of classes: S3, S4 and Regference Class. The last one is close to normal Python classes. The first two are repsenting the attribute of an object (in R everything considered as an object). In our program we only used the Reference class and didn't dive into the other two classes. More information on OO systems in R you can visit [22].
The second disadvantage of the R program is connected to Reference Class definition. In R you can not return multiple instances. Following implementation will return an error:

```
getData <- setRefClass("getData",
    fields = list( name="character", data="data.frame", companies="character"),
methods = list(
     get_data = function() {
            data <<- read.csv(name, header = TRUE, sep = ",",
            quote = "\"", dec = ".", fill = TRUE, comment.char = "")
companies <<- colnames(data)

return data, companies
})
```

In order to be able to return several objects you will have to unite them into list:

```
getData <- setRefClass("getData",
    fields = list( name="character", result="list"),
methods = list(
     get_data = function() {
            data <- read.csv(name, header = TRUE, sep = ",",
            quote = "\"", dec = ".", fill = TRUE, comment.char = "")
companies <- colnames(data)

            result <<- c(data, result)
            result<<- c(companies, result)
            return result
})
```

The smallest disadvantage of this workaround is additional variables assignment of the returned list in order to get needed values separately:

```
> d <- getData(name="../data/LearningSet.csv")
> result <- d$get_data()
>class(result)
[1] "list"
> data <- result[[1]]
> companies <- result[[2]]
```

Another disadvantage of the R program in the beginning is the slicing. On one hand, you can get the element or a data object using different approaches. On the other hand, they do not always return what you have expected and does not seem intuitive. For example let's get first 5 prices of Intel:

```
> data$Intel[1:5]
[1] 26.59 26.58 26.37 25.80 25.85
> class(data$Intel[1:5])
[1] "numeric"
>
> data[[1]][1:5]
[1] 26.59 26.58 26.37 25.80 25.85
> class(data[[1]][1:5])
[1] "numeric"
>
> data[names(data)[1]][[1]][1:5]
[1] 26.59 26.58 26.37 25.80 25.85
> class(data[names(data)[1]][[1]][1:5])
[1] "numeric"
>
> data[,1][1:5]
[1] 26.59 26.58 26.37 25.80 25.85
> class(data[,1][1:5])
[1] "numeric"
```

Slicing is a powerful tool in R, but for beginners it might be a problem to use it properly.

3.3 Results

4 Results

Our R program does not have the main class as in Python program, however main function is implemented, where all libraries are activated, classes are imported and the final calculation is run. The output is printed out to terminal from which the program was started. This is the extended output for gls function: summary.gls:

```
> source("main.r")
> system.time(main())
[1] "The parameters limit is not reached, searching further"
[1] "The parameters limit is not reached, searching further"
[1] "The parameters limit is not reached, searching further"
[1] "The parameters limit is not reached, searching further"
Generalized least squares fit by REML
  Model: as.formula(form[[k1]])
  Data: data
       AIC      BIC    logLik
  3421.976 3462.907 -1702.988

Coefficients:
                Value  Std.Error   t-value p-value
(Intercept)  4.588765 0.29835356  15.38029       0
Cardinal    -0.073992 0.00540346 -13.69342       0
Olympus      0.054461 0.01080308   5.04127       0
St.Jude      0.165351 0.00801178  20.63845       0
Lenovo       0.500866 0.01499778  33.39599       0
MicronTech   0.073363 0.01235370   5.93858       0
STMElectro   0.435386 0.02655304  16.39685       0

 Correlation:
           (Intr) Cardnl Olymps St.Jud Lenovo McrnTc
Cardinal   -0.208
Olympus    -0.492  0.210
St.Jude    -0.846 -0.113  0.212
Lenovo      0.447 -0.046 -0.834 -0.408
MicronTech -0.421  0.252  0.267  0.273 -0.180
STMElectro  0.494 -0.802 -0.590 -0.203  0.415 -0.614

Standardized residuals:
       Min         Q1        Med         Q3        Max
-2.3116653 -0.7048518 -0.0818459  0.6500465  3.5270072

Residual standard error: 0.9389829
Degrees of freedom: 1239 total; 1232 residual
```

```
   user  system elapsed 
 24.566   0.000  23.461 
```

