Capstone Project: Investment and Trading

Project Overview:

This project is about the idea of using AI to predict stock trends for different time frames.
It combines methods to download and explore data from different tickers, to get an idea of how well a stock price is performing in the future. For that, it uses multiple ML-models to conduct the predictions.

The program consists of multiple python classes and a Jupter notebook, which is divided into three main sections:

- Prepare Data (Download and Clean and calculate features)
- Analysis Datasets (Provides statistical numbers and visualizations to explore the dataset)
- Forecasting (Performs predictions via KI)

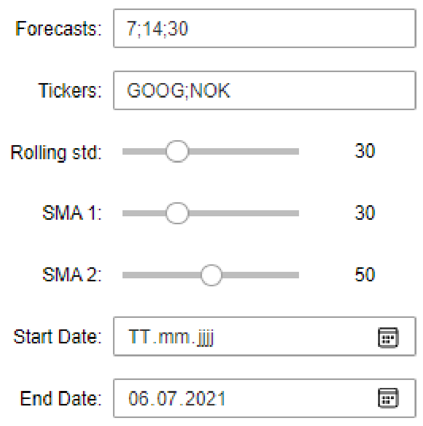
The notebook provides widgets, which enables the user to set a list of ticker symbols, statistical values, a range for historical data load and time frames by which the prediction should happen.

During the implementation of this program, I was going to use the yfinance API, which provided daily historical data for a large kind of tickers. Unfortunately, the API is facing some issues since 2nd of July 2021, thus, I decided to download CSV files from the tickers Nvidia and Daimler AG ranging from 22.02.2010 until the 5th of July 2021.
Once the API is back online, the notebook should make automatically use of it.
As a standard setting, the program provides forecasts for 7, 14 and 30 days.

Based on the given architecture, the classes can be used to build a web or mobile app later.

Data Preprocessing:

To grep data, the program is using the "yfinance API". It queries stock information from yahoo. To start with, the user adds ticker symbols into the form field and specifies the time range for the historical data load.



Forecasts:	7;14;30
Tickers:	GOOG;NOK
Rolling std:	30
SMA 1:	30
SMA 2:	50
Start Date:	TT.mm.jjjj
End Date:	06.07.2021

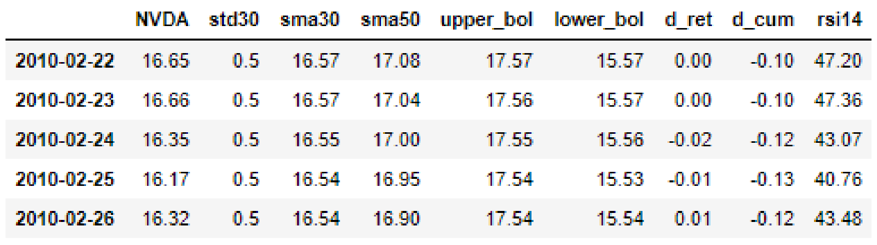
Figure 1: Set parameters

The API returns a data frame, which consist of the date as the index, “open-price”, “min-price”, “max-price”, “close-price” and the “volume” of traded stocks.

Additionally the user can set up multiple forecast windows and adjust some statistical rolling indicators, which are part of the features columns the KI is learning with.

During the data processing, the program provides in total seven different indicators:

- Rolling Standard deviation
- Simple moving average (two)
- Upper- lower Bollinger band
- Daily returns
- Cumulative returns
- Relative Strength Index (RSI)



	NVDA	std30	sma30	sma50	upper_bol	lower_bol	d_ret	d_cum	rsi14
2010-02-22	16.65	0.5	16.57	17.08	17.57	15.57	0.00	-0.10	47.20
2010-02-23	16.66	0.5	16.57	17.04	17.56	15.57	0.00	-0.10	47.36
2010-02-24	16.35	0.5	16.55	17.00	17.55	15.56	-0.02	-0.12	43.07
2010-02-25	16.17	0.5	16.54	16.95	17.54	15.53	-0.01	-0.13	40.76
2010-02-26	16.32	0.5	16.54	16.90	17.54	15.54	0.01	-0.12	43.48

Figure 2: Sample dataframe after data preparation

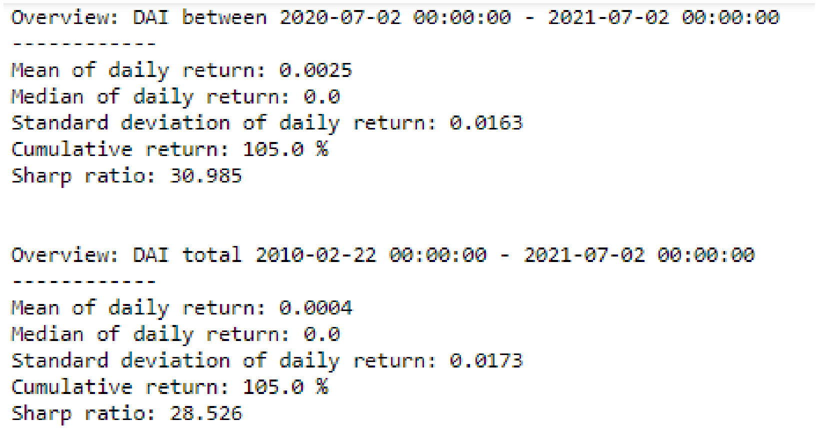
However, before I going to calculate the features above, I do some data cleansing tasks to provide a dataset, that is better for analyzation and visualization.
Usually, the API only provides data from trading days. That means, if you would make a visualization of the stock trend, there would be some gaps. Because of that, I am going to pick the first and the last date index of the downloaded dataset and create a new data-frame with the entire date range, hence weekends and public holidays are also included. Next, I join the given dataset with the new data-frame and using the front fill method to provide a continuous data.

