Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Master's Thesis

Time series analysis with Python

Illia Prazdnyk

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Bc. Illia Prazdnyk

Systems Engineering and Informatics

Informatics

Thesis title

Time series analysis with Python

Objectives of thesis

The main objective is to provide an overview of various time-series analysis techniques, where they can be applied and how to implement them using Python programming language.

Based on a bit of history, mathematics and statistics, different sorts of time related data and its common properties and differences shall be explained. This will be supported by an overview of various community-made tools and suitable use cases for them. At the end, the gathered knowledge will be demonstrated on a data-driven model based on open data.

Methodology

The methodology of the thesis is based on analysis and study of the relevant technical and scientific sources focusing on fundamental statistical and mathematical models that the time series analysis and forecasting are based on.

Specific attention will be paid towards Python programming language and explaining why it is such a popular tool for the area of study, and how does it allow people to develop advanced analytical applications based on the fundamental techniques of statistics and mathematics. Based on synthesis of gained knowledge a working prototype will be implemented using standard methods of software engineering, including step-by-step implementation guide. This program would be powered by different Python packages, such as NumPy, Pandas, Scikit-learn, TensorFlow, and Keras.

The proposed extent of the thesis

60-80 pages

Keywords

Time series, Python, Statistics, Machine learning

Recommended information sources

HAMILTON, James D., 1994. Time Series Analysis. Princeton, NJ. Princeton University Press: Levant Books. ISBN 978-0691042893.

NIELSEN, Aileen, 2019. Practical time series analysis: prediction with statistics and machine learning. Beijing: O'Reilly. ISBN 978-1-4920-4165-8.

VISHWAS, B V a Ashish PATEL, 2020. Hands-on Time Series Analysis with Python: From Basics to Bleeding Edge Techniques. Apress Media. ISBN 978-1484259917.

Expected date of thesis defence

2021/22 SS – FEM

The Diploma Thesis Supervisor

Ing. Petr Hanzlík, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 1. 3. 2022

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 7. 3. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 17. 03. 2022

Declaration

I declare that I have worked on my master's thesis titled "Time series analysis with Python" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 30.03.2022 _____

Acknowledgement

I would like to thank Ing. Petr Hanzlík, Ph.D. for all the advice and support he provided while I was writing the thesis. He helped me a lot with reducing the stress and keeping the deadlines on track.

Time series analysis with Python

Abstract

This thesis is focused on analysis of time ordered data using Python. The theoretical part uncovers a bit about how people achieved the understanding of time that we have nowadays, starting from the ancient invention of sundials and ending on the conventions for standardizing international time zones. After this part, there is a big block dedicated to explaining the statistical and mathematical foundations on which the time series analysis is based, including the description of basic principles of two statistical models: ARIMA and VAR. After that there is an introduction to Python programming language and deep learning together with a presentation of LSTM model and how it works.

The practical part presents possible use case for each time series forecasting model presented above using Python and its packages commonly used for data analysis and visualisation. These demonstrations work primarily with close price of publicly traded stocks. The resulting code is executed on a cloud server provided by Google, and the notebook with all the code is linked in the appendix.

Keywords: Time series, Python, Statistics, Machine learning

Analýza časových řad pomocí jazyka Python

Abstrakt

Tato diplomová práce je zaměřena na analýzu časově uspořádaných dat pomocí Pythonu. Teoretická část je věnovaná tomu, jak lidé dosáhli vnímaní času které máme dnes - od starověkého vynálezu slunečních hodin až po dohody o standardizaci mezinárodních časových pásem. Po této části následuje vysvětlení statistických a matematických základů, na kterých je analýza časových řad založena, včetně popisu základních principů dvou statistických modelů: ARIMA a VAR. Poté následuje úvod do programovacího jazyka Python a hlubokého učení, spolu s prezentací modelu LSTM a jeho fungování.

Praktická část ukazuje možný případy užití každého z výše zmíněných modelů pro predikci časových řad pomocí Pythonu a jeho balíčků, které jsou běžně používané pro analýzu a vizualizaci dat. Tyto ukázky pracují primárně s uzavírací cenou veřejně obchodovaných akcií. Výsledný kód se spouští na cloudovém serveru poskytovaném společností Google a notebook se všemi kódy je k dispozici pomocí odkazu v příloze.

Klíčová slova: Časové řady, Python, Statistika, Strojové učení

Table of content

List of pictures

List of abbreviations

ACF – Auto Correlation Function

ARIMA – Auto Regressive Integrated Moving Average

GPU – Graphical Processing Unit

IDE – Integrated Development Environment

LSTM – Long short-term memory network

OLS – Ordinary Least Squares

PACF – Partial Auto Correlation Function

PIP – Pip Installs Packages

RSS – Residual Sum of Squares

VAR – Vector Auto Regression

1

2 Introduction

We, humans, always had a desire to know what is going to happen. The natural way to predict that is to use the experience and learn from the past successes (or failures). Of course, it would be important for us to maximize the quality of that prediction. Throughout thousands of years various mathematical models have been developed in order to support the forecasting and decision-making. On top of that is the recent rise of electronic and digital technologies that has shown completely new horizons to measurement and analysis of the world around us. We can create autonomous devices to measure most of the physical, social or economic processes and use advanced information systems to analyse the aggregated data as a whole and create simulations of various possible future outcomes without missing a single detail from the past.

This thesis tells a bit about the history of how the time was measured, highlights the fundamental mathematical and statistical models, and shows how these models can be leveraged using modern tools.

The backbone for the coding part is Python programming language. It has a large set of packages available for implementation of time series analysis programs, which allows the developers to create software of a much higher functional complexity.

3 Objectives and Methodology

3.1 Objectives

The main objective is to provide an overview of various time-series analysis techniques, where they can be applied and how to implement them using Python programming language.

Based on a bit of history, mathematics and statistics, different sorts of time related data and its common properties and differences shall be explained. This will be supported by an overview of various community-made tools and suitable use cases for them. At the end, the gathered knowledge will be demonstrated on a data-driven model based on open data.

3.2 Methodology

The methodology of the thesis is based on analysis and study of the relevant technical and scientific sources focusing on fundamental statistical and mathematical models that the time series analysis and forecasting are based on.

Specific attention will be paid towards Python programming language and explaining why it is such a popular tool for the area of study, and how does it allow people to develop advanced analytical applications based on the fundamental techniques of statistics and mathematics. Based on synthesis of gained knowledge a working prototype will be implemented using standard methods of software engineering, including step-by-step implementation guide. This program would be powered by different Python packages, such as NumPy, Pandas, Scikit-learn, TensorFlow, and Keras.

4 Literature Review

4.1 History of time measures

It is most probable that living creatures had some concept of time on an instinct level long before humans started to wander earth. The prehistoric animals should have observed the sun go up and down, but their intelligence was a big limitation to how these observations can be used. This has changed millions of years later with the rise of humans, who want to find the deeper relationship between the time and the universe around us.

4.1.1 Ancient inventions

"Imagine a spinning bicycle wheel. If you observe the wheel alone and choose any particular point, for example the bottommost point, you will notice that it will return to the bottom as it spins as though nothing has changed. When you now, however, observe the wheel as part of the moving bicycle, you will realize that your bottommost point has not in fact returned to the same place as it spins, but rather it has moved forward along the ground. This exercise may seem very elementary, but when considering the way humans measure the passage of time, it becomes quite puzzling. Take any calendar date or lunar phase and it will inevitably repeat, so far as human timekeeping is concerned, but the originally chosen moment will never return as we say it has. Our fundamental timekeeping system is based on cycles—cycles of seconds in minutes, minutes in hours, hours in days and days in years—but we know time to be linear. The answer to why we have chosen to measure time cyclically lies in the origins of timekeeping itself. " (Going Back in Time: The History of Timekeeping., 2013)

The origins of systematic analysis of time date back to thousands of years ago and are based on findings of some sticks and bones with markings on them, which supposedly symbolised the passage of days. The next achievement of humanity was the calendar. There were many variations of the ancient calendars, one of which was invented by ancient Egyptians around 4500 BC. They are thought to be the first to divide the year into 12 month and 365 days

based on seasons and time. The beginning of the new year was indicated by appearance of the Sirius star on the night sky (it is not visible for the period when it is too close to the sun). To find out what time of a day it was the first thing that comes to mind is to look at where the sun is on the sky, this led to an invention of a sundial.
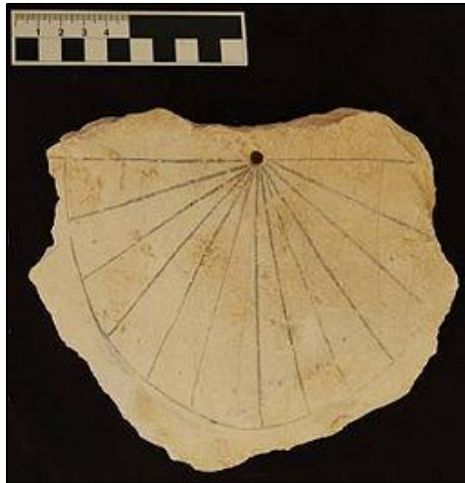
Figure 1 - Ancient Egyptian Sundial housed at University of Basel
(Reconstruction of anctient Egyptian sundials, 2014)

It should be obvious why sundial was not sufficient – the distance (and angle) between the planet earth and the sun changes throughout the year, which makes the amount of daylight and sun's position on the sky different throughout the year.

This led to a desire to make a device that would allow us to measure the flow of time without being dependent on the light of a star. The device is a water clock.

A water clock is a device that could be of two types: inflow and outflow. The difference is quite simple – the first one is a vessel that has markings on the inner side and a hole in it. The vessel gets filled with water and the hole gets opened to allow the water flow out and reveal the markings. The second type, outflow has the same idea, but contains two vessels with water going from one to the other, which has the marking on the inside walls that get hidden by the water and hide the markings as the time passes.

But both the hourglass and the water clock had two major issues – the accuracy and the length of time they could measure.

Figure 2 – Ancient water clock in Oriental Institute Museum, University of Chicago (Daderot, 2014)

"For most of history, ordinary people did not have regular and easy access to any kind of time measuring device whatsoever, other than to glance at the sky on a sunny day and see where the sun was. For them, time as we understand it today did not really exist. The one group in medieval times whose day was ruled by time in a way not unlike people today were the Benedictine monks, with their ecclesiastically regulated prayer times, the eight Canonical Hours: lauds (just before daybreak), prime (just after daybreak), terce (third hour), sext (sixth hour), nones (ninth hour), vespers (eleventh hour), compline (after sunset), and matins (during the night). The signal that announced each canonical hour and regulated the monks' day was a ringing bell. This gives us our word "clock," which comes from the medieval Latin word for bell, clocca." (About time, 1999)

4.1.2 Modern inventions

So, to increase the accuracy of measurement, people had to stop relying on the sun or flow of matter and find some new physical phenomenon that has some fixed duration features. The phenomenon is oscillation. Quite a scary word that means movement back and forth in a regular rhythm.

"Early oscillating mechanisms were called escapements. The first escapement, the verge-and-foliot, comprised a freely swinging horizontal bar (the foliot) attached to a centrally located vertical shaft (the verge). The mechanism was driven by gravity. A heavy weight hung from a cord wrapped round a horizontal spindle. As the weight slowly descended, the cord turned the spindle. A toothed crown-wheel on the spindle made the escapement oscillate, the escapement regulated the rate at which the spindle turned, and the rotation of the spindle measured the passage of time by moving a hand around a marked clock face. The rate of oscillation, and hence the speed of the clock, was adjusted by moving symmetrically placed small weights along the foliot bar.

In the fifteenth century, clockmakers started to use tightly coiled blades of metal -- springs -- to power their timepieces, instead of gravity. Following Galileo's famous 1583 observation that the period of oscillation of a swinging pendulum seemed to depend only on the dimensions of the pendulum, not on the size of the arc, the verge-and-foliot escapement was modified -- and improved -- so that the swing of a pendulum arm regulated the motion. The pendulum clock was itself improved when the verge-and-foliot mechanism for controlling the rate of rotation of the crown wheel was replaced by the anchor escapement, where a calliper-like "anchor" performed the task previously carried out by the verge-and-foliot." (About time, 1999)Even though it was a major improvement compared to the ancient devices, people still couldn't fully rely on these clocks and had to continuously readjust them according to the good old sun.

The first appearance of a somewhat accurate clocks is closely connected to the humanity's, or sailor's to be exact, need to determine their location. This problem was not as big for first naval traders, because they simply didn't go away too far from the shore. But after the discovery of great oceans and expansion of European civilization to the Americas the sailors became more and more interested in measuring the length of their trip.

"From the sixteenth century onwards, the need for an accurate clock to determine longitude became so important to growing world trade, that a number of monetary rewards were offered for the first person to produce such a device. In 1714, England's Queen Anne offered 20,000 (several million pounds in today's currency) for the first person to find a way to

determine longitude to within half a degree. Many attempts were made to solve the problem and win the various prizes. In 1759, a Yorkshireman called John Harrison tested a 5.2 inch diameter clock on a trip from Britain to Jamaica and back. The clock lost only five seconds on the outward journey, corresponding to a longitude error of only one and a quarter nautical mile. Harrison won Queen Anne's prize, and the world finally had a way to determine longitude: by the accurate measurement of time."

So, the problem of measuring time accurately was solved, but there were still some challenges to overcome. The biggest one was that even though people in different cities could measure time precisely, they were totally disconnected from each other which resulted in issues with cross-regional transportation and delivery. Many cities were measuring the time from some different moment, and people had to compensate for that and readjust their measurement while travelling from one city to another. But due to not having a better alternative – all the people who for some reason had to do scheduled movement from one town to another had to just deal with it.

"The problem became much worse with the arrival of the railway network in the nineteenth century. The greater speeds, together with the need to change from one line to another -- possibly from one railway company to another -- in the course of a single journey made the plethora of different local times a confusing annoyance. In England, the railroads decided that they would run their operations according to London time, as determined by the Royal Observatory at Greenwich, and by 1848 practically all British railroad companies operated according to what would eventually become known as Greenwich Mean Time (GMT). For a while, many local towns continued to keep their own time, determined by local observations of the sun, but gradually the benefits of having a single time began to outweigh tradition and local pride. By 1855, almost all public clocks throughout Great Britain showed GMT." (About time, 1999)

So, the British people were pioneers for the synchronization of clocks in their country, but the humanity was still in the need to form some unified system. The next big challenge was faced by the United States. A territory about 25 times larger than the

UK at that time. There, a much bigger travel between the cities has led to an existence of approximately 80 timetables for different parts of the country. It was in the year 1869 when Charles Dowd proposed to split the country into four time zones. He suggested for every time zone to be 15 degrees of latitude, which makes the time difference between the middle of each zone to be exactly one hour.

That is because we have already decided that an hour is one 24th of time it takes the earth to make a turn. The whole globe has 360 degrees of latitude, and if we divide 360 by 24 – you guessed it – we get 15 degrees of latitude.