As you can see, the best model yielded by the R program consist of 6 companies: STMElctro, Olympus, St.Jude, Lenovo, Microntech and Cardinal. It is interesting, that on the last step of the evaluation the algorithm have chosen Cardinal over Google (which have chosen the Python Program). Since the programs are identical this deviation can be caused by the underlying implementation in gls function or llr test, or AIC criterion evaluation. Unfortunately we haven't found the source code for R and thus we couldn't compare the implementations. This is why in the section Results we will compare the predictions of the programs. And add points to the program yielding closer predictions.
To sum up the objective arguments of using R for a small program to compute a linear regression for given data and restrictions we will give the points according to the evaluation table given in the description section. The Points for the R program are:

1. Useful sources: 1 Point.
2. Documentation: 0.66 Points.
3. Performance: 1 Point.
4. Memory: 1 Point (since no additional formatting was used the memory use for our specific case is less for R).
5. Uniform data structures: 1 Point (data.frame object was used everywhere directly).
6. Big Data: 1 Point (this is a relatively recent addition to the language, but there are several sources and libraries allowing to work with big data [23]).
7. Visualization: 1 Point (on the fly changing the graph makes it easy to work with).
8. Limitations: 1 Point (same as by Python language, after investing time into getting into R language there were noticed no limitations).
9. Workarounds: 1 Point (the program was build strait forward without having any issues).

The total objective score for R language is 7.66 points total.
The subjective arguments are:

1. Debugging messages are hard to understand.
2. Certain amount of time is needed to understand the code, since syntax is not completely intuitive.

3. Layered structures can very challenging to work with.
4. Slicing is very easy to work with, if you have understood it.
5. More.

4.1 Objective comparison

In this section the final comparison of the programs written in two languages will be presented. Here we will explain the difference in the scores and also check the accuracy of the predictions both programs yield.
The first difference between programs is the performance. In order to show the difference in the run time of the programs we will use system.time() for R and cProfile for Python (the examples of the code were already given above).

```
> system.time(data_read())
   user  system elapsed
  0.100   0.000   0.085
> system.time(stat(build_data, companies))
   user  system elapsed
  3.627   0.000   3.465
>> system.time(model(build_data, companies, rest, dependent))
   user  system elapsed
  3.837   0.000   3.067
```

The main function consists of three parts: read the data from the csv table and format it, the second part is the statistic test and the final part is building a model based on the results of the statistic test. And the overall time of the program can be measured via two functions: main and predict:

```
> system.time(main())
   user  system elapsed
  8.917   0.000   6.546
> system.time(build_predictions())
[1] "the mean of the y is: "
[1] 23.46462
[1] "the std of the y is: "
[1] 2.55532
[1] "the mean of the predictions is: "
[1] 20.86923
[1] "the std of the predictions is: "
[1] 1.544088
   user  system elapsed
  0.000   0.000   0.117
```

The run time of the main function is greater than the sum of its parts, since the main function loads needed libraries and also counts the time for the data transfer between the classes as well as the initialization of the classes.

The Python program performance is not that impressive as by the R program:
here compare overall run time, 3 main parts run time
The second difference is the memory used by each program. There are several possibilities to measure the memory load of the programs. In R you can use Rprofmem() from the utilities package to determine how much memory was used by the object (in R everything is considered as an object). In Python different tools and built-in classes can be used to do the profiling for example memory_profile [11]. However in this work we will use the unix time command from the console for both programs in order to get more accurate results. The printed message contains many details, but we will be interested in the Maximum resident set size, which means the amount of memory belogning to the process and currently presented in RAM.
The R memory use looks as following:

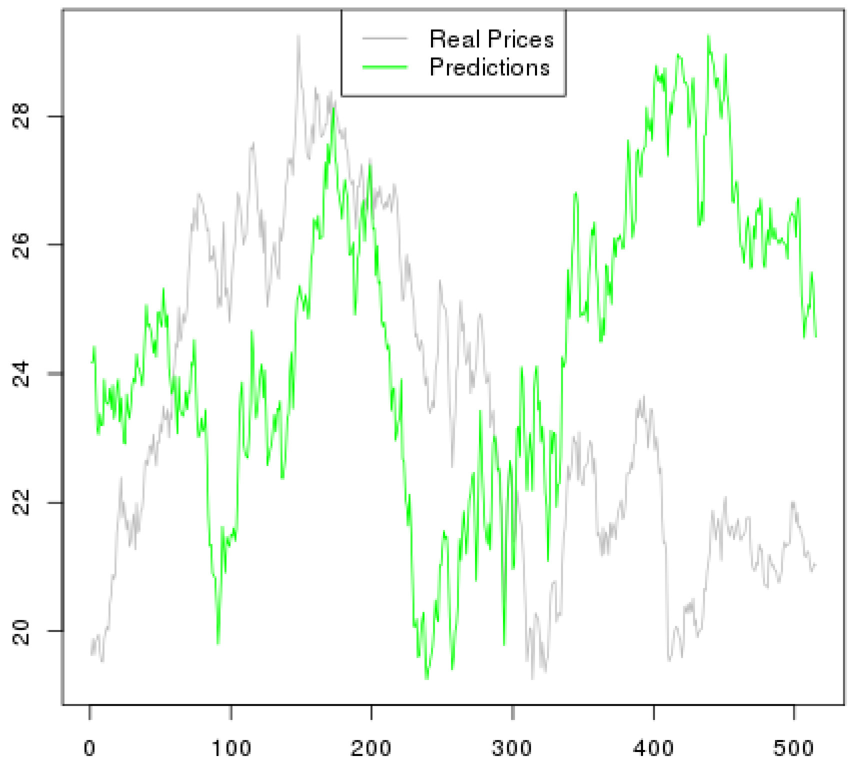
```
[20:32:19 - 15-10-25]
/home/alisa/uni-stuff/Bachelor/r % /usr/bin/time -v R  main.r
        Command being timed: "R main.r"
        User time (seconds): 7.41
        System time (seconds): 0.05
        Percent of CPU this job got: 18%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:41.45
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 78436
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 17697
        Voluntary context switches: 59
        Involuntary context switches: 2026
        Swaps: 0
        File system inputs: 0
        File system outputs: 2840
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

The maximum memory usage of the R program during running is 78,4 MB.
The Python memory use looks as following:

```
[19:11:27 - 15-10-25]
/home/alisa/uni-stuff/Bachelor/python % /usr/bin/time -v python __main___.py
        Command being timed: "python __main___.py"
        User time (seconds): 5564.19
        System time (seconds): 7.73
        Percent of CPU this job got: 100%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 1:32:46
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 637036
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 3633280
        Voluntary context switches: 3047
        Involuntary context switches: 22080
        Swaps: 0
        File system inputs: 0
        File system outputs: 8976
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

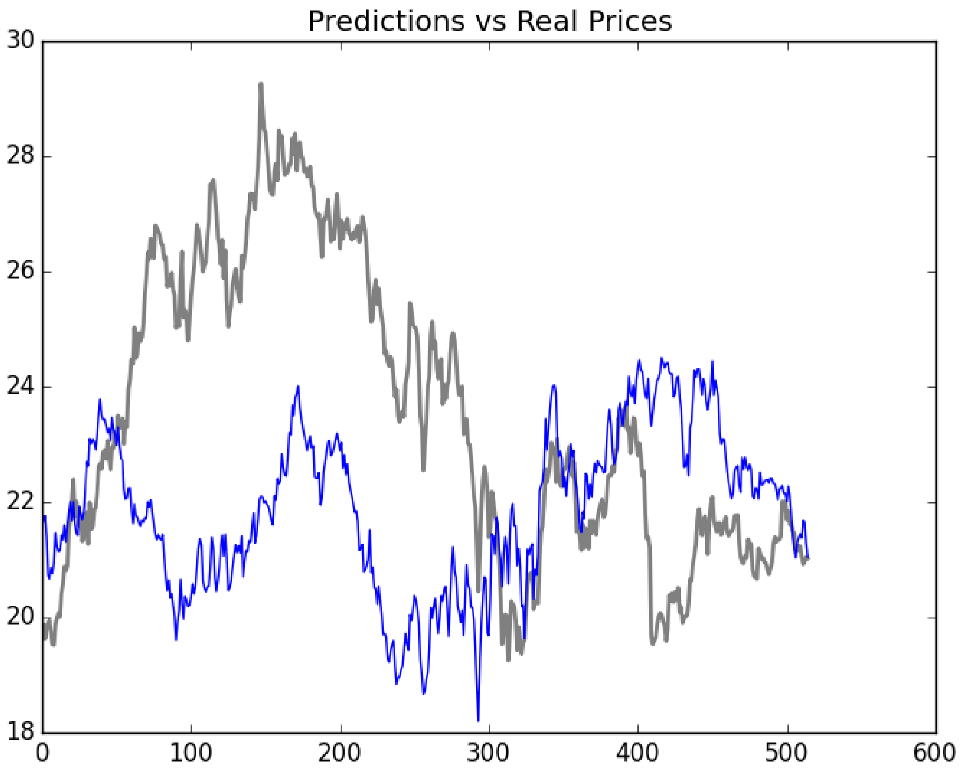
The maximum memory usage of the running Python program is 637,0 MB.
Since the final models from two programs differ, we will compare the predictions. Predictions were built on the TestingSet.csv containing 515 Prices for the time period from 1.01.2011 to 31.12.2012. We will show the main characteristics of the predictions row: mean and standard deviation as well as the comparison plot.
The R program gives us following results:

```
> p<-build_predictions()
[1] "the mean of the y is: "
[1] 23.46462
[1] "the std of the y is: "
[1] 2.55532
[1] "the mean of the predictions is: "
[1] 20.86923
[1] "the std of the predictions is: "
[1] 1.544088
[1] "the mean absolute error of the predictions is: "
[1] 3.29319
[1] "the root mean squared error of the predictions is: "
[1] 4.132798
```