Data Exploration & Visualization:

To get some insight of the given data sets, the section “Analysis Datasets” provides global statistical data like mean of daily return, the cumulative return, which describes the total return of a stock since the beginning of the record, standard deviation or the sharp risk ration.
A lower number of the standard deviation indicates if the stock is of less risk of high variability or better say, the volatility of a stock.

The sharp risk ratio indicates if a stock could generate a higher return in comparison to a risk free market investment like putting money to a bank account with some interest. As a conclusion. The higher number of the sharp risk ratio, the better are returns over time.

For each stock, I provide global statistics for a one-year period and for the entire dataset.
The one-year observation is actually interesting to see how a stock is performing lately.
The total number is more important for the prediction, which I going to explain below in this text.



```
Overview: DAI between 2020-07-02 00:00:00 - 2021-07-02 00:00:00
------------
Mean of daily return: 0.0025
Median of daily return: 0.0
Standard deviation of daily return: 0.0163
Cumulative return: 105.0 %
Sharp ratio: 30.985


Overview: DAI total 2010-02-22 00:00:00 - 2021-07-02 00:00:00
------------
Mean of daily return: 0.0004
Median of daily return: 0.0
Standard deviation of daily return: 0.0173
Cumulative return: 105.0 %
Sharp ratio: 28.526
```

Figure 3:Global statistics

The program also provides a bunch of visualizations for the data analyzation part. The first visual denotes the trend of the current stock combined with additional indicators.

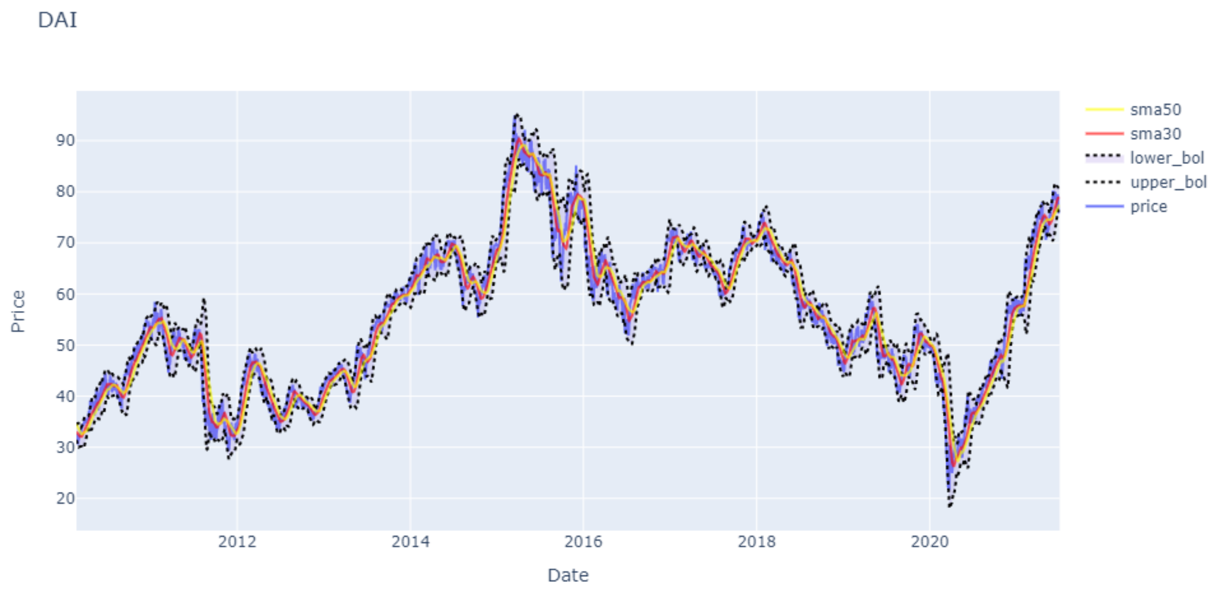


Figure 4: Stock trend of Daimler in total

I used ploty to be able to drill into the figure. A closer look shows how the actual price is moving between the “Bollinger Bands”. A trading strategy says, buy if the stock price is below the lower Bollinger band, and sell if the stock price is above the upper Bollinger band.
- An automatically notification could be actually a nice extension for this program.