The system proposed by Charles Dowd was very close to the one in which we live nowadays. All that needed to be done was to explain the benefits of synchronization to the humanity and decide on some "zero point" time zone from which the count should start.
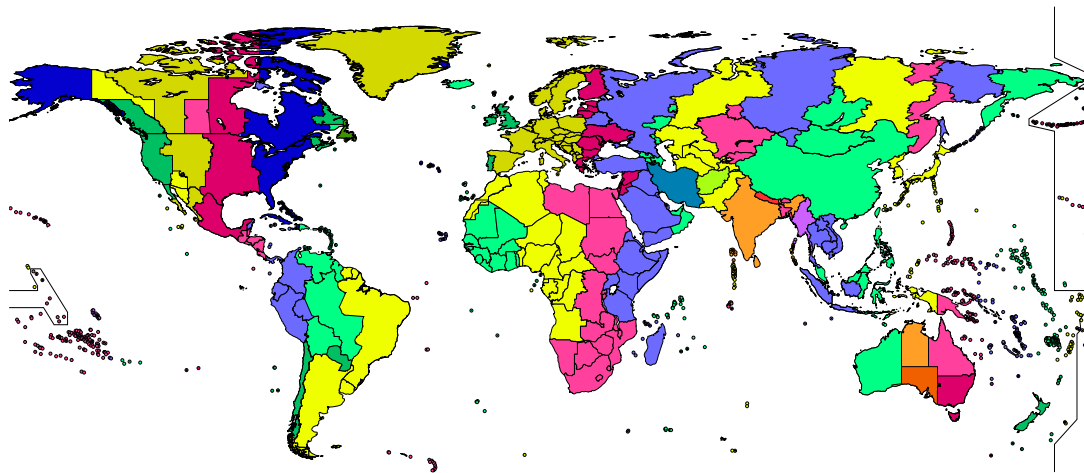
Figure 3- UTC time zones (Heitordp, 23)

Yet, it was a very gradual process which took a lot of effort, cooperation, and most importantly listening to what other people had to say. The baseline was set to Greenwich, first by Sweden, US and Canada in 1883. Later, in 1884 there was an International Meridian Conference held in Washington, D.C. to come to an agreement upon using Greenwich as the baseline for the whole planet. The global adoption of which took almost 100 years with the last country to make the change being Liberia in 1972. (About time, 1999)

It's fascinating how our time measurement system has evolved throughout the years, and how it was striving to finally reach the elegant mathematical form in which it currently is.

4.2 Scientific foundation

Even though it might seem like it from the first chapter of this thesis – scheduling of transportation was not the only use of time measurement systems. Another tremendous thing it allowed us to start doing is logging of some observations on a time plane.
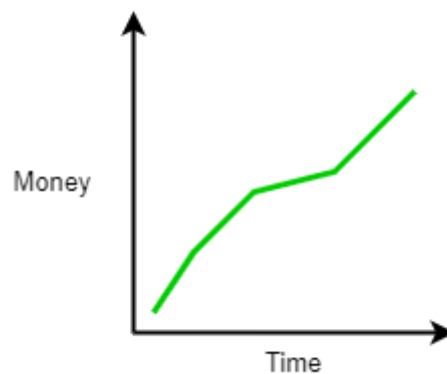
Figure 4 - Example of a time based plot

This is something that a specialist (or a beginner) of any area would find relevant, from stock exchange trader to the farm's worker, as it is a basic instrument of displaying and logging any continuous information.

The following part will give some insight about the basic techniques of how such data is processed and analysed.

4.2.1 Types of time series data

Any sorts of data can be recorded through the passage of time as long as it is available, which means that time series data can be of any type that data in general could be.
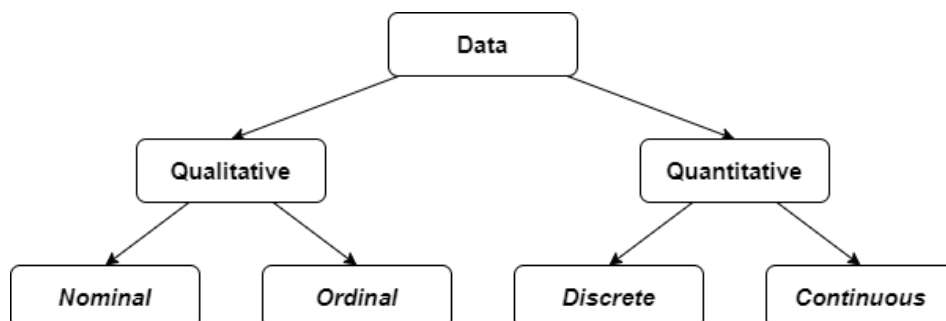
Figure 5- Types of data

There are two main types of data that are also split into four subtypes. The first one is **qualitative** data, which as the name suggests describes some quality of the subject we try to model. This may be the genre column in a list of movies or a grade mark in some teacher's journal. The difference between those two examples is also a hint to which of the data subtypes they relate – the movie genre would of **nominal** data subtype because there is no quantitative information in words like "Comedy" or "Horror", and on the other hand the student's grade would be of an **ordinal** type, because it has quantitative information about how good is the order of such results. Keep in mind that the students grade example would be of qualitative type as long as it is some category of possible grades like "Excellent", "Very Good" etc., but if it would be the amount of the answers he got correct on the test – it would be of the second type that is explained below.

Quantitative data are the numerical results of some measurable activity, phenomenon or matter that must be represented by a number. For example, a number of students in a class or a height of a flower. Just like before, those two examples represent both subtypes of quantitative data – **discrete** and **continuous**. Discrete data is the data that can only be counted as full integers, like how many students there are in a class. While the continuous is the information that can be split and looked at a finer scale. Like the growth of a flower can be tracked down to a particle of a meter, while there can't 23.5 students in a class.

4.2.2 Characteristics of time series data

In order to be able to work with time series some of its fundamental characteristics must be understood. Those are the so-called behaviors that any time series might exercise, and it is crucial to know what they are:

- Seasonality – the data follows some pattern based on some specific period, like Christmas toys sales which rise and then drop once per year.
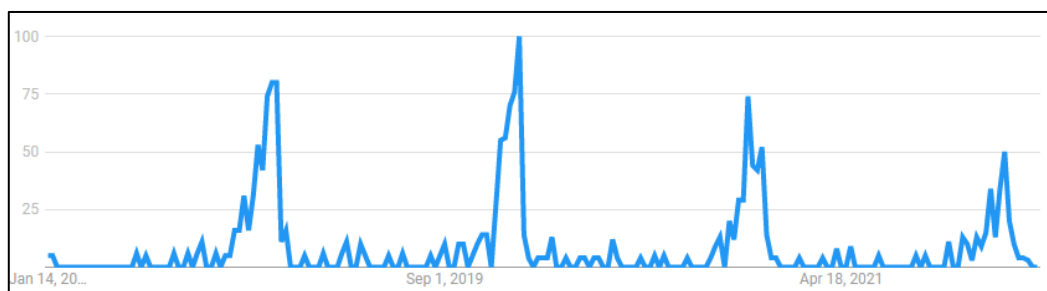
Figure 6 - Seasonality example, Google search trend for Christmas tree (2022)

- Trend – the direction which the data is trying to reach, like a growth or decline that is present throughout the observations.
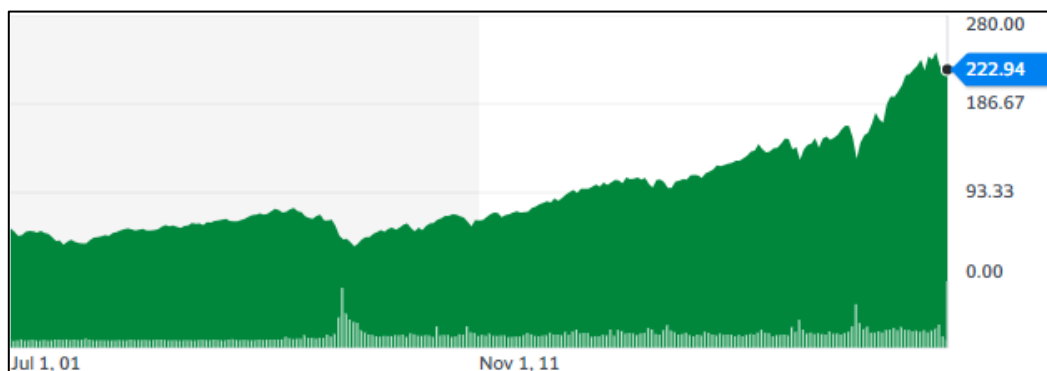
Figure 7 - Trend example, ETF index price (2022)

- Cyclicity – the repetition of data going up and down. The difference from seasonality is that the periods between cycles may not be fixed and are caused by some economic (or different) events.
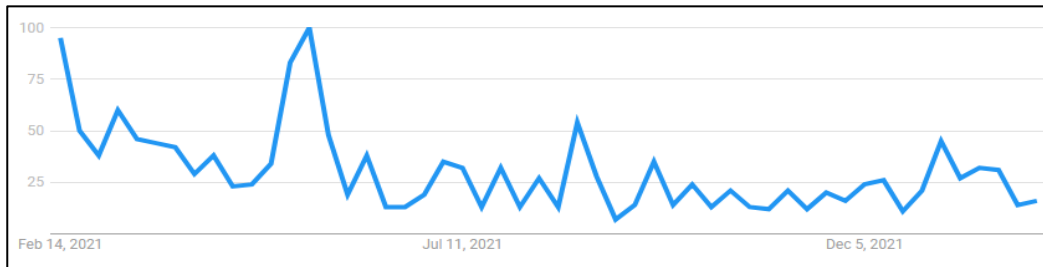
Figure 8- Cyclicity example, Google search trends for umbrella (2022)

- Randomness – the final characteristic of time series data, which is supposed to represent the effect of unknown variables.
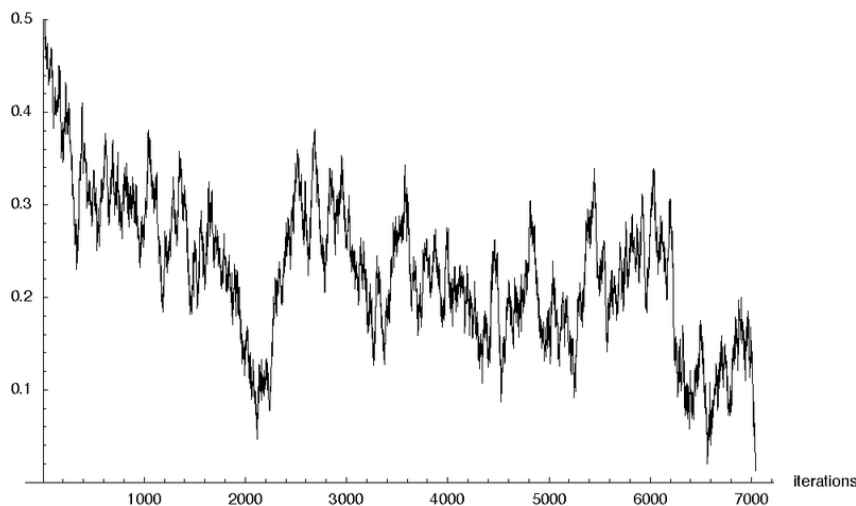
Figure 9 - Randomness example (Elser, 2007)

Please keep in mind that those are not categories of time series, but rather different components of its behavior that are present in any data to some extent.

4.2.3   Some basic statistical definitions

Time series analysis is a part of statistics, and in order to perform it, some basic statistical definitions must be explained.

Distribution – one of the fundamental terms in statistics. It demonstrates how different are the observations in the dataset by putting them on an ordered plane.
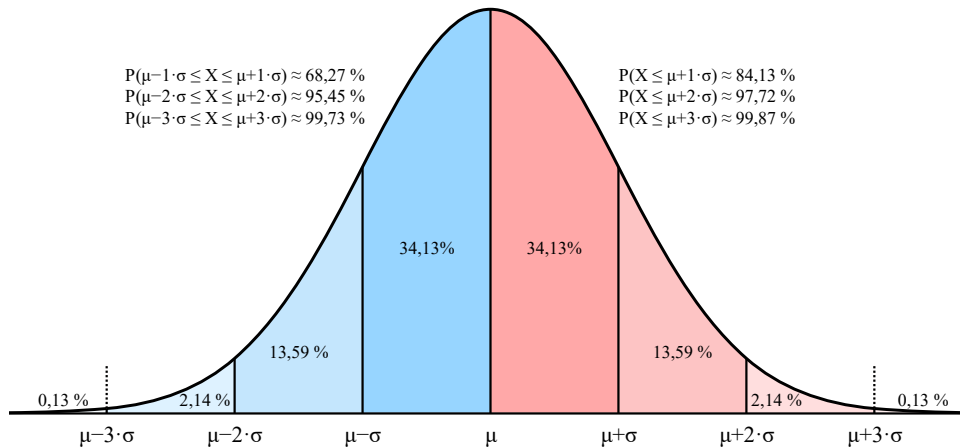
Figure 10- Distribution plot (Kowarschick, 2012)

Mean – the easiest statistical term that represents an average value of all observations. If we have a dataset where we track height of students in a group with such results: 183 cm, 164 cm, 160 cm, 175 cm, 190 cm, 158 cm. The mean is equal to sum of all the results divided by number of observations, so for our example it is:

(183 + 164 + 160 + 175 + 190 + 158) / 6 = 171.66 cm

$$m = \frac{\text{sum of the terms}}{\text{number of terms}}$$

Figure 11 - Mean formula

Median – the center observation of sorted dataset. If we have 5 (odd) total observations in our dataset – after we sort it the third value would be the median. If the total number of observations would be 6 (even) – after we sort our data, the median would be the mean (average) between the third and the fourth values. So, for our height example:

158 160 **164 175** 183 190; (164 + 175) / 2 = 169.5 cm

Mode – the most popular value in a dataset. If our dataset has 5 ladybugs, in which we document the three of them are red and two are yellow, then red is the mode of colour

25

variable for our dataset. In our height example all observations are unique, so there is no mode.

Variance – the degree of how far a variable is spread from its mean value. To calculate it, we subtract the mean from every observation, sum the squares of resulting numbers and divide it by (number of observations – 1). For our height example the variance is equal to 171.46

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{1}$$

$$S = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

Figure 13- Standard deviation formula

Standard Deviation – actual distance of how far a variable is spread from the mean. It is calculated as the square root of variance.

Correlation – the measure of how one variables movement can explain the movement of a different variable. Like the relationship of the distance from earth to sun and the air temperature.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Figure 14- Correlation coefficient formula

The terms that were presented in this chapter are just a little grain of all the different definitions that one may stumble upon in the world of statistics but understanding them is crucial to comprehend even the most basic texts about statistical analysis.

4.2.4 Regression

One of the primary goals of time series analysis is to be able to predict the future based on the values of the variables that will be known to us. Regression is one of the most used techniques for that matter.

"Linear regression is a mathematical technique that attempts to describe the relationship between two or more variables with a linear or straight-line function. Based on an analysis of the available data or sample, the technique also can be used to draw inferences about a larger population or data set, or to make predictions about future data. Simple linear regression is a subtype of linear regression in which there is a single outcome or dependent variable and a single predictor or independent variable." (Advanced Statistics: Linear Regression, Part I:, 2004)

First, the simple linear regression:
"Simple linear regression uses the equation for a line to model the relationship between two variables. If z is the outcome variable and x is the predictor variable, then: **z = kx + c** where k is a coefficient that represents the slope of the linear relationship between the variables x and z, and c is a constant. The constant c is termed the ''z intercept'' because this is the value of z where x = 0 and the regression line crosses the z axis." (Advanced Statistics: Linear Regression, Part I:, 2004)

The same equation can be extended to include more variables:
"Multiple linear regression extends simple linear regression to include more than one explanatory variable. In both cases, we still use the term 'linear' because we assume that the response variable is directly related to a linear combination of the explanatory variables." (Multiple Linear Regression (2nd Edition), 2020)

One interpretation of a linear regression formula was already presented, it can be written in another form as below:

$$\mathbf{y_t = \alpha + x_t^{'}\beta + u_t}$$

y stands for target variable, **alpha** is the value of **y** while **x** is equal to zero, meanwhile **x** is the explanatory variable and **beta** is the coefficient that **x** should be multiplied by in order to get **y**, the final component – **u** is the residual, or in other words, error which cannot be estimated but can be seen upon testing. Those coefficients are estimated by using the Ordinary Least Squares method.