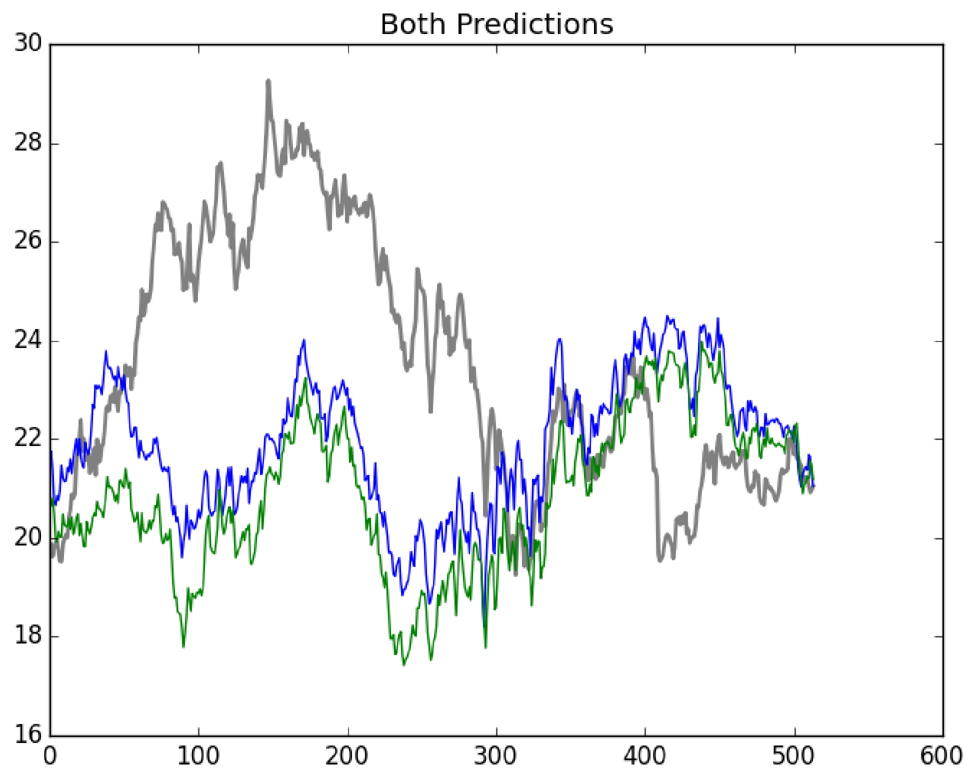


The Python program results are:

```
>>> build_predictions()
the mean of the y is: 23.4646213592
the std of the y is: 2.55283813168
the mean of the prediction is: 21.8848266352
the standard deviation is: 1.35949376438
the mean absolute error of the prediction is: 2.83819120505
the root mean squared error is: 3.47169327209
```



As you can see, the Python prediction statistically speaking is better, than the R prediction. Both mean absolute error and root mean square error are smaller by the Python results. The Python program gets 1 additional point for the relative accuracy. On the following graph you can see both predictions together, where the R forecasts have green color and blue color belongs to the Python forecasts.