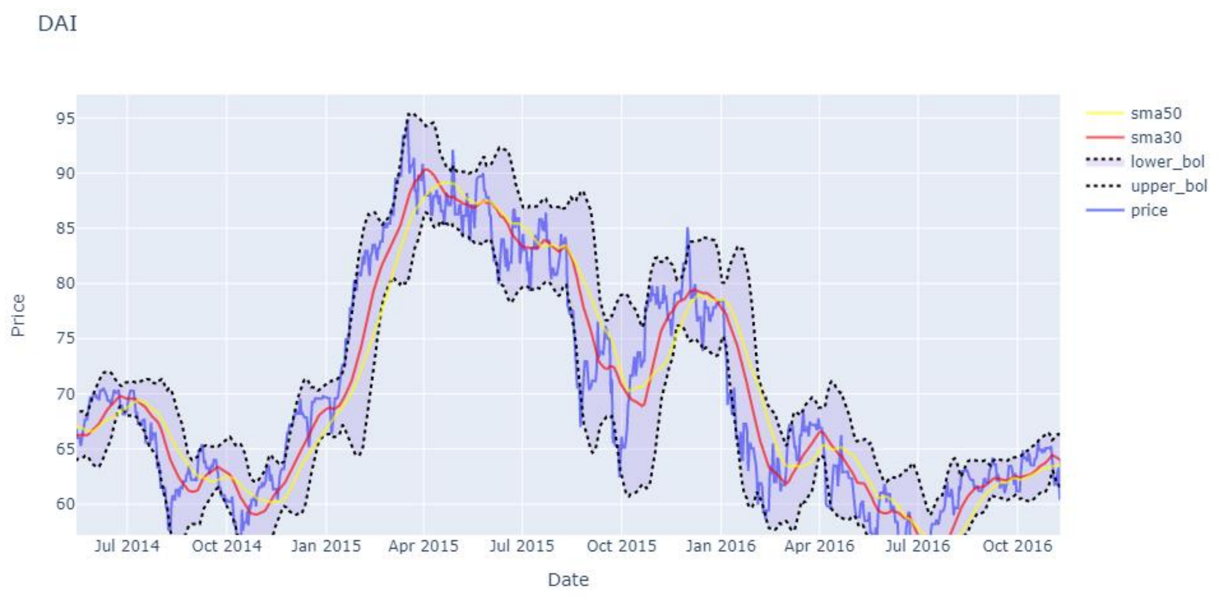


Figure 5: Stock trend of Daimler - drilled

However, we can see that the stock trend seems to be volatile over the years. To get a closer look, I created a histogram of the distributions from the daily returns.

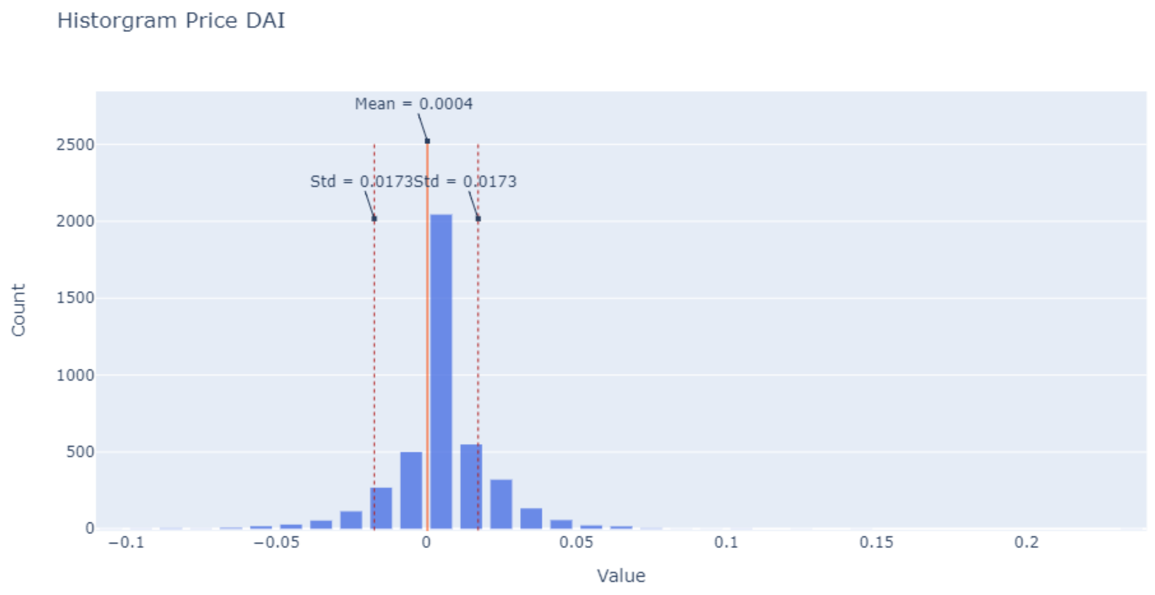


Figure 6: Distribution of daily returns

Here, we can see that the stock has mostly a slight increase. However, there are also high number of ups and downs, that explains the volatility of the stock price. So way is this from interest?
My assumption here: The more volatile a stock, the harder to predict.

I also want to show the stock trend of Nvidia. The figure shows that a long period of the dataset looks linear, but at the end, there is a high increase. Since the training data set is mostly with low volatility, it will be interesting to see how the algorithms perform with does data.

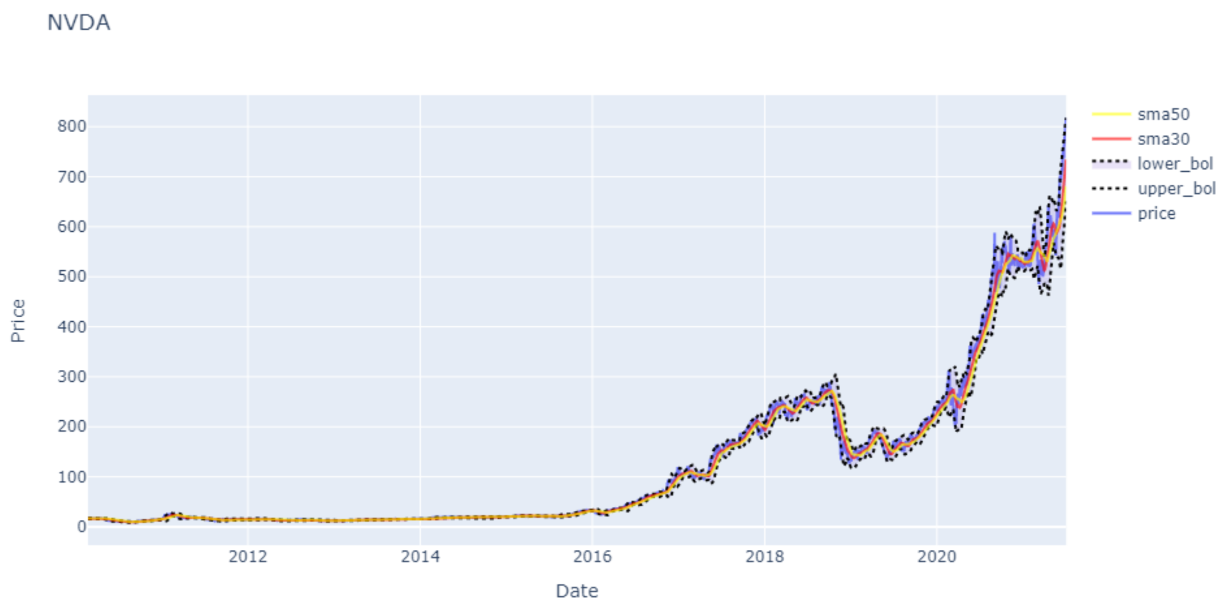


Figure 7: Price trend of Nvidia

The last figure I provide in the analyzation section is a correlation matrix. I going to use this plot to check, whether the features I select, have influence to the stock price or not.