The OLS method was invented around 200 years ago by Adrien-Marie Legendre and Carl Friedrich Gauss. It works by estimating the optimal alpha and beta coefficients for the model. The optimal coefficients would be the ones that produce the smallest difference between the prediction and the observation it is tested on, for all observation in the dataset. The word "Squares" means the squared distance between the prediction and actual value of our target value. We have to square it because our model may miss the target both by a positive and a negative distance, so in order to sum those positive and negative errors we have to square them first.

Below is a formulated OLS model for beta coefficient and residual sum of squares formula from the book "Time series analysis" by James Hamilton.

$$RSS = \sum_{f=1}^{T}(y_t - x_t'\beta)^2$$
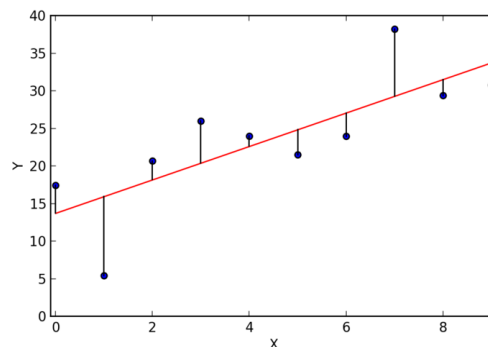
Figure 15 - Residuals visualised (Halswanter, 2013)

$$b = \left[\sum_{t=1}^{T}(x_t x_t')\right]^{-1}\left[\sum_{t=1}^{T}(x_t y_t)\right]$$

28

The theory presented above can be very easily applied to the data that has a linear relationship between target and explanatory variables, but in case their relationship would not be linear, but more like some sorts of a curve – we would need to use non-linear regression.
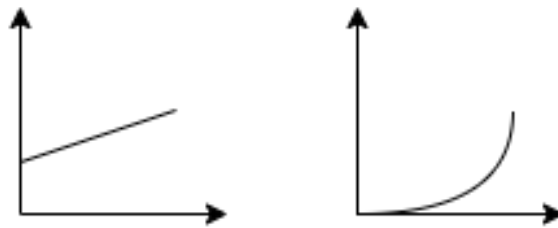
Figure 16 - Linear vs nonlinear relationship

"In nonlinear regression analysis the dependent variables are modelled as a nonlinear functional model with unknown coefficients and one or more free variables. In literature, there are a large number of nonlinear regression models. There are some important nonlinear growth models which are very useful to know the growth behavior in a particular period

namely Maltus model, Monomolecular model, Logistic model, Gompertz model and Richards model." (Estimation Methods of nonlinear regression models, 2019)

A nonlinear regression model can be put as

$$Z_j = g(Y_{1j}, Y_{2j}, \ldots Y_{lj}; \alpha_1, \alpha_2 \ldots \alpha_r) + \varepsilon_j; \; j = 1, 2, 3 \ldots m.$$

where Z is dependent variable,

$Y_1, Y_2, \ldots \ldots Y_l$ are independent variables,

$\alpha_1, \alpha_2 \ldots \alpha_r$ are parameters,

ε is error random variable,

m= number of rows,

g (.)= functional which is not linear.

Figure 17 - Nonlinear regression model (Sorce: Estimation of nonlinear regression models)

As mentioned before, regression is one of the fundamental techniques in data analysis and will receive more spotlight later in this thesis. The models above would need a separate book to explain them in detail they really deserve, but hopefully this information extract is sufficient to grasp the information in the following chapters.

4.2.5 Differencing

The trend and seasonality characteristics of time series make the values behavior over time unstable. Our goal is to learn how to measure those changes and remove (or reduce) their effect on our data. This is also called making the data stationary. Stationary data is the one that doesn't change its properties based on when it is looked at. For example, flipping a coin. The chances of getting heads or tails are the same, no matter when it is thrown.

The technique that is used to achieve it is called differencing. There are two kinds of it: usual and seasonal. The first one removes only the trend characteristic, while the second one tries to address both seasonality and trend together. But how is it actually done?

The technique is easy and is very well explained by its name, let's start with the usual differencing:

$$Y'_t = Y_t - Y_{t-1}$$

Figure 18 - First-order differencing formula (Vishwas, et al., 2020)

First order differencing means finding the difference between observations and making it a separate column in our dataset. It can not only be of one order, but of any nth order, which would mean that we perform the same operation n number of times on resulting columns. If the order was 3 – we would do the first order difference and put the change between the values into a separate column, after that we would do the second order differencing and calculate the difference between the values in the column that we got as the result of the first order.

With the seasonal component present in the data, the idea is nothing more complicated - the only thing is that we need to know about the seasonality in our data.

Like in our Christmas toys example – it would be weird to compare their sales between January and December, but what we could do is to find the difference between this year's December sales and the ones from the previous year December (and creating a separate column with such year to year change).

4.2.6   Autoregression Model

It is very common to have a limited number of variables in data analysis. Sometimes we need to make some assumptions by observations of just one value over time. This is what autoregressive models are used for. It works by setting a lagged value of our target variable (shifting all data one period forward) as the explanatory variable.

"An autoregression model (AR) predicts future behavior based on its past data. It is when data is correlated with a consecutive sequence of a time series and the values before and after the sequence. The autoregressive model uses only past behavior data to forecast the value. AR models use past values to forecast as shown here:

$$\hat{Y}t = \mu + \phi1Yt\text{-}1 + \phi2Yt\text{-}2 \ldots + \phi pYt\text{-}p + \varepsilon t$$

where εt is white noise. This model is known as the AR(p) model, where p is the order for the autoregressive model. The AR model is easy to use to handle a wide range of time-series models.

AR is part of a time series Yt, which contains a value that depends on some linear grouping of the previous value, which defined maximum lags (signified p). It also contains an arbitrary error term εt, given as follows:

First-order AR:

$$\hat{Y}t = α + b1Yt\text{-}1$$

Second-order AR:

$$\hat{Y}t = α + b1Yt\text{-}1 + b2Yt\text{-}2$$

Third-order AR:

$$\hat{Y}t = α + b1Yt\text{-}1 + b2Yt\text{-}2 + b3Yt\text{-}3 \ yt = φ1yt\text{-}1 + φ2yt\text{-}2 + …… + φpyt\text{-}p + εt$$

where the parameters **φt** are constants.

" (Vishwas, et al., 2020)

4.2.7   Moving Average Model

The autoregression model uses previous values of our target variable in order to create a prediction. It is also possible to use the errors of the previous forecast to create a regression model. That is what moving average model is.

"A moving average (MA) is a method to get all the trends in a time series. It is the average of any subcategory of numbers. It is utilized for long-term forecasting trends. Basically, a moving average forecasts future points by using an average of several past data points. The moving average model practices past forecast errors:

$$\hat{Y}t = μ0+ εt – ω1 \ εt\text{-}2 \ \text{-} \ ω1 \ εt\text{-}2 … ωq \ εt\text{-}q$$

where εt is white noise. This model is known as the MA(q) model, where q is ordered for the moving average model. The MA model is easy to use to handle a wide range of time-series models.

Here are some definitions to understand:

ACF: The correlation of a variable with its lagged values.

PACF: The correlation of a variable with its lagged values, but after removing the effects of in-between time lags.

Correlograms: The plots of ACF and PACF against the lag length. This can give you an idea about the relation of autocorrelation between variables" (Vishwas, et al., 2020)
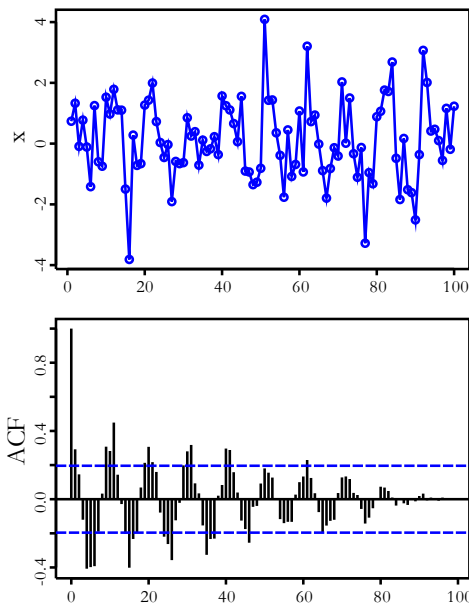
Figure 19 - Correlogram example (Manning, 2009)

"The MA part of a time-series Yt, which is an observed value in terms of a random error and some linear grouping of previous arbitrary error terms, up to a described maximum lag (signified q).

First order MA(1):

Ŷt = γ+ d0ut + d1ut-1

Second order MA(2):

Ŷt = γ+ d0ut + d1ut-1 + d2ut-2

Third order MA(2):

Ŷt = γ+ d0ut + d1ut-1 + d2ut-2+ d3ut-3 yt = Zt + θ1Zt-1 + θ2Zt-2 + …… + θqZt-q

where the parameters θt are constants.

let's combine AR and MA

ARMA (1,1): Ŷt = μ + ϕ1Yt-1+ d0ut+ d1ut-1

" (Vishwas, et al., 2020)

33

4.2.8 ARMA to ARIMA

The chapters above have already covered all the components that power the ARIMA model - autoregression, differencing and moving average.

"Autoregressive integrated moving average—also called ARIMA(p,d,q)—is a forecasting equation that can make time series stationary with the help of differencing and log techniques when required. A time series that should be differentiated to be stationary is an integrated (d) (I) series. Lags of the stationary series are classified as autoregressive (p), which is designated in (AR) terms. Lags of the forecast errors are classified as moving averages (q), which are identified in (MA) terms.
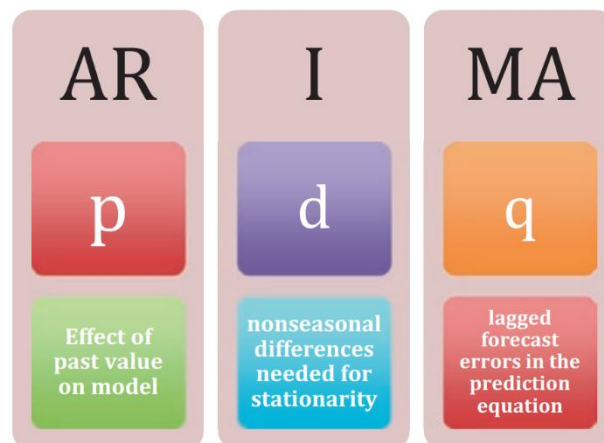
Figure 20 - ARIMA breakdown (Vishwas, et al., 2020)

A nonseasonal ARIMA model is called an ARIMA(p,d,q) model, where:
• **p** is the number of autoregressive terms.
• **d** is the number of nonseasonal differences needed for stationarity.
• **q** is the number of lagged forecast errors in the

prediction equation.

	p	d	q	Differencing	Method
ARIMA (0, 0, 0)	0	0	0	$y_t = Y_t$	White noise
ARIMA (0, 1, 0)	0	1	0	$y_t = Y_t - Y_{t-1}$	Random walk
ARIMA (0, 2, 0)	0	2	0	$y_t = Yt - 2Yt-1 + Yt-2$	Constant
ARIMA (1, 0, 0)	1	0	0	$\hat{Y}t = \mu + \phi_1 Y_{t-1} + \varepsilon$	AR(1): First-order regression model
ARIMA (2, 0, 0)	2	0	0	$\hat{Y}t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \varepsilon$	AR(2): Second-order regression model
ARIMA (1, 1, 0)	1	1	0	$\hat{Y}t = \mu + Yt-1 + \phi1 (Yt-1 - Yt-2)$	Differenced first-order autoregressive model
ARIMA (0, 1, 1)	0	1	1	$\hat{Y}t = Yt-1 - \phi1et-1$	Simple exponential smoothing
ARIMA (0, 0, 1)	0	0	1	$\hat{Y}t = \mu_0 + \varepsilon t - \omega_1 \varepsilon_{t-1}$	MA(1): First-order regression model
ARIMA (0, 0, 2)	0	0	2	$\hat{Y}t = \mu_0 + \varepsilon t - \omega_1 \varepsilon_{t-1} - \omega_2 \varepsilon_{t-2}$	MA(1): Second-order regression model

Figure 21 - p, d, q representation and methods, part 1 (Vishwas, et al., 2020)

ARIMA is a method among several used for forecasting univariate variables, which uses information obtained from the variable itself to predict its trend. The variables are regressed on its own past values. AR(p) is where p equals the order of autocorrelation (designates weighted moving average over past observations) z I (d), where d is the order of integration (differencing), which indicates linear trend or polynomial trend z. MA(q) is where q equals the order of moving averages (designates weighted moving average over past errors). ARIMA is made up of two models: AR and MA.

ARIMA (1, 0, 1)	1 0 1	$\hat{Y}t = \phi_0 + \phi_1 Y_{t-1} + \varepsilon t - \omega_1 \varepsilon_{t-1}$	ARMA model
ARIMA (1, 1, 1)	1 1 1	$\Delta Y_t = \phi_1 Y_{t-1} + \varepsilon t - \omega_1 \varepsilon_{t-1}$	ARIMA model
ARIMA (1, 1, 2)	1 1 2	$\hat{Y}_t = Y_{t-1} + \phi_1 (Y_{t-1} - Y_{t-2}) - \theta_1 e_{t-1} - \theta_1 e_{t-1}$	Damped-trend linear Exponential smoothing
ARIMA (0, 2, 1) OR (0,2,2)	0 2 1	$\hat{Y}_t = 2 Y_{t-1} - Y_{t-2} - \theta_1 e_{t-1} - \theta_2 e_{t-2}$	Linear exponential smoothing

Figure 22 - p, d, q representation and methods, part 2 (Vishwas, et al., 2020)

" (Vishwas, et al., 2020)

4.2.9 VAR model

The chapters above were focused on using the previous values of some single variable to be able to create forecasting models based on it. This chapter is about how we can use similar concepts with more than one variable available – by creating the combinational vectors out of them.

The next model that will be explained is called **vector autoregressive** model. So, using more variables in one model and analysing their behavior to determine the future:
"In the real world, we are often lucky enough to have several time series in parallel that are presumably related to one another. We already examined how to clean and align such data, and now we can learn how to make maximal use of it. We can do so by generating an AR(p) model to the case of multiple variables. The beauty of such a model is that it provides for the fact that variables both influence one another and are in turn influenced—that is, there is no privileged y while everything else is designated as x. Instead, the fitting is symmetric with respect to all variables. Note that differenc- ing can be applied as is in other models previously if the series are not stationary." (Nielsen, 2020)

"Vector autoregression (VAR) is a stochastic process model utilized to seize the linear relation among the multiple variables of time-series data. In other words, it is a multivariate forecasting method utilized when two or more time-series variables have a strong internal

relationship with each other. VAR is a bidirectional model, while others are unidirectional models. In a unidirectional model, a predictor influences the target, but not vice versa. In a bidirectional model, variables influence each other.