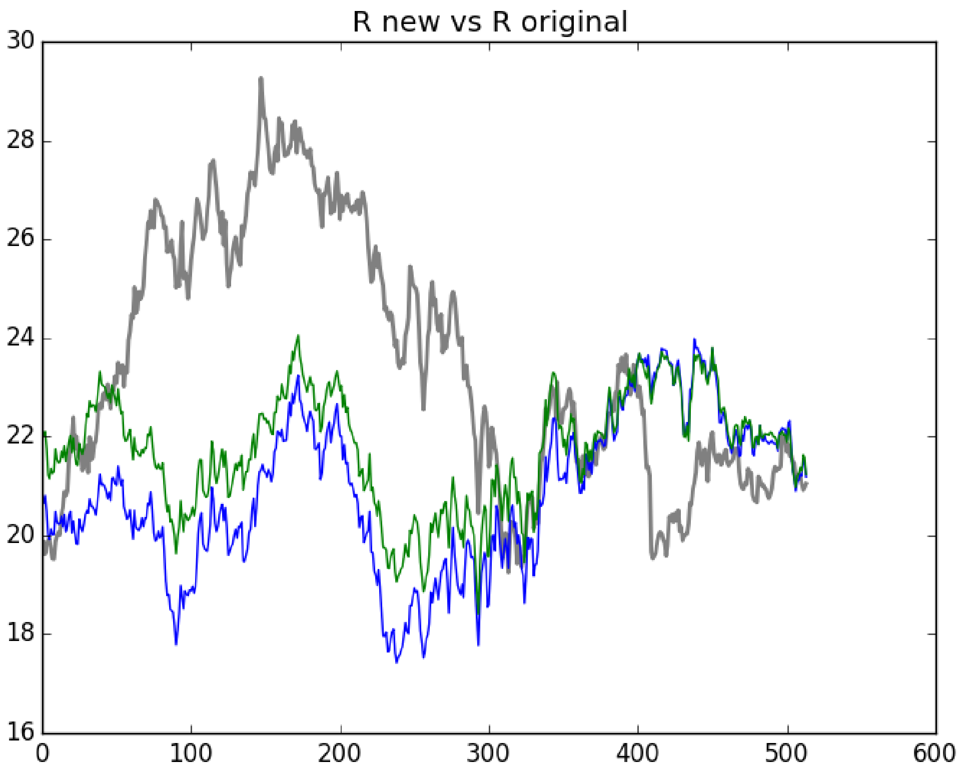


As the following step we will force both Python and R programs to show the cross results. The python will yield predictions for the final model from R: [”Cardinal”, ”Olympus”, ”St.Jude”, ”Lenovo”, ”MicronTech”, ”STMElectro”]. And the R program will return the forecast for the Python final model: [”Google”, ”Olympus”, ”St.Jude”, ”Lenovo”, ”MicronTech”, ”STMElectro”]
The R cross-result is:

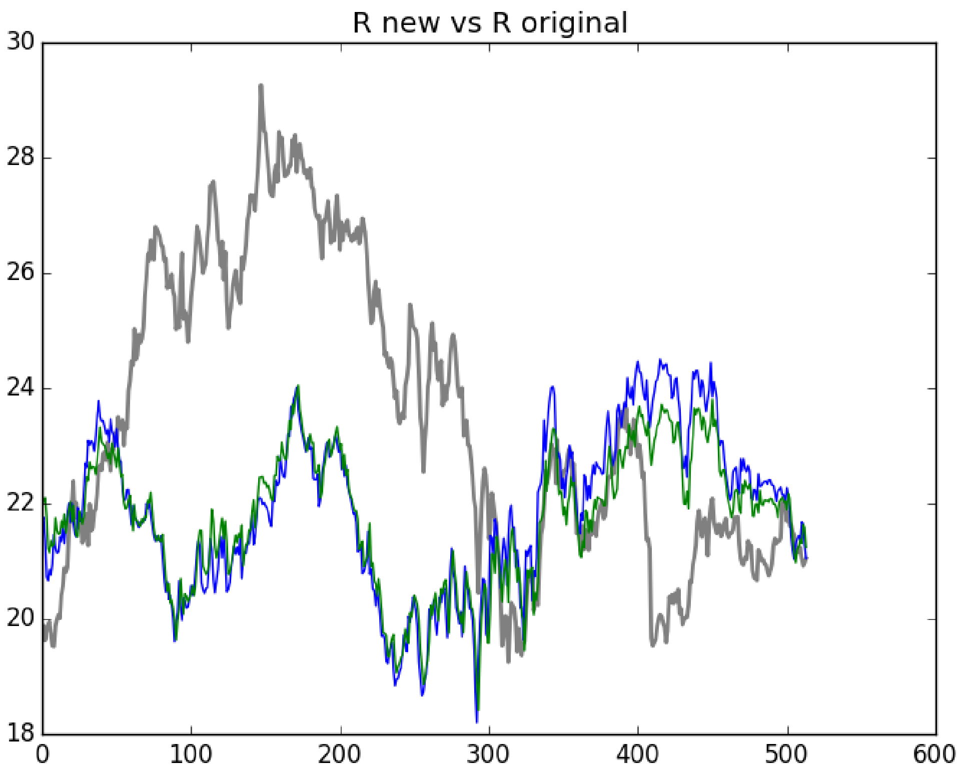
```
> p<-build_predictions()
[1] "the mean of the y is: "
[1] 23.46462
[1] "the std of the y is: "
[1] 2.55532
[1] "the mean of the predictions is: "
[1] 21.73608
[1] "the std of the predictions is: "
[1] 1.152263
[1] "the mean absolute error of the predictions is: "
[1] 2.631951
[1] "the root mean squared error of the predictions is: "
```

```
[1] 3.305496
> p$model
Generalized least squares fit by REML
  Model: Intel ~ Olympus + Google + St.Jude + MicronTech + STMElectro +      Lenovo
  Data: build_data
  Log-restricted-likelihood: -1760.532

Coefficients:
(Intercept)     Olympus      Google     St.Jude  MicronTech  STMElectro
2.299629176 0.047291974 0.005870799 0.165562665 0.105875773 0.167784354
     Lenovo
0.401918284

Degrees of freedom: 1239 total; 1232 residual
Residual standard error: 0.982245
```