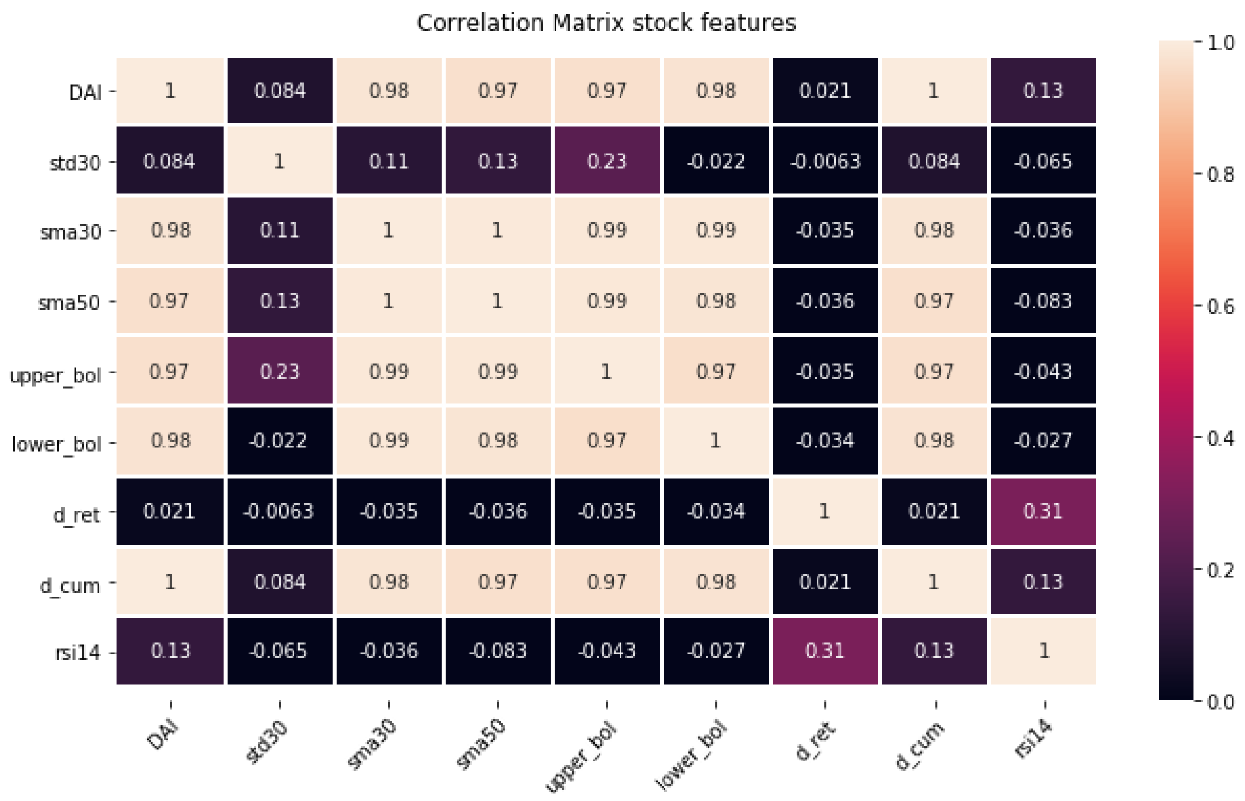
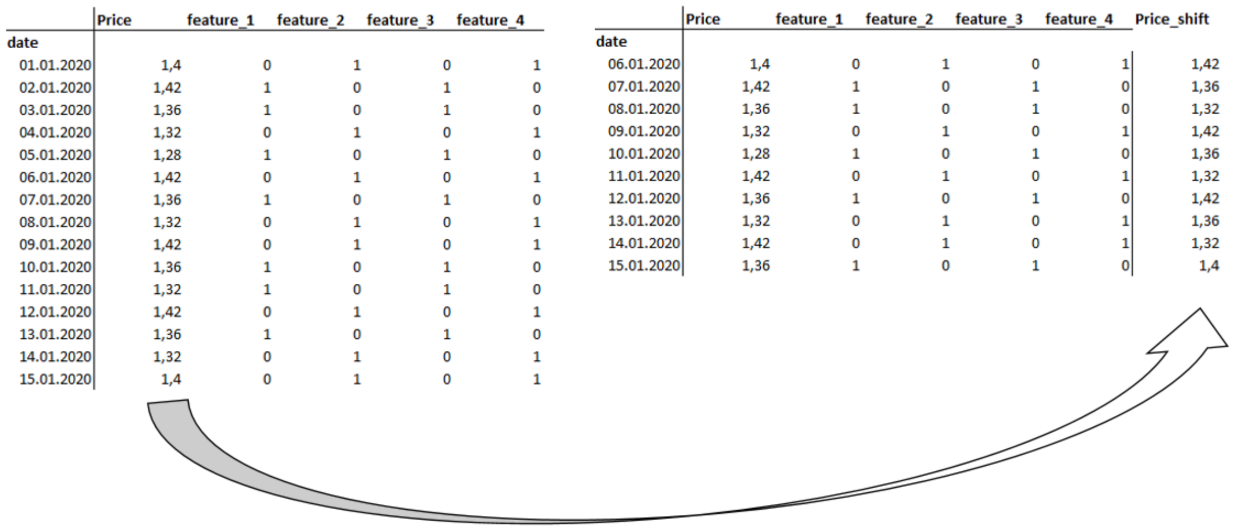


Figure 8: Correlation matrix features and price

We can denote that the most features have a very strong positive influence into the price. Only RSI and the daily return seems to have no influence at all. I could observe the same with the Nvidia stock. However, for this project, I decided to keep both features within the dataset, but for the next development iteration, a feature selection based on the correlation values would be an important feature.