The normal AR(p) model equation looks like this:

$$\hat{Y}t = \mu + \phi1Yt\text{-}1 + \phi2Yt\text{-}2 \ldots + \phi pYt\text{-}p + \varepsilon t$$

where μ is intercepting, and $\phi1$, $\phi2\ldots$ ϕn are the coefficient of the lags of Y. In the VAR model, every single variable is modeled as a linear grouping of its past values and the past values of other variables in the time series. If you have multiple time series, which is determined to each other. So, one variable per equation will be designed. For instance, imagine that we have two variables of a time series, Y1, Y2. We want to forecast the value of these at time (t).

Here is the VAR (1) model with two time series (Y1 and Y2):

$$\hat{Y}1,t = \mu1 + \phi11Y1,t\text{-}1 + \phi12Y1,t\text{-}2 + \varepsilon1,t$$

$$\hat{Y}2,t = \mu2 + \phi21Y2,t\text{-}1 + \phi22Y2,t\text{-}2 + \varepsilon2,t$$

where y1,t-1, y2,t-1 are the first lag of the time series Y1 and Y2" (Vishwas, et al., 2020)

The citation above implies that the VAR model doesn't attempt to predict a single target variable, but rather a vector of them together using the relationship among them.

The chapter above should have provided the necessary overview of statistical models that are widely used for statistical analysis of time series data. There are several important concepts from statistics that did not receive a separate section in this chapter, like different measures and testing concepts, due to the focus point of this thesis being time series analysis in Python rather than statistics fundamentals. Of course, those concepts still will be leveraged in the practical part for analysis and will receive some explanation about their use in the example.

4.3   Python, Machine Learning and Deep Learning

The models that were presented in the previous chapters have great potential for prediction of the future. But, finding the optimal parameters could require many repetitive calculations, especially if we are trying to optimize them for a big number of past observations. This is when automation comes out onto the scene. The humankind has created computational machines that can do all the mathematical labor instead of us, as long as we explain it and provide clear instructions, of course.

In the year 2022 the most popular tool for automated time series analysis is Python. There are multiple reasons for this:

- **Language simplicity** – this factor shouldn't make the reader think of Python as of a "not serious" programming language, but rather one that is created and continuously developed with a focus on readability and conciseness. "Simple is better than complex" is one of the fundamental ideas expressed in zen of Python.
- **Big community support** – a large number of developers uses python in their research and projects. Many of them are playing an educational role and create great tutorials about how to put the newest ideas to use.
- **Suite of tools –** the reason for Python being leveraged in so many different areas is the number of packages that are available for free to anyone interested in working with them. The use cases are ranging from mathematical calculations to both ends of website development or even art and music.

As per the negative side of Python I would like to mention the thing that makes it strong – its interpreter. In simple terms, Python interpreter is the engine that runs the commands that are given to it. Its strength is the ability to use different functions from inside or outside packages, that saves time from writing them, but adds another level to the execution. For most tasks its performance is satisfactory and good sides very much outweigh the bad ones, but if the informational process requires some high intensity of performance, the lower-level languages might be a better alternative. Python developers also try to leverage the performance speed of lower-level languages, for example inner parts of the NumPy library

are developed in C programming language and allow achieving better performance while keeping the simple concepts of Python programming.

Some terms like package or function will be better explained in the practical example, but first there is one last topic that needs to be spotlighted, which is - deep learning.

4.3.1 Deep Learning

Nowadays the terms of machine learning and deep learning can be heard all over the internet and IT area, which might already tire some people. But there is an explanation to that – the technology is quite fascinating. Engineers around the worlds are creating systems that perform calculations which would take thousands of lifetimes of manual work. This allows them to find the best models for forecasting by quickly optimizing them and greatly increasing their complexity. The concept of using a computer for optimization of a statistical model is called machine learning and combining such models into much bigger ones is what makes them deep learning models.

Neural networks are implemented in layers of units; each layer has a specific type of units and serves some specific function. The first layer is the input layer, and it is used to store the data that we would like to present for analysis of a deep learning into the explanatory variables. The next units are so-called hidden layers, they have some sorts of an activation function inside of it and weights outside of it. Those functions and weights get optimized by
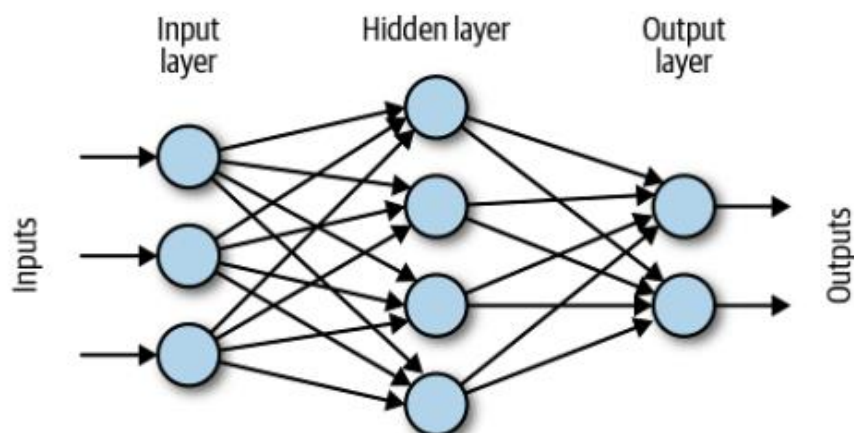

Figure 23 - A simple feed forward network (Nielsen, 2020)

trying to find the combinations of parameters that produce the lowest error. There can be

39

multiple hidden layers, of different types. Their type is decided by the developer and represents the inner computation that happens inside. And the last layer of any deep neural network is the output layer - it is where the actual answers our network provides are going to be. If our network's goal would be to detect whether there is a cat or a dog on the image – it would have two units in an output layer, one for cat and one for dog. After the photo would go through the network – these two output units would have a value between 0 and 1 which would represent the probability that it is indeed a cat or a dog.

"We can see that the input consists of three channels, or a vector of length 3. There are four hidden units. We multiply each of the three inputs by a different weight for each of the four hidden units for which it is destined, meaning that we need $3 \times 4 = 12$ weights to fully describe the problem. Also, since we will then sum the results of these various multiplications, matrix multiplication is not just analogous to what we are doing but exactly what we are doing. If we wanted to write out the steps for what we are doing, they would go something like this:

1. Input vector X has three elements. Layer 1 weights are designated by W1 , a 4×3 matrix, such that we compute the hidden layer values as $W1 \times X1$. This results in a 4×1 matrix, but this is not actually the output of the hidden layer: $W1 \times X1$

2. We need to apply a nonlinearity, which we can do with various "activation functions," such as hyperbolic tan (tanh) or the sigmoid function (σ). We will usually also apply a bias, B1, inside the activation function, such that the output of the hidden layer is really: $H = a(W1 \times X1 + B1)$

3. In the neural network depicted in Figure above , we have two outputs to predict. For this reason, we need to convert the four-dimensional hidden state output to two outputs. Traditionally, the last layer does not include a nonlinear activation function, unless we count applying a softmax in the case of a categorization problem. Let's assume we are just trying to predict two numbers, not two probabilities or categories, so we simply apply a last "dense layer" to combine the four outputs of the hidden layer per ultimate output. This dense layer will combine four inputs into two outputs, so we need a 2×4 matrix, W2 : $Y = W2 \times H$" (Nielsen, 2020)

In the practical example the development, training and predictions of LSTM network will be demonstrated. LSTM stands for Long-Short Term Memory, and it is a type of recurrent neural network. The recurrent means that the layers are not only able to pass the information forward to the next layers, but also send it back and repeat the process.

"Here is a step-by-step explanation of LSTM.

Step 1: Let's say we must predict an upcoming sequence based on all the forthcoming timestamps. In such a problem, the cell state can store all the information for the present input so that the correct prediction can be made. When we get new input data, we link it to the previous pattern in the sequence time-series data.

Here is the forget gate equation:

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right)$$

Figure 24 - Forget gate equation (Vishwas, et al., 2020)

Step 2: The next step is to make a decision on which information is important to us so we can store it. This has been classified into two parts. The first input gate layer, which contains the sigmoid layer, makes a decision on which values to update. Next, a tanH layer produces a vector for the new candidate values; this vector is called Ct. That could be further to the state. We can associate these two gates with each other to create an updated state.

Here are the input and candidate gate equations:

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right)$$
$$\tilde{C}_t = \tanh\left(W_C \cdot \left[h_{t-1}, x_t\right] + b_C\right)$$

Figure 25 - Input and candidate gate equations (Vishwas, et al., 2020)

Step 3: It is time to update the old cell state, Ct-1, into the new cell state, Ct. We multiply the old state by ft, forgetting the unnecessary parts. Then we multiply the input gate (it) by the candidate gate ( ), and this becomes the new candidate value, scaled by how much we decided to update each state value.

Here is the cell state equation:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 26 - Cell state equation (Vishwas, et al., 2020)

Step 4: We have to make a decision about the output state. This is based on a cell state, but it can be a filtered version. The first sigmoid layer makes the decision about which part of the output of the cell state we will produce; then we put a cell state to tanH and multiply it by the output of the sigmoid gate. So, we can only generate the output we make a decision on. Here are the equations for the output state and hidden state:

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 27 - Output state and hidden state equations (Vishwas, et al., 2020)

" (Vishwas, et al., 2020)

4.3.2 Development environment

As mentioned in the previous section – there are lots of different community made tools for Python programming, this chapter will provide an overview to the most commonly used ones.

IDE – stands for Integrated Development Environment. In simple terms this is a program in which the code is written. The two main approaches in Python are to use a traditional one, like Visual Studio Code or PyCharm; The second approach is to use an IPython notebook, like Jupyter notebook. The difference is that in the first approach the file represents a single piece of code that is executed all-together once the file is executed. In the IPython approach the file contains multiple blocks of code that can be executed in any desired order. PyCharm can be considered a suitable choice for beginners due to the ease of working with virtual Python environments and everything being functional out-of-the-box.

Figure 28 - Code in PyCharm

Figure 29 - Code in IPython notebook

There is another great development tool that is created using the IPython notebook technology, which is Google Colaboratory. It is essentially a **free** Google server that has Python and most of the tools the developer is going to need are installed by default (while leaving an option to install new/replace existing ones). So, if the computer is very slow or lacks storage space for the Packages – Google Colab notebook can be used, even with an enabled GPU. Though the time for using GPU is restricted to some hours daily based on the service load.

Packages and PIP – as mentioned before, one of the biggest strengths of Python is the ability to use community-made packages. A package is a bunch of code written in Python that can be used. For example, if the developer wanted to implement a Telegram chatbot using Python – they could use the official Telegram Web API documentation and write every line of their chatbot from scratch. Another option would be to use some community-made package, like python-telegram-bot, that already has everything needed to setup a working chatbot, so they could concentrate on unique aspects of its functionality. To install and manage the packages we use PIP, a very simple tool accessible through the command line.

```
!pip install --upgrade pandas_datareader

Requirement already satisfied: pandas_datareader in /usr/local/lib/python3.7/dist-packages (0.9.0)
Collecting pandas_datareader
  Downloading pandas_datareader-0.10.0-py3-none-any.whl (109 kB)
     |████████████████████████████████| 109 kB 15.7 MB/s
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.7/dist-packages (from pandas_
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.7/dist-packages (from pan
Requirement already satisfied: lxml in /usr/local/lib/python3.7/dist-packages (from pandas_datarea
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-date
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from r
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from re
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from request
Installing collected packages: pandas-datareader
  Attempting uninstall: pandas-datareader
    Found existing installation: pandas-datareader 0.9.0
    Uninstalling pandas-datareader-0.9.0:
      Successfully uninstalled pandas-datareader-0.9.0
Successfully installed pandas-datareader-0.10.0
```

Figure 30 - Updating a package to the latest version using PIP

In the practical part, a few different packages will be used. Their functionality will be explained right next to the screenshots of their use. To use some package in a program the *import* statement is used.

Figure 31 - Importing a package

Virtual Environment – Python is a language with several decades of history. It has seen dozens of versions of itself and its packages. A Python interpreter and the packages are installed together in one place that is called a virtual environment. This allows us to create multiple virtual environments to experiment or work with different versions of Python or its packages. Virtual environments can be created from command line or in PyCharm project settings. In terms of Google Colab, the whole notebook can be treated as a virtual environment.
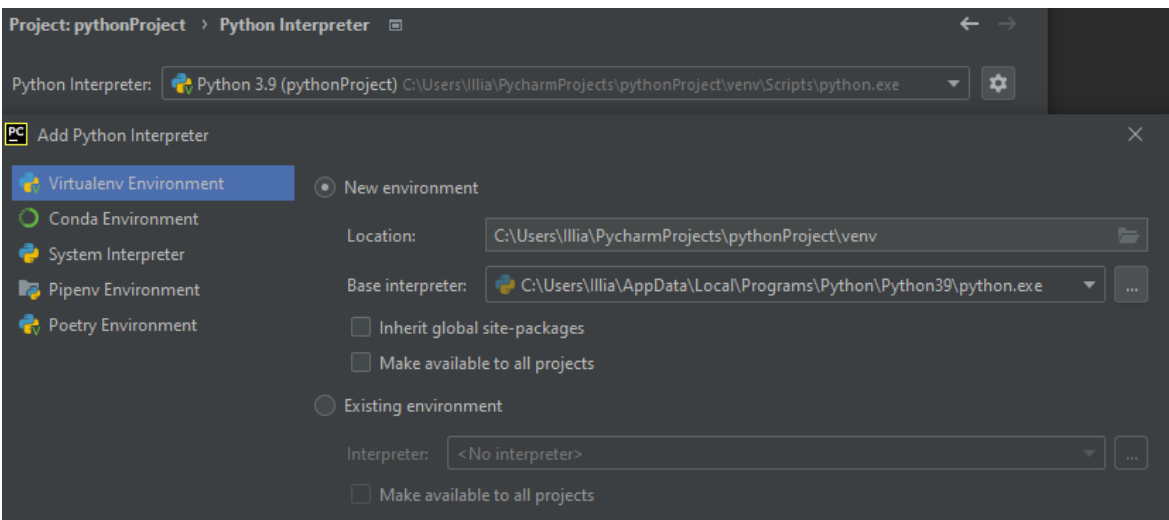
Figure 32 - Editing the virtual environment in PyCharm

The goal of the previous chapters was to explain what time-series analysis is and what models/tools are available in Python. Now it is time to put all of that to use, create some models and try to predict the future.

5 Practical Part

In this chapter the concepts from the literature are going to be demonstrated. The primary goal will be a prediction of next day's close price for some stock. In this example it will be AAPL, but the code presented could be used to experiment with prediction of some other stock by changing a single variable.

For the development environment Google Colaboratory will be used.