On the following image you can see the difference in two models forecasts returned by the R program, where the gray line is as above the real values, the green line is the new forecasts and the blue line is the old model forecast.



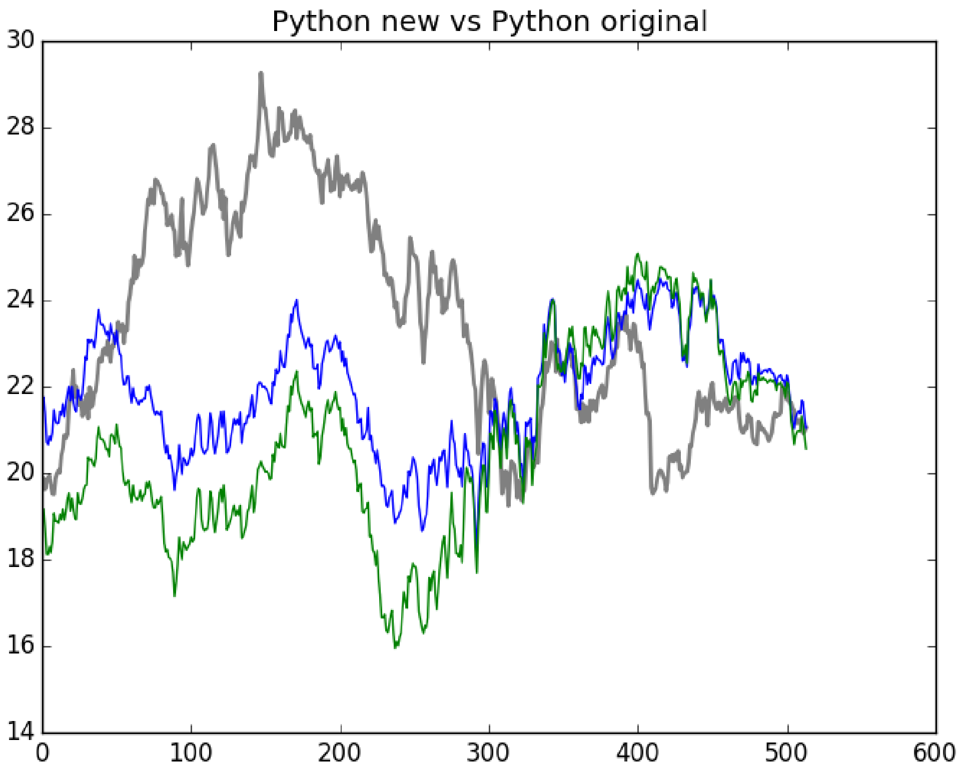
The R program considers the new model to be better in comparison to it's original suggestion, since the MAE is 0.661 smaller by the new predictions row and the RMSE of the new predictions row is 0.827 smaller. The original model was chosen based on the AIC and the predictions were not taken into account during that step.
On the next image you can see the predictions for the same model but estimated by different program. Same notation as above is used: gray - real values, green - R forecast, blue - Python forecast.