Further data preparation:

Before I can start with the prediction of the stock price, the dataset needs to be further prepared. Currently I have no y_train/ y_test target where a supervised model can be trained on.
In order to implement this, I take the “price” column and shift it by the number of days, I want to forecast.
For instance, if I want to predict the price in five days, I make a copy of the price column and shift it by the number of five like shown in figure below. Afterwards I adapt the index column. Otherwise, I would have a misleading date index.
Now I’m able perform the split into the test and training dataset. In this project, the ratio between training and test 75% to 25%.



	Price	feature_1	feature_2	feature_3	feature_4
date					
01.01.2020	1,4	0	1	0	1
02.01.2020	1,42	1	0	1	0
03.01.2020	1,36	1	0	1	0
04.01.2020	1,32	0	1	0	1
05.01.2020	1,28	1	0	1	0
06.01.2020	1,42	0	1	0	1
07.01.2020	1,36	1	0	1	0
08.01.2020	1,32	0	1	0	1
09.01.2020	1,42	0	1	0	1
10.01.2020	1,36	1	0	1	0
11.01.2020	1,32	1	0	1	0
12.01.2020	1,42	0	1	0	1
13.01.2020	1,36	1	0	1	0
14.01.2020	1,32	0	1	0	1
15.01.2020	1,4	0	1	0	1

	Price	feature_1	feature_2	feature_3	feature_4	Price_shift
date						
06.01.2020	1,4	0	1	0	1	1,42
07.01.2020	1,42	1	0	1	0	1,36
08.01.2020	1,36	1	0	1	0	1,32
09.01.2020	1,32	0	1	0	1	1,42
10.01.2020	1,28	1	0	1	0	1,36
11.01.2020	1,42	0	1	0	1	1,32
12.01.2020	1,36	1	0	1	0	1,42
13.01.2020	1,32	0	1	0	1	1,36
14.01.2020	1,42	0	1	0	1	1,32
15.01.2020	1,36	1	0	1	0	1,4

Figure 9: Price shifting

It is also important to understand, that for each window prediction, the dataset must be adapted first.

Algorithm Technics:

For the experiment, I decided to use three different machine-learning (ml) algorithms, which all belongs to the class of supervised machine learning algorithms:

- Linear Regression
- Multi-layer Perceptron
- LSTM

The idea is, to create a model of each ml technique and compare the results by the numbers of the:

- coefficient of determination (R2),
- mean square error (MSE)
- mean absolute error (MAE)

In the notebook, I also added the ratio of accuracy, but since there are a lot of rounding errors, the expressiveness of this number is low.

Linear Regression:

The linear regression model is an approach that tries to explain an observed variable by the use of different independent variables. In other words, it tries to identify the relationships between the target and the dependent variables, whereby the target is the linear combination of the regression coefficients [1]. As input, I take the entire feature list.

Multi-layer Perceptron:

The Multi-layer perceptron (MLP) belongs to the class of the artificial neural network (ANN) which consists at least of three layers of nodes (input layer, hidden layer and output layer). All the given nodes except from the input layer are neurons that uses a nonlinear activation function [2].

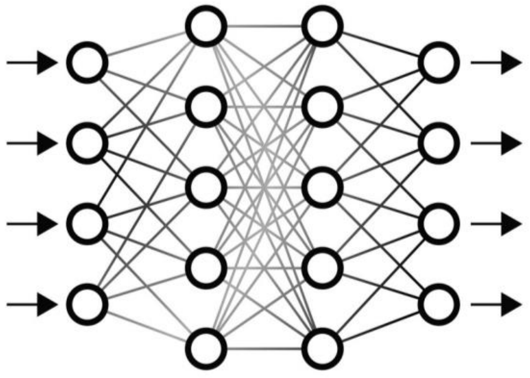


Figure 10: Neuronal Network

[1] Source: Lineare Regression, https://de.wikipedia.org/wiki/Lineare_Regression
[2] Source: Multilayer perceptron, https://en.wikipedia.org/wiki/Multilayer_perceptron

Since neuronal networks are more effective by using normalized values, I perform a normalization of the entire dataset between 0 and 1. For the MLP model, I set the maximum number of iterations to 1000 and the number of hidden layers to 100.

LSTM:
The term LSTM is the abbreviation for "long short-term memory" and belongs to the class of the artificial neural networks as well. The main difference to other neuronal networks is the memory function by using three types of gates: The input gate, remember and forget gate and the output gate [3].

Like with the MLP model, I also perform a normalization on the dataset first. Next, I set up a model with three input layers using 100 units each and one output layer. The batch size is set to 100 and the epochs (iterations) is set to 10.

I have chosen the settings for MLP and LSTM based on the results of several test runs, whereby these settings lead to the best performance yet.