5.1 Setting up the colab notebook

First, the colab.research.google.com notebook needs to be created. In case the developer would want to train some deep learning models - I would recommend (like we want for this example) to enable the GPU hardware option. To do that – unhide the navigation ribbon by clicking or an arrow (circled in red), then go to runtime > change runtime type, and select GPU hardware accelerator.
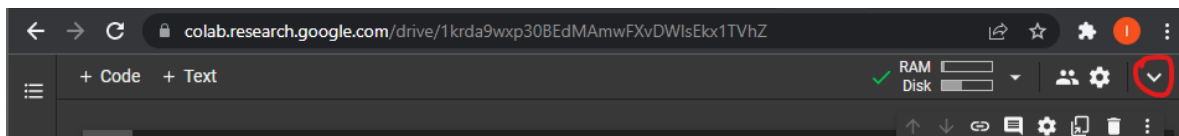
Figure 33 - Unhiding navigation ribbon on Colab
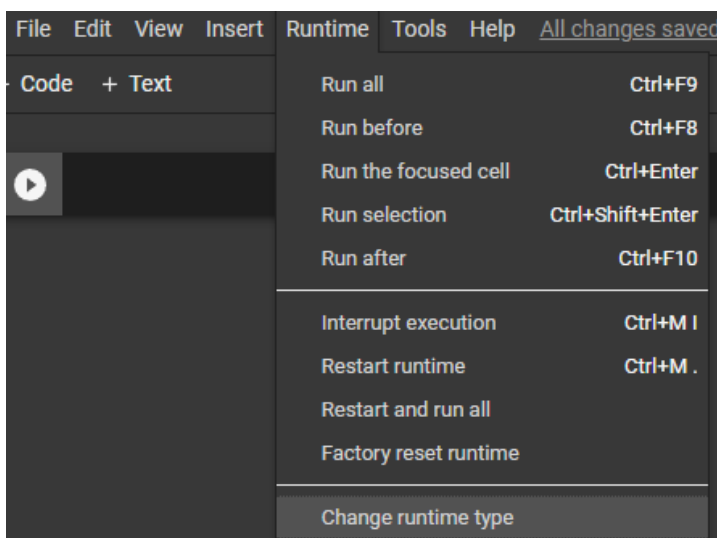
Figure 34 - Change runtime type in colab
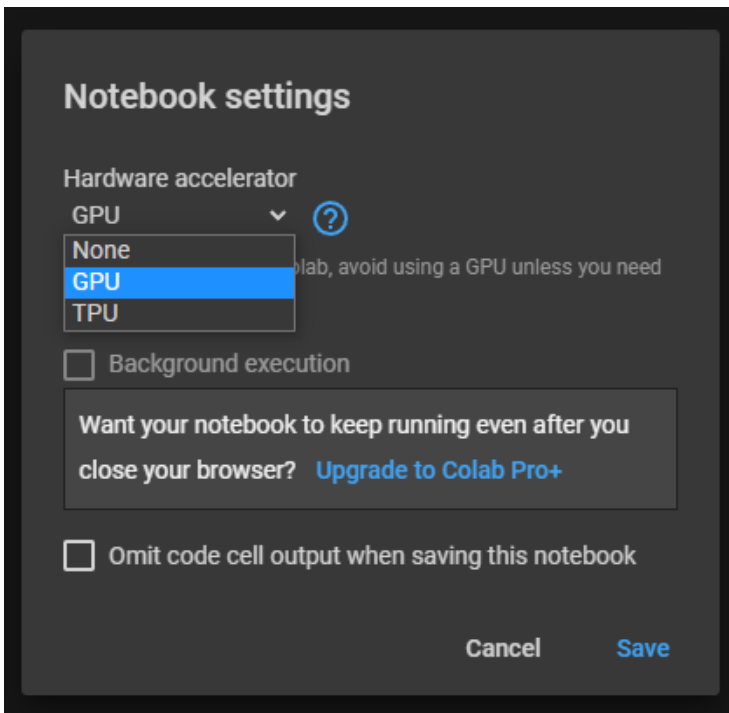
Figure 35 - Selecting GPU

Click save. That is it - our colab notebook is ready.

5.2 Getting the data

First step to predict tomorrow's values of some stock is to analyse their values from the past to understand their behavior. There is an amazing tool that allows us to create a dataset of stock prices using just one command. That tool is called "pandas_datareader".

Since Google Colab is used in this example, we don't need to install it, but the version that is available is 0.9.0 which needs to be updated to a newer one. This is very easy – just adding "--upgrade" parameter to the usual command that would be used to install it. The exclamation mark is needed only in colab to identify that it is a bash command.
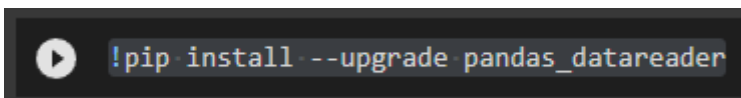
Figure 36 - Updating a package in Google Colab
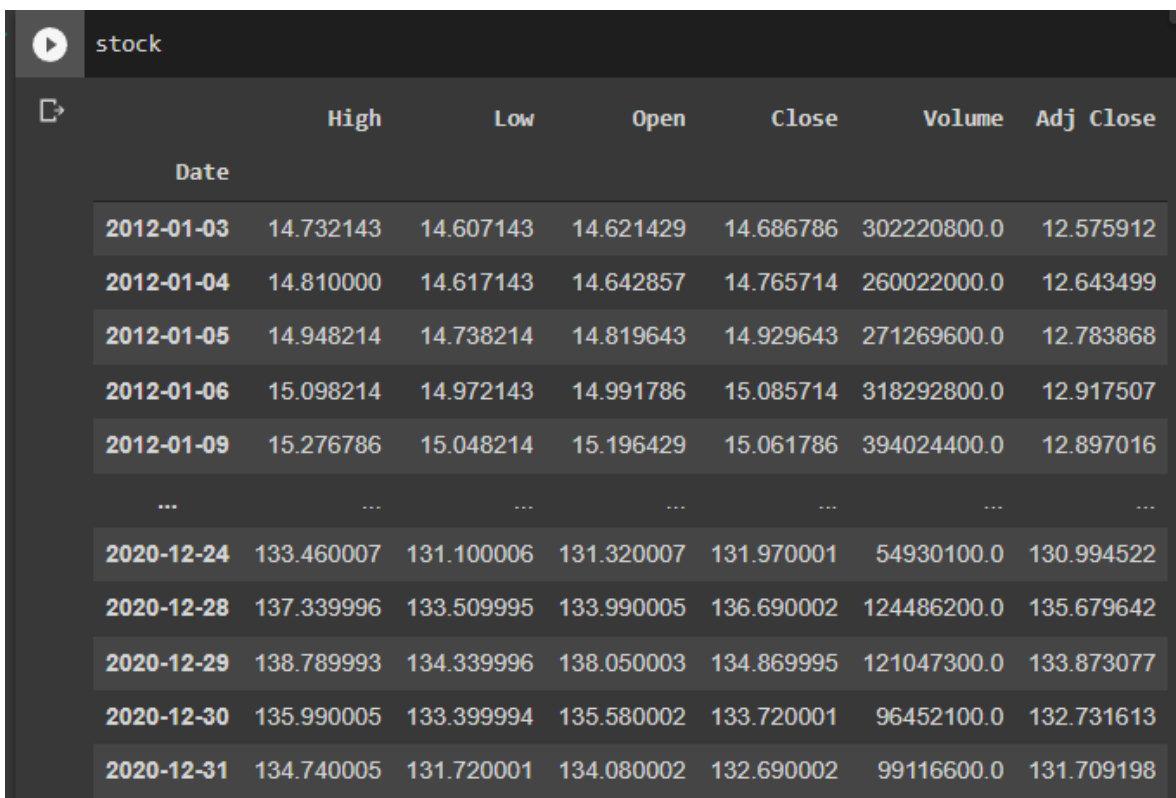
After the package has been installed, it can be imported into the project. We also need to import the datetime package that is used to create or transform timestamps. What we have to do now is to set the company stock ticker that we are interested in, and also set the first

and last day for our dataset. Once that is done too, the last thing is to use the datareader for creating our dataset.

```python
import pandas_datareader as data
import datetime as dt
company = 'AAPL' # change ticker here to select different stock
first_day = dt.datetime(2012, 1, 1)
last_day = dt.datetime(2021, 1, 1)
stock = data.DataReader(company, 'yahoo', first_day, last_day)
```

Figure 37 - Dataset creation

We have just created a Pandas dataframe without even importing Pandas. That is a package that is used to efficiently work with multidimensional tables. To see its contents run its name.

```
stock
```

Date	High	Low	Open	Close	Volume	Adj Close
2012-01-03	14.732143	14.607143	14.621429	14.686786	302220800.0	12.575912
2012-01-04	14.810000	14.617143	14.642857	14.765714	260022000.0	12.643499
2012-01-05	14.948214	14.738214	14.819643	14.929643	271269600.0	12.783868
2012-01-06	15.098214	14.972143	14.991786	15.085714	318292800.0	12.917507
2012-01-09	15.276786	15.048214	15.196429	15.061786	394024400.0	12.897016
...	...	...	...	...	...	...
2020-12-24	133.460007	131.100006	131.320007	131.970001	54930100.0	130.994522
2020-12-28	137.339996	133.509995	133.990005	136.690002	124486200.0	135.679642
2020-12-29	138.789993	134.339996	138.050003	134.869995	121047300.0	133.873077
2020-12-30	135.990005	133.399994	135.580002	133.720001	96452100.0	132.731613
2020-12-31	134.740005	131.720001	134.080002	132.690002	99116600.0	131.709198

Figure 38 - Looking at the data

As mentioned before – we want to predict the close price, which is only one column in our dataset. We can call a column from a data frame by typing its name and the column name in square brackets like this:

Figure 39 - Calling a dataframe column

5.3 Exploring the data

Now that we have our data ready, we want to find more about its statistical properties. We can find out about the measures of central tendency or variation by using the describe function of our dataframe.
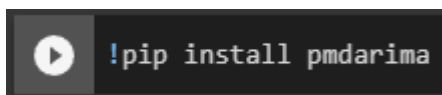
Figure 40 - Exploring the DataFrame

From the information above we can see the lowest and the highest price of AAPL since 2012, its mean, standard deviation, and 5-number summary.

Now we would like to concern ourselves with the matter of stationarity, or in other words - remove the trend from our data if there is one.
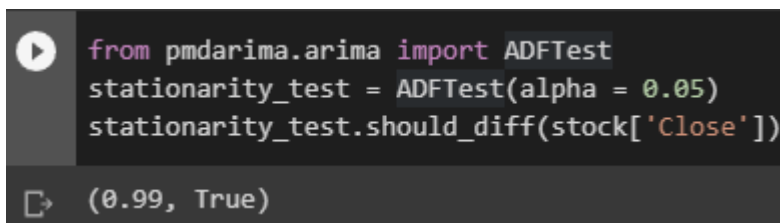
For the stocks data it is obvious that the data should not be stationary (the price is going up or down), but this thesis focuses on use of Python for time series analysis, not on how to predict stock prices, and thus it might be used for any different data, which may or may not be stationary. To test if the data needs to be differenced –a stationarity test can be used. For example, Augmented Dickey-Fuller test. It is a unit root test, which identifies whether a trend is present in time series. A package called "pmdarima" has a built-in function to perform it. Since it is not installed on Google Colab – it needs to be installed using pip.

```
!pip install pmdarima
```

Figure 41 - Installing a package in Google Colab

Now we need to import the test from the package, create an instance of it, and use its function *should_diff* on our close prices.

```
from pmdarima.arima import ADFTest
stationarity_test = ADFTest(alpha = 0.05)
stationarity_test.should_diff(stock['Close'])
```
```
(0.99, True)
```

The answer confirms the hypothesis of trend being present in our data. As it was explained in the theoretical part – to make the data stationary, differencing is used. Since our data is not seasonal (time of the year/month doesn't have a stable effect on stock price) we can do usual (nonseasonal) differencing. Luckily pandas dataframes have a built-in function to do it (the *dropna* function is used to remove the first row, it would have an empty value because there is no previous one in the dataframe to differentiate it):

```
[7]  stock['Close'].diff().dropna()

     Date
     2012-01-04     0.078928
     2012-01-05     0.163929
     2012-01-06     0.156072
     2012-01-09    -0.023929
     2012-01-10     0.053928
                      ...
     2020-12-24     1.009995
     2020-12-28     4.720001
     2020-12-29    -1.820007
     2020-12-30    -1.149994
     2020-12-31    -1.029999
     Name: Close, Length: 2264, dtype: float64
```

Figure 42 - First difference of the dataset

Now the next step would be to check once again if the data needs to undergo second differencing.

```
▶  stationarity_test.should_diff(stock['Close'].diff().dropna())

   (0.01, False)
```

Figure 43 - Stationarity check after differencing

The output demonstrates that stationarity was achieved and using first order of difference is satisfactory for our data (and thus the models). Let's explore our differenced set a bit deeper.

```
▶  first_difference = stock['Close'].diff().dropna()
   first_difference.describe()

   count    2264.000000
   mean        0.052122
   std         1.004468
   min       -10.519997
   25%        -0.208125
   50%         0.024107
   75%         0.315001
   max        10.070000
   Name: Close, dtype: float64
```

Figure 44 - Exploring differenced set

One of the things this data shows us is that the daily change in AAPL stock close price was not bigger than 10.6$ in the years from 2012 till 2021. It is time to create some plots.

Matplotlib package is used for that. This is how to create a very simple histogram plot, that will demonstrate the distribution of our data. First, we import matplotlib, then we use a command that is exclusive for Google Colab to be able to see the plots there. After that we create a histogram plot of our data with 20 columns and show it.

```
[14] import matplotlib.pyplot as plt
     %matplotlib inline
     plt.hist(first_difference, bins=20)
     plt.show()
```
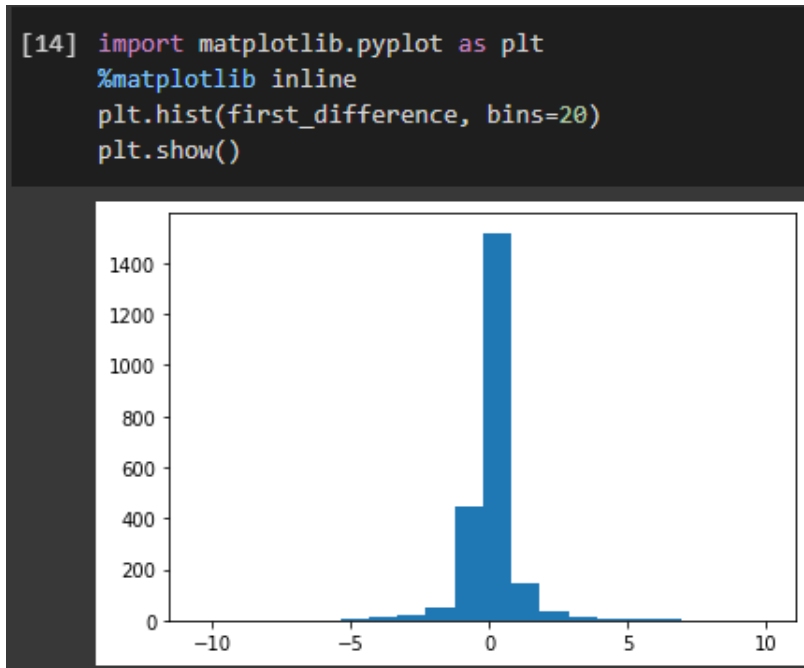

Figure 45 - Creating a distribution plot

Data is normally distributed if its fluctuation from the mean is symmetrical. In this example Jarque-Bera normality test can be used, but keep in mind that it requires a large dataset (> 2000 entries) to work. This test is available from the Scipy package, we will now use the p-value of this test to determine the test result. This value is used in any statistics test to determine the probability of the null hypothesis being true. So, when we would test if our data is normally distributed – we would get the probability of it not being normally distributed as the p-value. Here we print the second value of a list that is the output of the test, since that is where the p-value is.

```
from scipy.stats import jarque_bera
print('p-value:', jarque_bera(first_difference)[1])

p-value: 0.0
```

Figure 46 - Normality testing

Based on this test, we can see that the probability of our data being not normal is 0, so we can consider our data normally distributed.