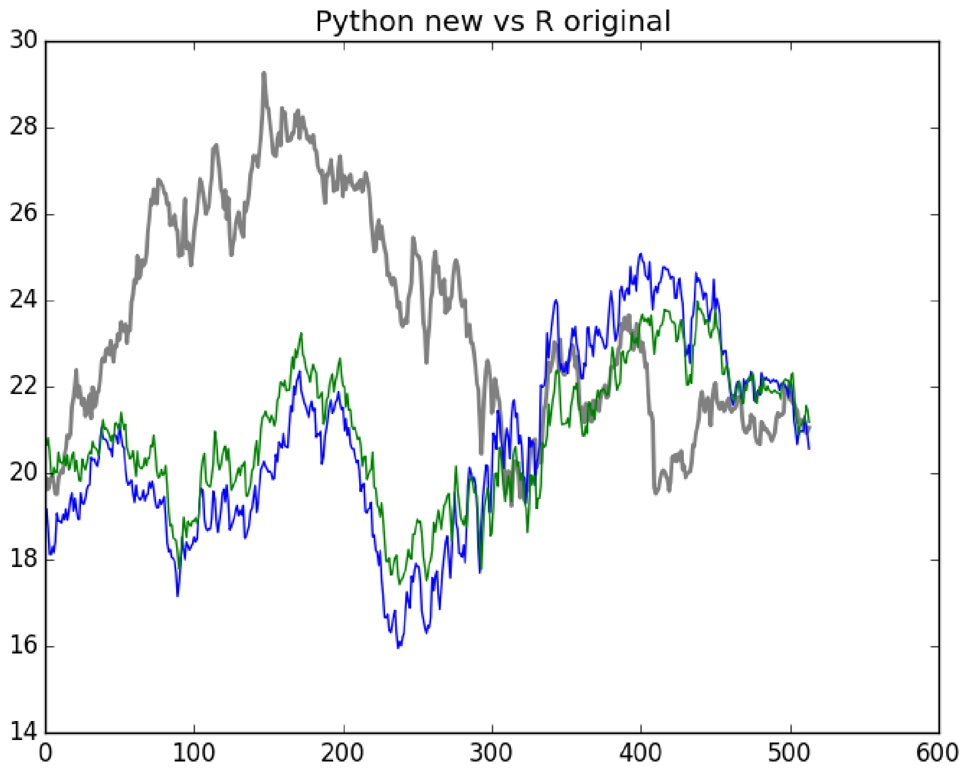
The interesting thing that predictions built by the R program for the Python original model are more accurate than the python predictions for the same model. The R MAE is 0.208 smaller and the R RMSE is 0.166 smaller. The Python cross results are:

```
>>> build_predictions()
the mean of the y is: 23.4646213592
the std of the y is: 2.55283813168
the mean of the prediction is: 20.699905831
the standard deviation is: 2.18673197501
the mean absolute error of the prediction is: 3.9100716967
the root mean squared error is: 4.81718549588

the model is: ['STMElectro', 'Olympus', 'St Jude', 'Lenovo', 'MicronTech', 'Cardinal']
                            GLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.998
Model:                            GLS   Adj. R-squared:                  0.998
Method:                 Least Squares   F-statistic:                 8.261e+04
Date:                Mon, 26 Oct 2015   Prob (F-statistic):               0.00
Time:                        14:34:25   Log-Likelihood:                -1785.4
No. Observations:                1239   AIC:                             3583.
Df Residuals:                    1233   BIC:                             3613.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
x1             0.2337      0.025      9.273      0.000         0.184     0.283
x2             0.1363      0.010     13.277      0.000         0.116     0.156
x3             0.3977      0.015     27.166      0.000         0.369     0.426
x4            -0.0567      0.006     -9.826      0.000        -0.068    -0.045
x5             0.2696      0.005     57.903      0.000         0.261     0.279
x6             0.1533      0.012     12.533      0.000         0.129     0.177
==============================================================================
Omnibus:                       32.097   Durbin-Watson:                   0.140
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               18.122
Skew:                           0.114   Prob(JB):                     0.000116
Kurtosis:                       2.453   Cond. No.                         70.4
==============================================================================
```

On the following image you can see the difference in two models forecasts returned by the Python program, where the gray line the real values, the green line is the new forecasts and the blue line is the old model forecast.