[3] Source: LSTM, https://de.wikipedia.org/wiki/Long_short-term_memory

Benchmark & Results:

Now let's take a look at the results after running the last section of the notebook. For the first observation, we are looking to the Daimler stock on a seven-day forecast.

```
7 days out:
------------
Linear Regression
Linear Regression R2: 0.9599340051810165
Linear Regression MSE: 5.849530208769271
Linear Regression MAE: 1.8381742201585387
Accuracy: 0.026061776061776062
```

With the linear regression model (LR) and the MLP model, the coefficient of determination (R2) is at 0.96. The mean square error is at 5.8 and the mean absolute error is at 1.8

```
Muli-layer Perceptron
Muli-layer Perceptron R2: 0.9597540943486109
Muli-layer Perceptron MSE: 5.875796718656164
Muli-layer Perceptron MAE: 1.842234816874342
Accuracy: 0.019305019305019305
```

With LSTM, the R2 value is at 0.79. The MSE and MAE are facing an increase.

```
LSTM
33/33 [==============================] - 2s 4ms/step - loss: 0.0055 - mse: 0.0055
LSTM R2: 0.7987103971670845
LSTM MSE: 29.387754323884256
LSTM MAE: 3.660702528419642
Test loss: 0.005484593100845814
Test accuracy: 0.005484593100845814
Accuracy: 0.55%
```

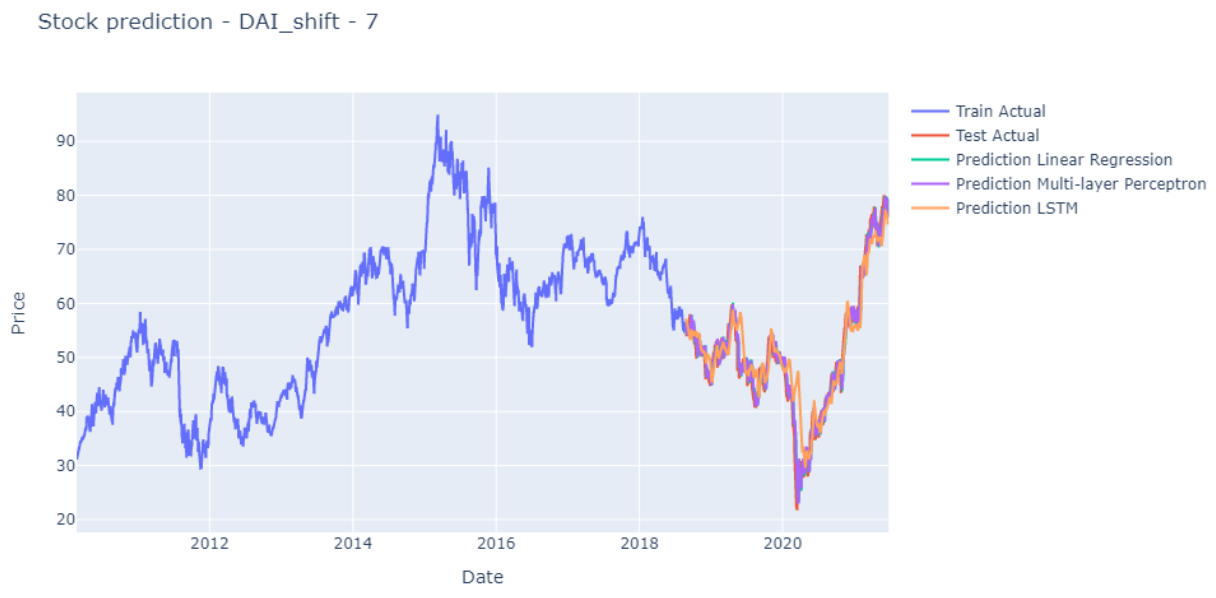


Figure 11: Prediction Daimler stock

The first look on the trend chart accredit the numbers. LR and MLP are close to the actual trend whereby LSTM has more variance.

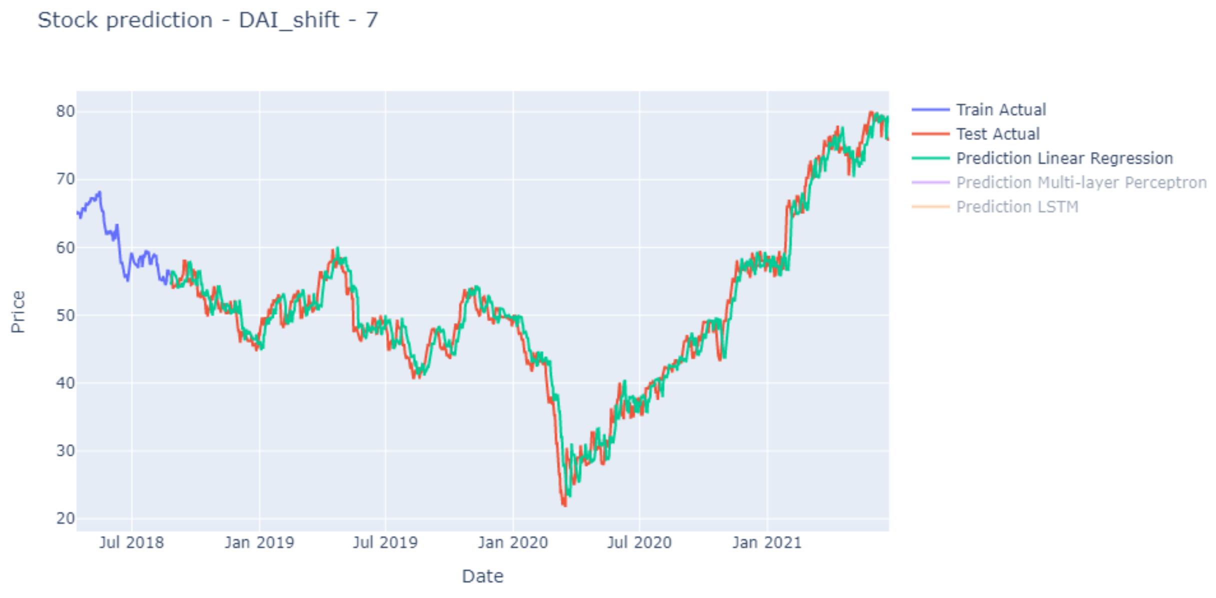


Figure 12: Prediction Daimler stock with linear regression

If I going to drill into the chart, the figure denotes, that LR is following the stock trend but has a right shift on the date column. The same is observed with MLP as well.