5.4 ARIMA model

The first model that will be demonstrated is the ARIMA model. First, we will use the automatic ARIMA function from Pmdarima package to find the best suitable configuration of the model for our data. After that we will simulate the retraining and prediction on data for every day since 2021 until 2022. The parameters that are passed to the function are responsible for: data to train on, stationarity test, whether or not the data is seasonal, whether or not to show the training process in the output, whether or not to use stepwise search.

```
from pmdarima.arima import auto_arima
best_model = auto_arima(stock['Close'],
                        test='adf',
                        seasonal=False,
                        trace=True,
                        stepwise=False)

ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=6448.137, Time=0.15 sec
ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=6415.559, Time=0.58 sec
ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=6414.277, Time=0.81 sec
ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=6414.848, Time=1.12 sec
ARIMA(0,1,4)(0,0,0)[0] intercept   : AIC=6416.722, Time=1.49 sec
ARIMA(0,1,5)(0,0,0)[0] intercept   : AIC=6415.769, Time=1.86 sec
ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=6413.278, Time=0.36 sec
ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=6413.529, Time=1.36 sec
ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=6415.475, Time=3.41 sec
ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=6416.819, Time=1.86 sec
ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=6415.418, Time=8.33 sec
ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=6413.609, Time=0.69 sec
ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=6415.489, Time=2.16 sec
ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=6417.527, Time=1.76 sec
ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=6333.983, Time=10.34 sec
ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=6415.267, Time=0.80 sec
ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=6417.200, Time=2.77 sec
ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=9.63 sec
ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=6416.697, Time=1.29 sec
ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=6386.169, Time=5.53 sec
ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=6412.243, Time=1.21 sec

Best model:  ARIMA(2,1,3)(0,0,0)[0] intercept
Total fit time: 57.583 seconds
```

Figure 47 - ARIMA parameter optimization

As the result we can see that the optimal model configuration for our data is (2, 1, 3).

Last step is simulating its use (and retraining) throughout last year. First, we have to create a new dataset which actually has that data. And a separate set with only the test data for our cycle (will also be used for training next models).

```
[25] first_full_day = dt.datetime(2012, 1, 1)
     last_full_day = dt.datetime(2022, 1, 1)
     full_stock = data.DataReader('AAPL', 'yahoo', first_full_day, last_full_day)


  ▶  test_first = dt.datetime(2021, 1, 1)
     test_last = dt.datetime(2022, 1, 1)

     test_stock = data.DataReader(company, 'yahoo', test_first, test_last)
     test_stock_prices = test_stock['Close'].values
```

Figure 48 - Creating a dataset with full data and with data for testing

Now we will use Statsmodels package to create ARIMA model and predict our data from the last year using it. As it was shown in data exploration – our training dataset has 2265 entries, this number is incremented in a cycle to show on which day the training data stops.

```
  ▶  from statsmodels.tsa.api import ARIMA

     x = 2265
     arima_predictions = []
     for observation in test_stock_prices:
       mod = ARIMA(full_stock['Close'][0:x], order = (2, 1, 3))
       res = mod.fit()
       output = res.forecast()
       arima_predictions.append(output)
       x += 1
```

Figure 49 - simulating ARIMA model's use

The first thing to do now would be to plot our predictions. Here is how to create a usual line plot with two observed variables using matplotlib.

```
plt.plot(test_stock_prices, color='green', label='Real data')
plt.plot(arima_predictions, color ='black', label='ARIMA Prediction')
plt.title(f'{company} Price')
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()
plt.show()
```
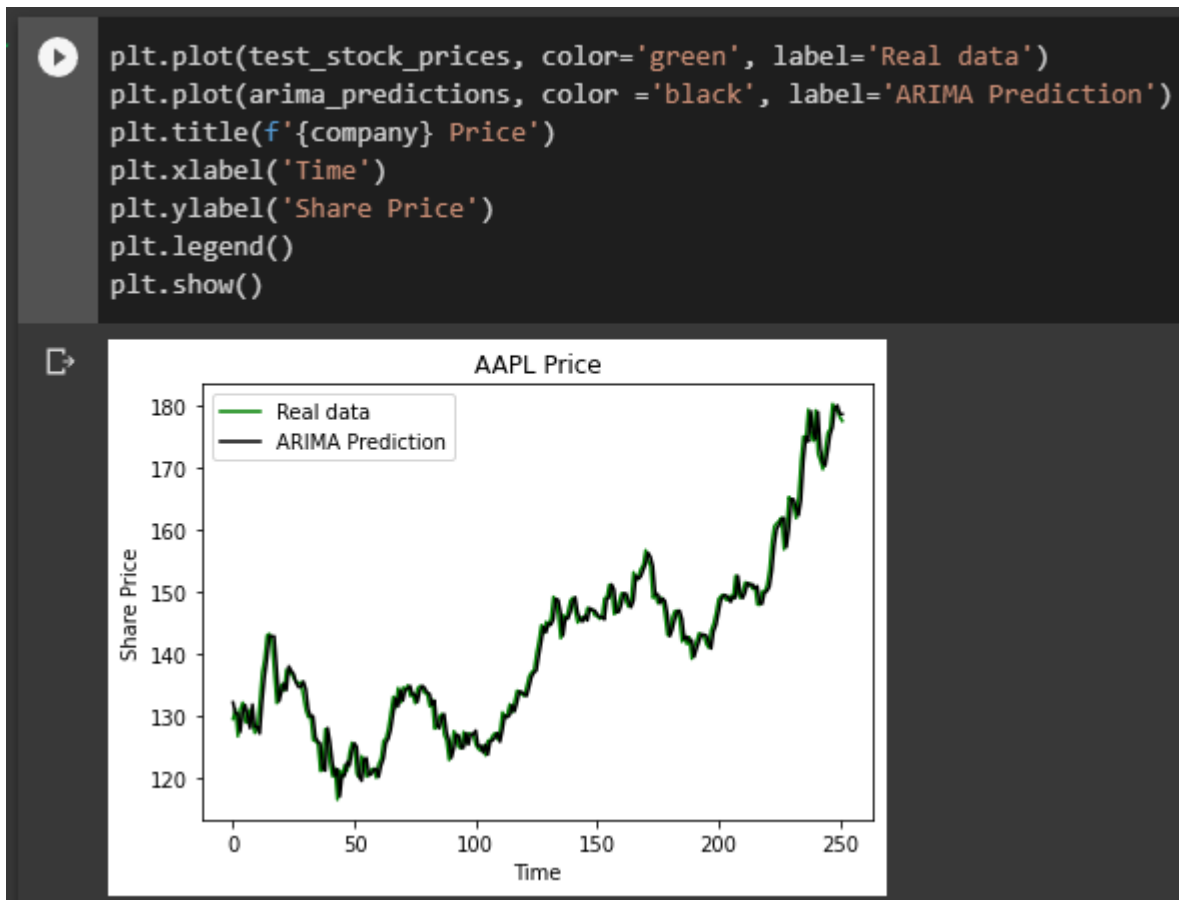

Figure 50 - Plotting the resulting predictions

On this plot we can see that the model is doing very well with predicting what price would it be tomorrow. Let's find the error in our predictions and plot it:

```
[▶]  arima_residual = []
     i = 0
     for value in test_stock_prices:
         arima_residual.append(value - arima_predictions[i])
         i += 1
     plt.plot(arima_residual, color='red', label='Error')
     plt.title(f'{company} Price')
     plt.xlabel('Time')
     plt.ylabel('Share Price')
     plt.legend()
     plt.show()
```
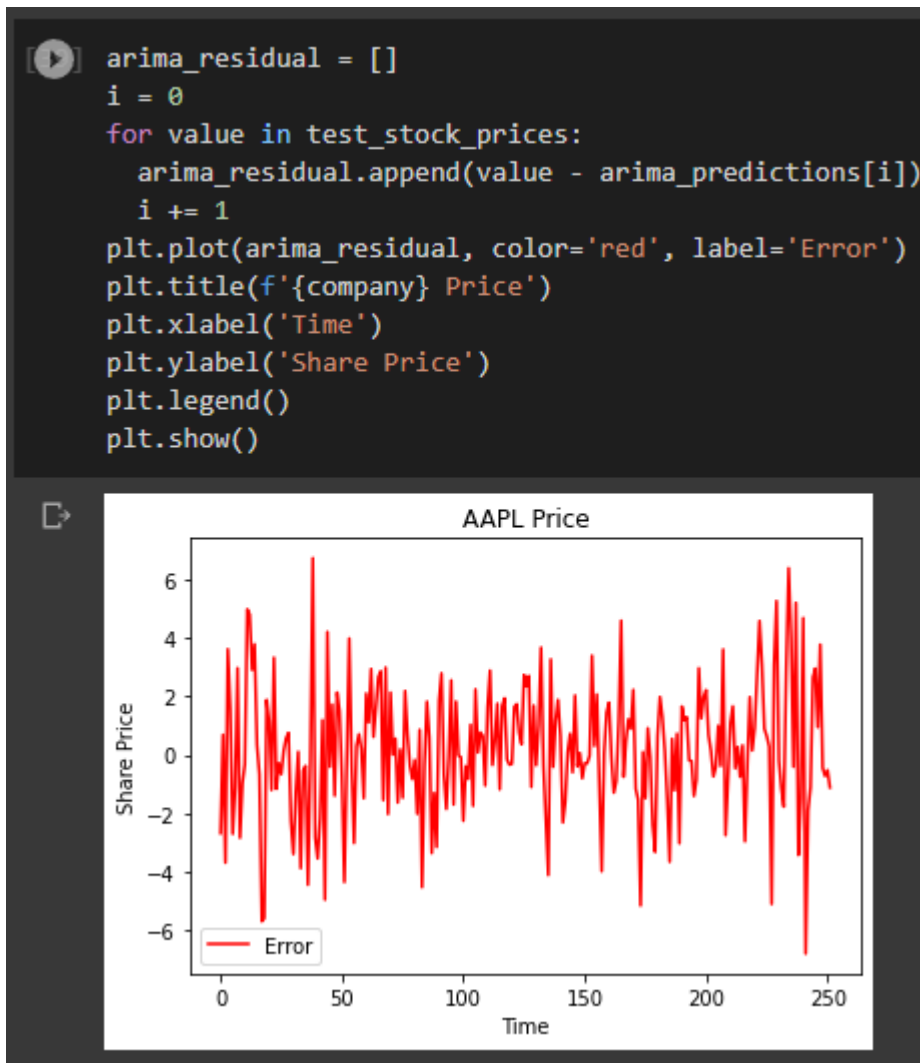

Figure 51 - Finding and plotting the residuals

Now Pandas package is actually imported to convert the list of residuals into a dataframe in order to explore them. Before doing that the list of lists with prediction results is joined into one list.

```
[65] import pandas as pd
     arima_residual_joined = [x for l in arima_residual for x in l]
     pd.DataFrame(arima_residual_joined, columns=["Residual"]).describe()
```

	Residual
count	252.000000
mean	0.186529
std	2.240197
min	-6.790964
25%	-1.104357
50%	0.102480
75%	1.672870
max	6.729006
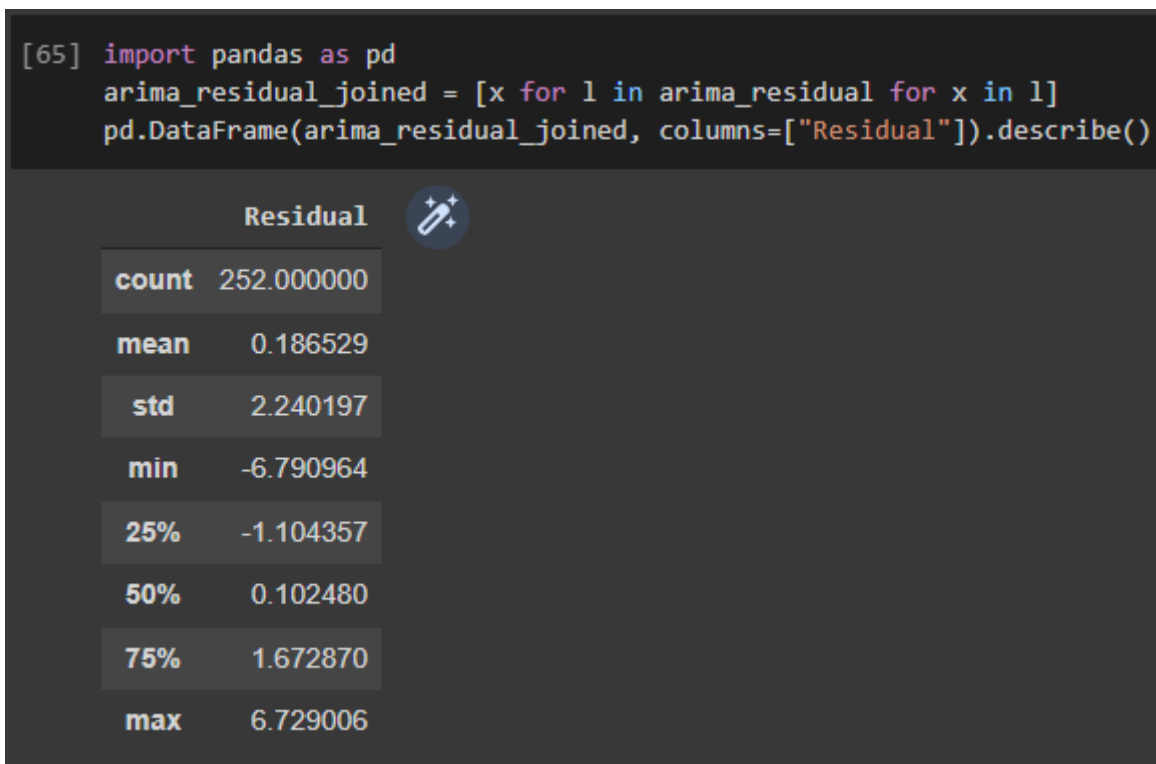
Figure 52 - Exploring the residuals

This info shows us that the average error is 0.18 dollars. We can use r^2 (coefficient of determination) testing to see how much of our real data's variation was explained by the predictions. One of the packages that includes it is Scikit-learn.

```
from sklearn.metrics import r2_score
r2_score(test_stock_prices, arima_predictions)

0.9764851770813001
```

Figure 53 - R-square testing

The results show that a very high value of r^2 coefficient was achieved suggesting that our model shows great results being correct for 97%.

After final evaluation of the models the reader will find out why the statement above should be taken very-very skeptically.