The Python program considers the new model to be worse in comparison to it's original suggestion, since the MAE is 1.072 smaller by the old predictions row and the RMSE of the old predictions row is 1.4 smaller. The original model was chosen based on the AIC and the predictions were not taken into account during that step.
On the next image you can see the predictions for the same model but estimated by different program (new model for the python program). Same notation as above is used: gray - real values, green - R forecast, blue - Python forecast.



Here predictions built by the Python program for the R original model are less accurate than the R predictions for the same model. The R MAE is 0.617 smaller and the R RMSE is 0.684 smaller.
The best prediction is built by the R prediction function for the Python original model. This result can be explained by the fact, that R gls.predict() module is higher accuracy computation process. At the same time Python managed to figure out better regression using AIC. We give Python additional Point for yielding more accurate results.
The final score for both programs is 7.66 Points for R program versus 6.71 Points for Python program. This mean that objectively R is more suitable for beginners to use as the programming language for a small statistic task with small data sets.

4.2 Subjective comparison

During the process of the program development Both languages have shown their good and bad sides. Good sides of the Python language are:

1. Easy refactoring from functions to classes.

2. Sufficient amount of tutorials and examples for every function used in the program.
3. Intuitive data structures.
4. Very clear debugging messages.

The bad sides of the language for this work are:

1. Too slow third-party functions to do the full data set test.
2. Formatting step took much time to figure out.
3. Pandas library does not work with pypy interpreter, which prohibits the program from easy speed up.

The good thing about R program development are:

1. The run time.
2. Data structures are very powerful.
3. Fast real testing possibility.
4. Low memory allocation.

The negative experience during working with R was caused by:

1. Non-intuitive syntax.
2. Hard to run R script from the console in non-interactive mode.
3. Not enough examples for function used in this work.

Subjectively speaking R is slightly better for data analysis programs, when you have already thought about a program architecture and you know approximately what functions the program will use. The Python language is better for fast prototyping on small data sets. Python is better for very beginners, when no architecture of the program is created.

4.3 Conclusion

After point to point comparison and collecting the personal opinion, we have decided, that R programming language is more suitable for pure analytical task.

1. The language is fast.
2. The language is not memory hungry.
3. The results are adequately accurate.

The R language has a big high-quality community behind it. There are certain standards for the documentation and the code quality, which ensures the quality of the libraries.
The negative moment can be the development of a bigger program. Prototyping in R can be tricky for mid- and very complex programs. You have to think in advance about class connections. Although you can also keep everything in functions form - this will allow you to prototype faster, but will cause some difficulties later on during refactoring.

5 Literature

1. https://en.wikipedia.org/wiki/Python_(programming_language)
2.
3. https://www.r-project.org/about.html
4. http://www.revolutionanalytics.com/what-r
5. https://en.wikipedia.org/wiki/R_(programming_language)
6. http://stackoverflow.com/questions/2770030/r-or-python-for-file-manipulation
7. http://datascience.stackexchange.com/questions/326/python-vs-r-for-machine-learning
8. http://www.kdnuggets.com/2015/05/r-vs-python-data-science.html
9. http://blog.datacamp.com/r-or-python-for-data-analysis/
10. http://101.datascience.community/2015/05/12/data-science-wars-r-vs-python/
11. Memory Profiler for Python https://github.com/fabianp/memory_profiler
12. Python 3 https://www.python.org/doc/
13. SciPy http://www.scipy.org/
14. Statmodels http://statsmodels.sourceforge.net/
15. Six https://pythonhosted.org/six/
16. Base R library https://stat.ethz.ch/R-manual/R-devel/library/base/html/00Index.html
17. Stats R library https://stat.ethz.ch/R-manual/R-patched/library/stats/html/00Index.html
18. Nlme R library https://cran.r-project.org/web/packages/nlme/nlme.pdf
19. Tseries R library https://cran.r-project.org/web/packages/tseries/tseries.pdf
20. Coursera: "R Programming"https://www.coursera.org/course/rprog
21. RStudio IDE https://www.rstudio.com/
22. OO systems in R http://adv-r.had.co.nz/OO-essentials.html
23. Big Data in R http://r-pbd.org/