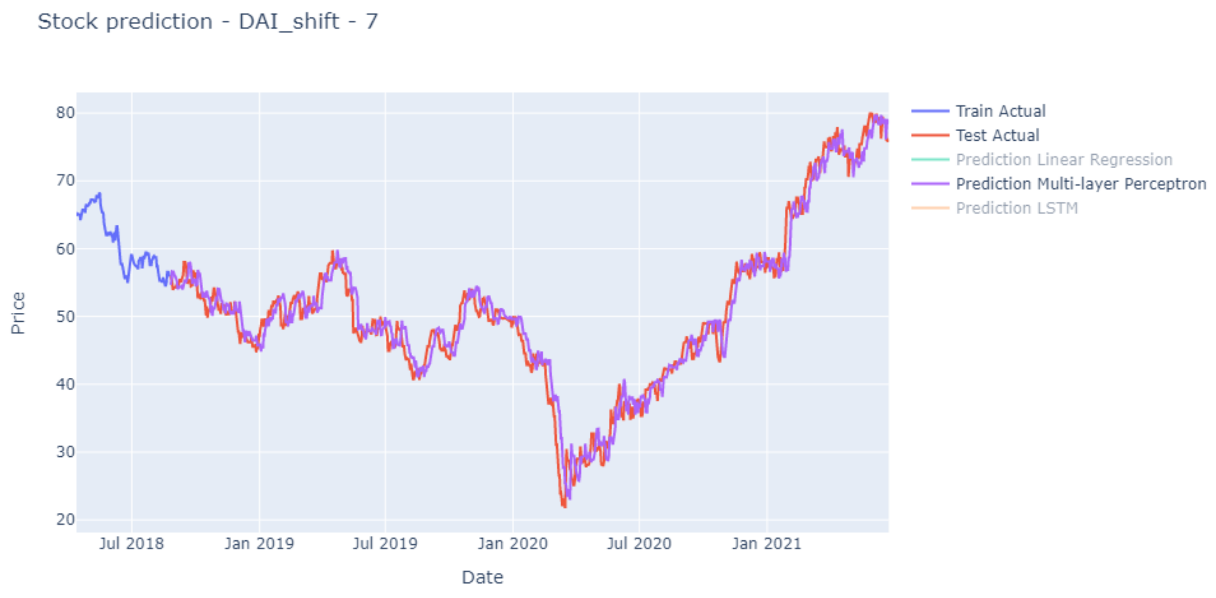


Figure 13: Prediction Daimler stock with MLP

With LSTM, there is also a shift on the x-axis but additionally, the predictions do not correctly follow the trend of the actual price.