5.5 LSTM model

The next model will try to use the same data in a more complex manner. It is a sequential LSTM model. The idea is to give it an opportunity to look back at three months of past data so that it would try to predict the price tomorrow. To do that we need to create two sets where 90 previous days in set x will be corresponding to one actual day in set y. Beforehand the dataset will be scaled to limits from 0 to 1 according to its minimum and maximum values, an important step of preprocessing that optimizes the calculations.

```python
import numpy as np
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
scaled_stock = scaler.fit_transform(stock['Close'].values.reshape(-1,1))
prediction_days = 90
x_train = []
y_train = []
for x in range(prediction_days, len(scaled_stock)):
  x_train.append(scaled_stock[x-prediction_days:x, 0])
  y_train.append(scaled_stock[x, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
```
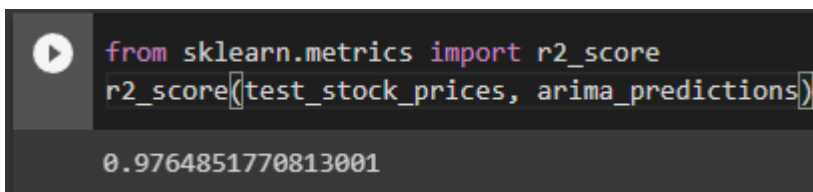
Figure 54 - Scaling the data and splitting it into exploratory and target sets

Now we need to define the layers of the neural network and and train it on our data. This is done using the Google's package TensorFlow and its higher-level interface Keras.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM
model = Sequential()
model.add(LSTM(units=64, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=64, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=64))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, epochs=32, batch_size=32)

Epoch 3/32
68/68 [==============================] - 4s 64ms/step - loss: 0.0012
```

Figure 55 - Definition and training of a TensorFlow model

After the training process is done the model needs to be tested on the data that it has not yet seen before. Note that it will be trained only once unlike the ARIMA model and thus won't tweak its parameters to compensate for the new data. To do that we are assembling a testing set of x with the same logic as in the training one (we have already created the y set as the testing one for ARIMA, and it is also present in the full stock).

```python
model_inputs = full_stock[len(full_stock) - len(test_stock) - prediction_days:].values
model_inputs = model_inputs.reshape(-1,1)
model_inputs = scaler.transform(model_inputs)

x_test = []

for x in range(prediction_days, len(model_inputs)):
  x_test.append(model_inputs[x-prediction_days:x, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
predicted_prices = model.predict(x_test)
predicted_prices = scaler.inverse_transform(predicted_prices)
```

Figure 56 - Testing of a TensorFlow model

The last step on the image above uses the parameters that were saved in the scaler upon initial transformation of our data to reverse-scale the prediction from a form of 0 to 1 to a usual one.

After the testing is done, we can plot the predictions.

```
plt.plot(test_stock_prices, color='green', label='Real data')
plt.plot(predicted_prices, color ='red', label='LSTM Prediction')
plt.title(f'{company} Price')
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()
plt.show()
```
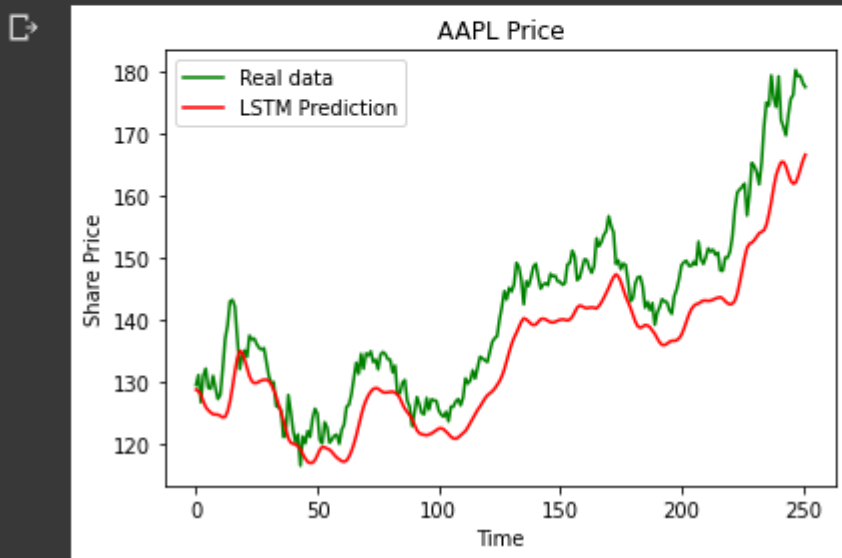

From looking at the plot we can see that the model performs worse compared to the ARIMA model due to not being retrained every time.

The residuals can be found and plotted in the same manner:

```
lstm_residual = []
i = 0
for value in test_stock_prices:
    lstm_residual.append(value - predicted_prices[i])
    i += 1
plt.plot(lstm_residual, color='red', label='LSTM Residual')

plt.title(f'{company} Price')
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()
plt.show()
```


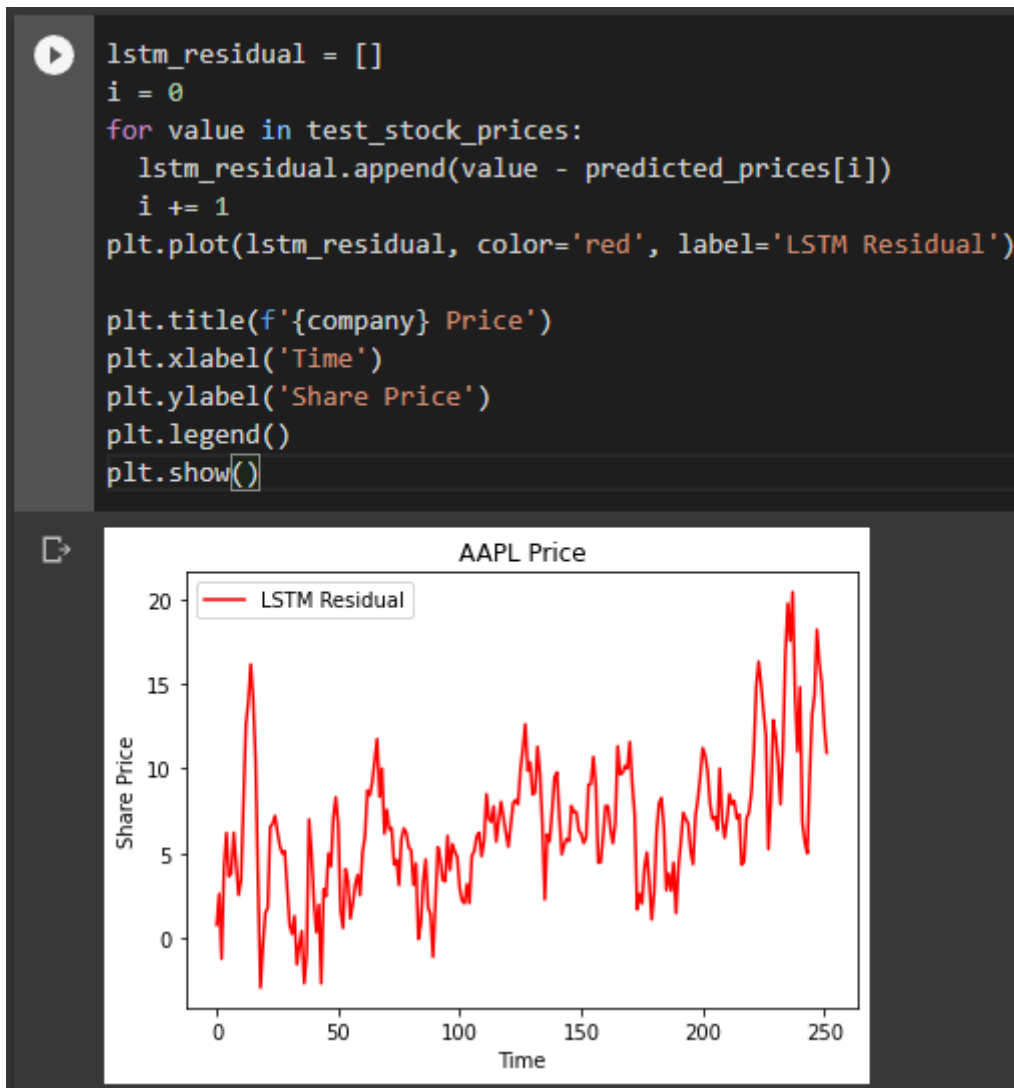*Figure 57 - Finding and plotting LSTM error*

Now let's explore the residuals and find the coefficient of determination.

The info shows that the average error is of 6$, with the biggest one being 20$.

And based on the coefficient of determination we can conclude that the model explained only 72% of the target variable variation, which is 3.5 times worse than in case of the previous approach.

```
[72] lstm_residual_joined = [x for l in lstm_residual for x in l]
     pd.DataFrame(lstm_residual_joined, columns=["Residual"]).describe()
```

	Residual
count	252.000000
mean	6.549322
std	4.087192
min	-2.947235
25%	4.055347
50%	6.218758
75%	8.499086
max	20.442856
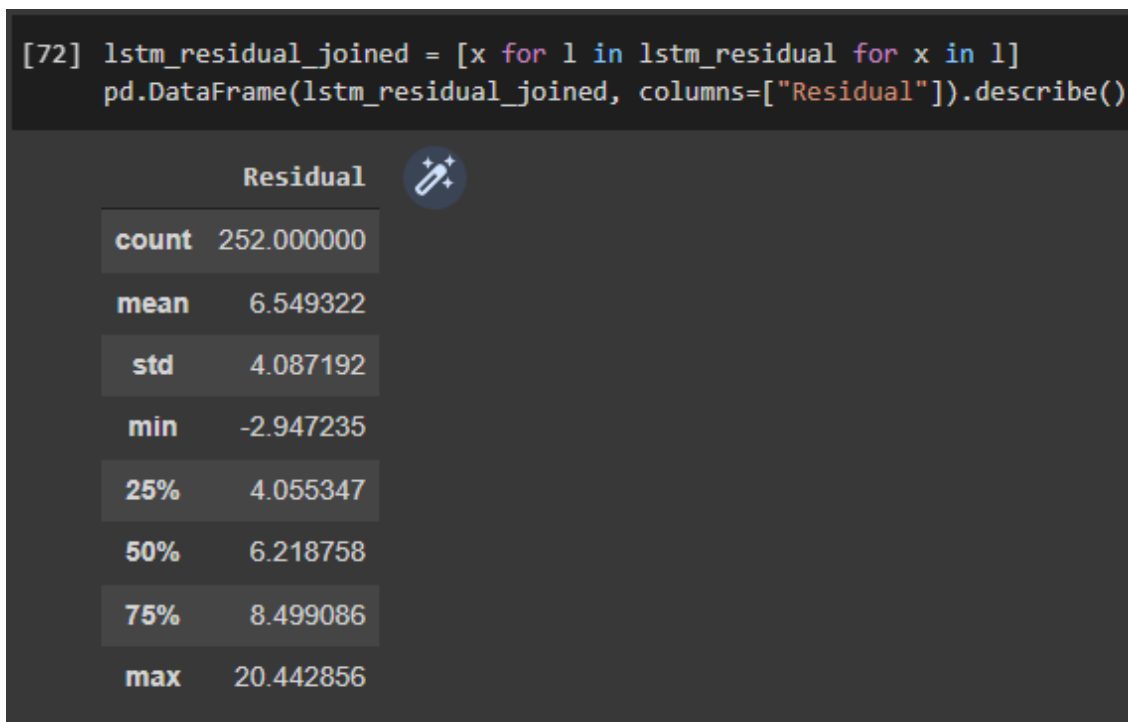
Figure 58 - Exploring LSTM residual

```
[73] r2_score(test_stock_prices, predicted_prices)

     0.7218766048476858
```
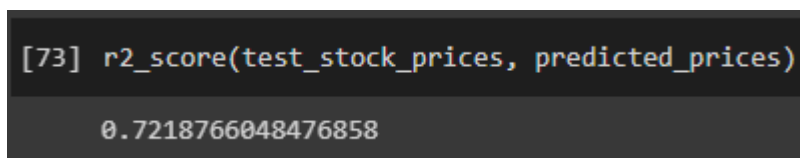
Figure 59 - R-squared coefficient of LSTM predictions

5.6 VAR model

The last approach to be demonstrated is creation of a system of multiple variables using VAR model. For this example the past data for two competitor companies will be used as additional variables in an attempt to check if predicting tomorrow's value for three companies together and then extracting the value for a companies of our interest would work better than using a single one.

To begin, we need to construct a dataset with the new stocks. For the example of AAPL, MSFT and GOOGL stocks were selected.

```
company2 = 'MSFT'
company3 = 'GOOGL'
var_stock = data.DataReader([company, company2, company3], 'yahoo',
                              first_full_day, last_full_day)
```

Figure 60 - Constructing a dataset of three stocks

Explanatory variables are also called independent, this name implies that we don't want them to have a relation between each other. We can check it for our new dataset using the bult in *corr* function of pandas dataframe.

```
var_stock['Close'].corr()
```

Symbols	AAPL	MSFT	GOOGL
Symbols			
AAPL	1.000000	0.982164	0.956946
MSFT	0.982164	1.000000	0.974940
GOOGL	0.956946	0.974940	1.000000

Figure 61 - Correlation matrix using pandas

As we can see the variables are indeed very dependent on each-other, it is better to have correlation of less than 80%. Apart from that, it most probably is not stationary. Luckily, the first differencing will deal with both issues.

```
var_differenced = var_stock['Close'].diff().dropna()
var_differenced.corr()
```

Symbols	AAPL	MSFT	GOOGL
Symbols			
AAPL	1.000000	0.696896	0.569406
MSFT	0.696896	1.000000	0.734329
GOOGL	0.569406	0.734329	1.000000

Figure 62 - Effect of first differencing

As we can see the correlation issue was resolved, so we can continue with creating the VAR model. This is done by using the Statsmodels package. After the model would be created, we have to identify the optimal order for our data, it is limited to 90 as that's the order used in LSTM example.

```
from statsmodels.tsa.api import VAR
var_model = VAR(var_differenced)
var_order = var_model.select_order(maxlags=90)
var_order.summary()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/
    self._init_dates(dates, freq)
        VAR Order Selection (*
      highlights the minimums)
        AIC    BIC    FPE   HQIC
    0 6.322  6.330  556.9  6.325
    1 6.282  6.311* 534.8  6.292
```

Figure 63 - Selecting the optimal order for VAR model

In the summary the best resulting value of several measures is achieved. One of them is AIC which stands for Akaike information criterion. It is used to determine how well the model represents the data. It was already used in the auto ARIMA model to select the optimal parameters. The minimal value for AIC was achieved by using an order of 84, so that is the selected one for this model. Now the usage of such model needs to be simulated by retraining it every day on the new data for the past year.

```
lag_order = 84
x = 2265
var_predictions = [[],[],[]]
for obs in test_stock_prices:
  var_model = VAR(var_differenced[0:x])
  results = var_model.fit(lag_order)
  x += 1
  var_prediction = results.forecast(var_differenced.values[-lag_order:], 1)
  var_predictions[0].append(var_prediction[0][0])
  var_predictions[1].append(var_prediction[0][1])
  var_predictions[2].append(var_prediction[0][2])
```

Figure 64 - Simulating the usage of VAR model

Now it is needed to convert the list of lists of predictions into a dataframe and do the reverse differencing.

```
var_predictions = pd.DataFrame({company:var_predictions[0],
                                company2:var_predictions[1],
                                company3:var_predictions[2],})
var_predictions.index = var_stock['Close'][2264:-1].index

reverse_p = var_stock['Close'][2264:-1] + var_predictions
reverse_p = reverse_p.iloc[1: , :]
```

Figure 65 - Converting list into DataFrame and reversing the difference

Now the predictions need to be plotted.

```
plt.plot(test_stock_prices, color='green', label='Real data')
plt.plot(reverse_p['AAPL'].values, color ='orange', label='VAR Prediction')
plt.title(f'{company} Price')
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()
plt.show()
```
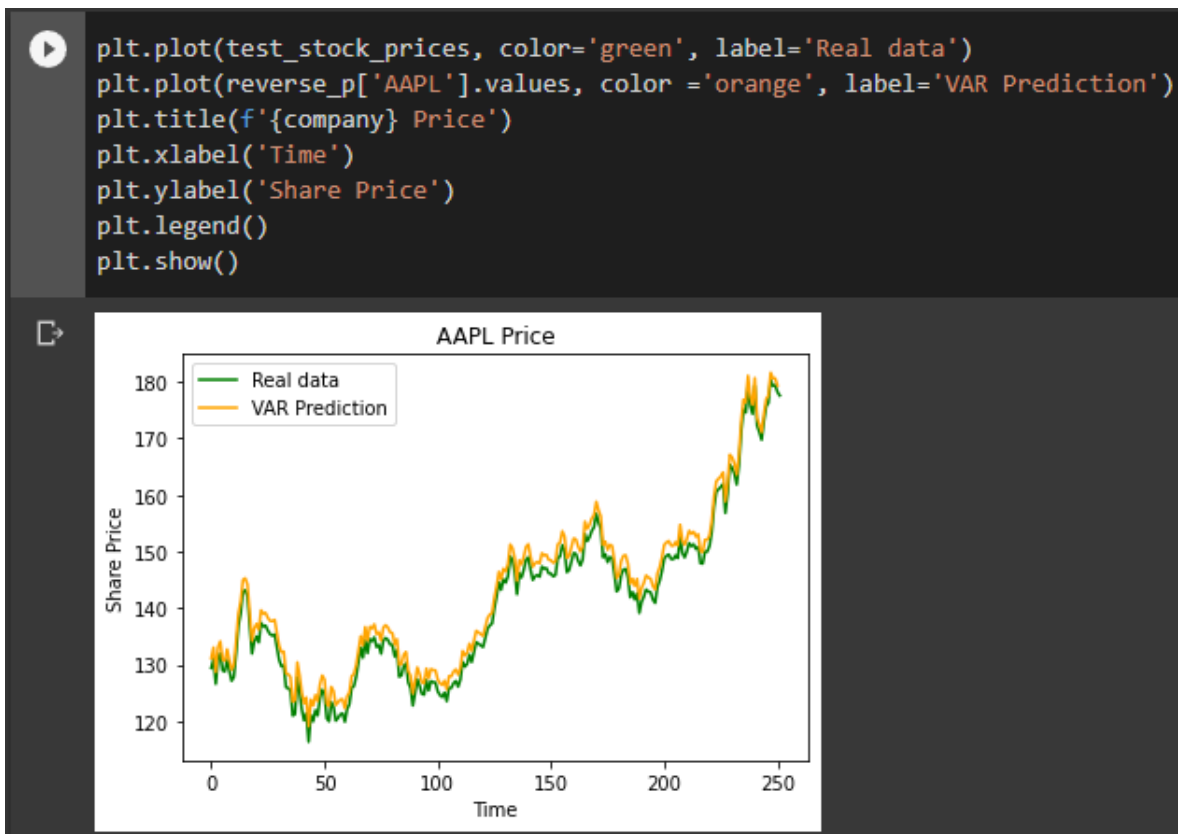

Figure 66 - Plotting VAR predictions

On this plot we can see that the model is overestimating the price. Let's find the residuals and explore them once again.
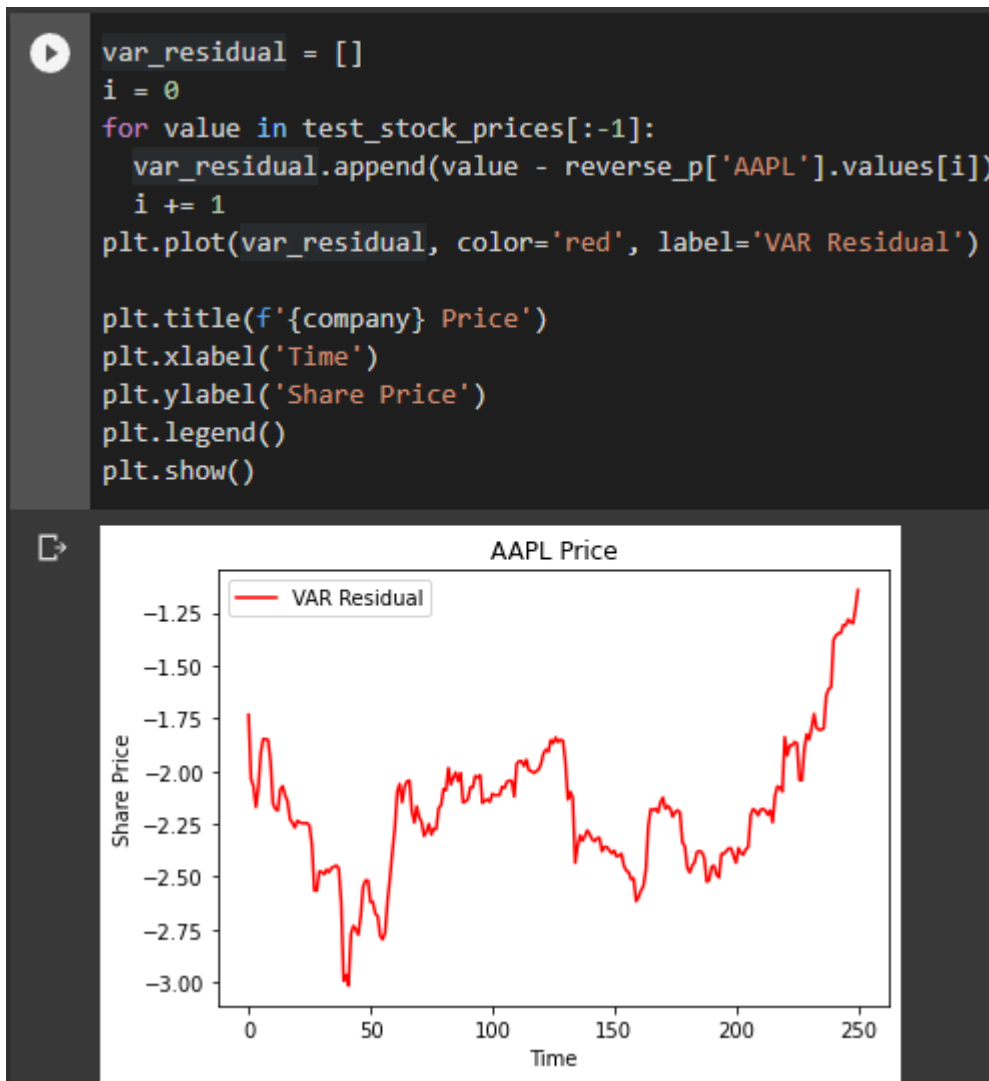
```python
var_residual = []
i = 0
for value in test_stock_prices[:-1]:
    var_residual.append(value - reverse_p['AAPL'].values[i])
    i += 1
plt.plot(var_residual, color='red', label='VAR Residual')

plt.title(f'{company} Price')
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()
plt.show()
```


Figure 67 - Calcualting and plotting VAR residuals

The negative error confirms our observation of model's overestimation.

The exploratory info below shows that the average error is negative 2$, and the maximum one being of negative 3$. The R-square coefficient states that the model explains 95% of the target variable variation.

```
[95] pd.DataFrame(var_residual, columns=["Residual"]).describe()
```

	Residual
count	251.000000
mean	-2.185242
std	0.316267
min	-3.016717
25%	-2.389382
50%	-2.185800
75%	-2.034439
max	-1.142536

Figure 68 - Exploring VAR residuals

```
r2_score(test_stock_prices[1:], reverse_p['AAPL'])
0.9575124243899172
```

Figure 69 - R-squared calculation for VAR predictions

6 Results and Discussion

Two of the three models have shown quite a significant accuracy of evaluation. Though, they are not suitable for prediction of future stock price. Here is why:

6.1 Strategic evaluation of the models

The primary issue is common sense. The price of a stock is not determined by its previous price, nor by the previous price of its competitors. Even excluding their trend characteristic there are no specific cycles that would determine its behavior. The price of a stock is only dependent on the market situation, which is affected by various factors that are much more significant than the previous price behavior. Those factors include things like business decisions of the company, and political situation. The words above can be easily demonstrated by taking a closer look at the predictions of the models.

```python
plt.plot(test_stock_prices[len(test_stock_prices)-20:len(test_stock_prices)+1],
         color='green', label='Real data')
plt.plot(predicted_prices[len(test_stock_prices)-20:len(test_stock_prices)+1],
         color ='red', label='LSTM Prediction')
plt.plot(arima_predictions[len(test_stock_prices)-20:len(test_stock_prices)+1],
         color ='black', label='ARIMA Prediction')
plt.plot(reverse_p['AAPL'].values[len(test_stock_prices)-21:len(test_stock_prices)+1],
         color ='orange', label='VAR Prediction')
plt.title(f'{company} Price')
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()
plt.show()
```

Figure 70 - Creating a plot with all predictions

On the plot below we will be able to find the predictions for last twenty days of using the models. If we observe the behavior of VAR and ARIMA predictions, we will conclude that it looks almost like a lagged variable. Every time the stock price goes down it takes them a day to understand them and simply readjust for it.

The reason for it was already stated – it is the irrelevance of past price to the current one.
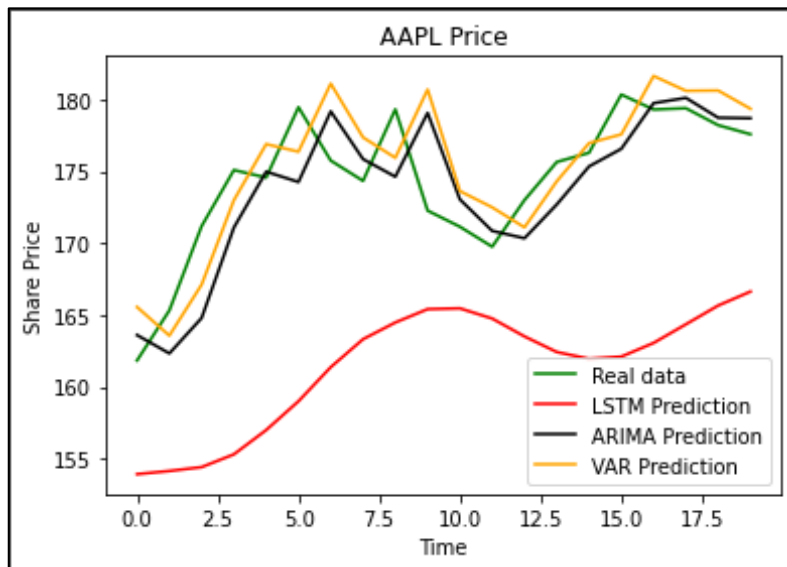
Figure 71 - Plot of all predictions

The ARIMA and VAR models can be of a great help when working with data that has more cyclic characteristics and less random behavior. Be aware that each of those models, especially the LSTM one, would greatly benefit from inclusion of some additional variables that have more relevance to the company's performance, like its quarterly earnings results (or the difference in them). This is the information that actually influences the stock price and could be very beneficial to determine it in the future.

7 Conclusions

Although the results outlined in previous chapter might disappoint ambitious investors, the goal of this thesis was to introduce several techniques for time series analysis and to demonstrate how to use statistical forecasting models in Python, but not how to build an investment strategy. This goal was achieved in its fullness.

The initial chapters of this thesis have described the context and history of time measurement and explained the statistical and mathematical foundations for their analysis. The theoretical examples used in the thesis demonstrated the differences among various algorithms for time series forecasting. The ease of reading and writing Python code was demonstrated by a practical study. In that example, all of the important stages of analytical software development were covered:

1. Selecting appropriate tools

2. Setting up the programming environment

3. Getting the data

4. Exploring the data

5. Model implementation

6. Model evaluation

As the author I hope that this work would be valuable to a future reader and could serve them as a beginners guide on: What is time series analysis? Where did it come from? And how to approach it?

8 References

Google Trends. [Online] 2022. https://trends.google.com/.

Devlin, Keith. *About time.* 1999, Devlin's Angle.

Marill, Keith A. *Advanced Statistics: Linear Regression, Part I:.* 2004, ACAD EMERG MED, pp. 87-93.

Daderot. File:Water clockm Egypt, Ptolematic Period.. *Wikimedia Commons.* [Online] August 22, 2014.

Elser, Veit. File:Time series of norm of difference-map increment Δ, during solving random 3-SAT instance.png. *commons.wikimedia.org.* [Online] 2007. https://commons.wikimedia.org/wiki/File:Time_series_of_norm_of_difference-map_increment_%CE%94,_during_solving_random_3-SAT_instance.png.

Donthi, Ranadheer, et al. *Estimation Methods of nonlinear regression models.* 2019, AIP Conference Proceedings.

Frazel, Thomas. *Going Back in Time: The History of Timekeeping.* 2013, English Composition Program Director, pp. 15-21.

Halswanter, Thomas. File:Residuals for Linear Regression Fit.png. *Wikimedia Commons.* [Online] January 30, 2013. https://commons.wikimedia.org/wiki/File:Residuals_for_Linear_Regression_Fit.png.

Hamilton, James D. 1994. *Time Series Analysis.* Princeton : Princeton University Press, 1994. ISBN: 0-691-04289-6.

Heitordp. File:World time zones.svg. *wikimedia commons.* [Online] March 2021, 23. https://commons.wikimedia.org/wiki/File:World_time_zones.svg.

Kowarschick, Wolfgang. File: Normal Distribution Sigma.svg. *Wikimedia Commons.* [Online] October 4, 2012. https://commons.wikimedia.org/wiki/File:Normal_Distribution_Sigma.svg.

Manning, Jeremy. File:Acf new.svg. *Wikimedia Commons.* [Online] June 30, 2009. https://commons.wikimedia.org/wiki/File:Acf_new.svg.

Tranmer, M, et al. *Multiple Linear Regression (2nd Edition).* 2020, Cathie Marsh Institute Working Paper.

Nielsen, Aileen. *Practiacal Time Series Analysis Prdeiction with Statistics & Machine Learning.* Sebastopol : O`Reilly Media, Inc., 2020. ISBN: 978-1-492-04165-8.

Vodolazhskaya, Larisa. *Reconstruction of anctient Egyptian sundials.* 2014, Journal Archaeostronomy and Ancient Technologies.

Vincent, Adrien F. File:Mpl exaple scatter plot.svg. *Wikimedia Commons.* [Online] September 26, 2016.
 https://commons.wikimedia.org/wiki/File:Mpl_example_scatter_plot.svg.

Vishwas, B V and Patel, Ashish. *Hands-on Time Series Analysis with Python.* s.l. : Apress, 2020. ISBN: 978-1-4842-5992-4.

9   Appendix

Link to colab notebook:

https://colab.research.google.com/drive/1f0p12H3RBL6yzHxiT97rXpPSEudnZP4_?usp=sharing