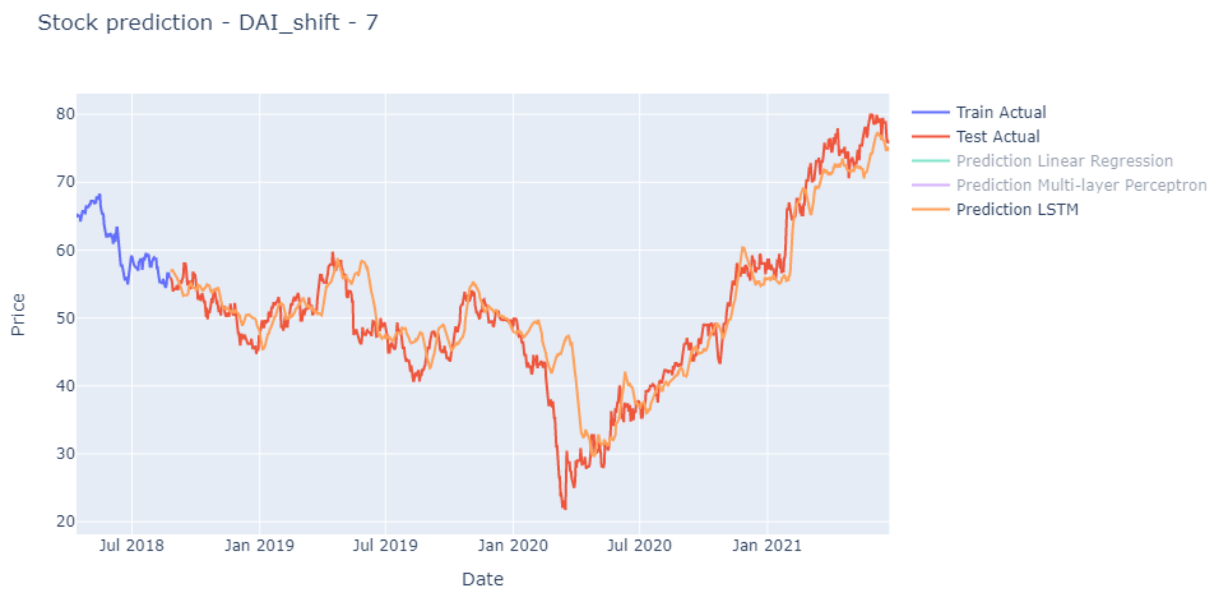


Figure 14: Prediction Daimler stock with LSTM

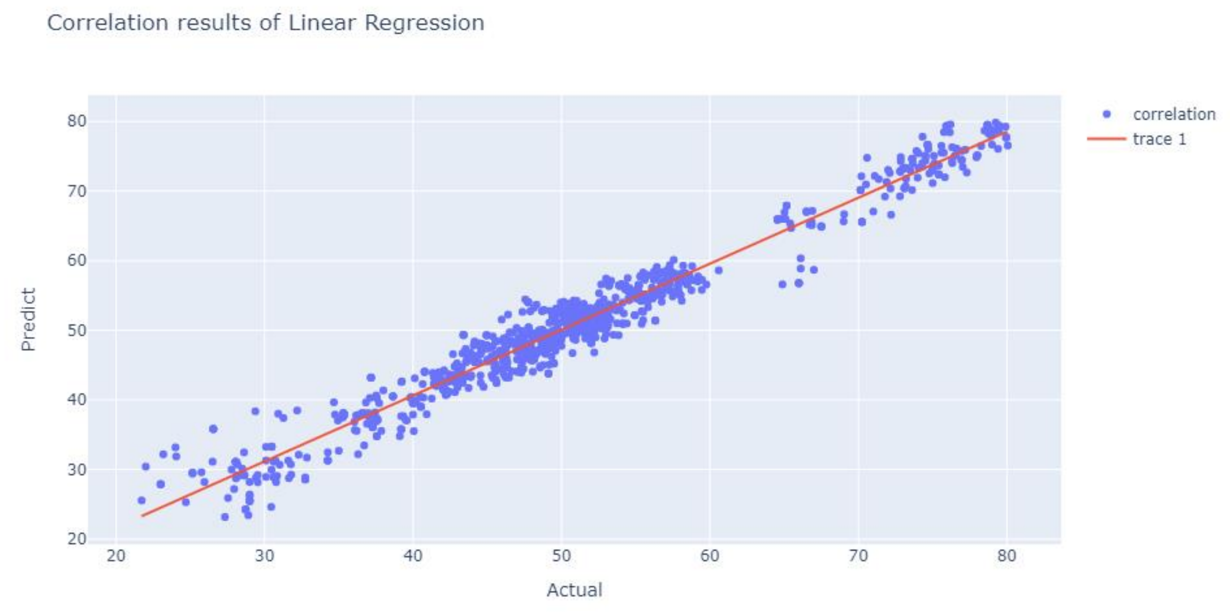


Figure 15: Scatter plot linear regression

Let's also take a look at the correlation scatter plot of LR and LSTM. The predicted/ actual dots of LR are closer to the line as with LSTM, which are sometimes widely spread. However, even with LR, some dots are scattered at the tails of the line, which probably indicates more variance because of the volatility of the stock.

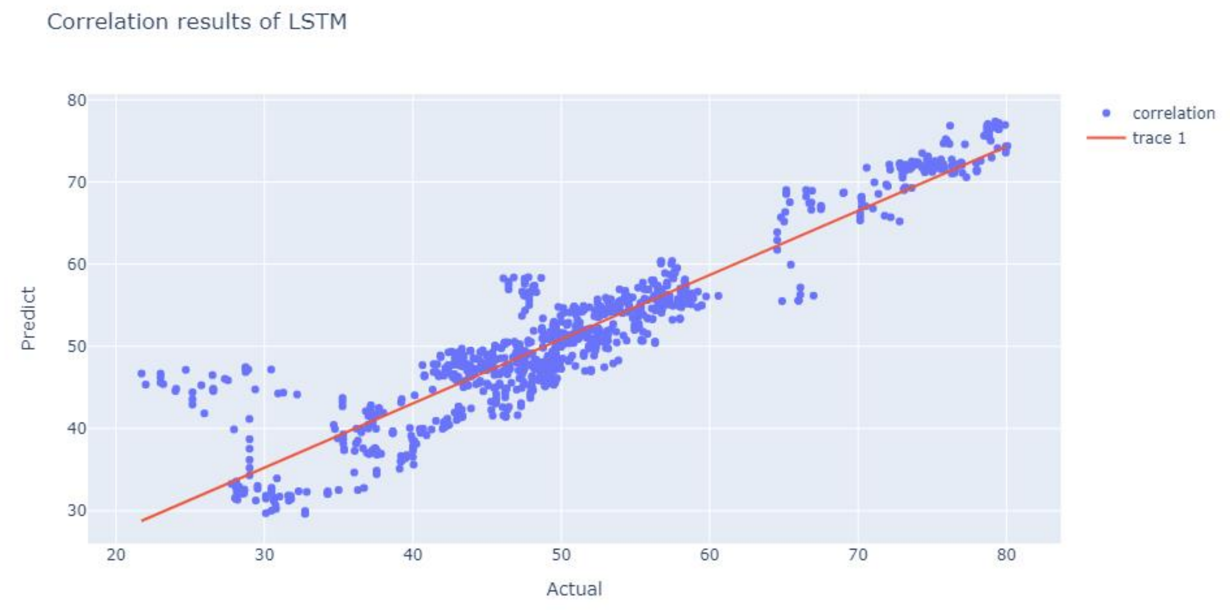


Figure 16: Scatter plot LSTM

Lets take a look at the following results with higher number of days to predict.
As expected, with all ML algorithms the accuracy drops by the increasing number of days.

```
14 days out:
------------
Linear Regression
Linear Regression R2: 0.9210534027157851
Linear Regression MSE: 11.205277229153344
Linear Regression MAE: 2.480697728061077
Accuracy: 0.02131782945736434


Muli-layer Perceptron
Muli-layer Perceptron R2: 0.9208279486779142
Muli-layer Perceptron MSE: 11.237277025011299
Muli-layer Perceptron MAE: 2.4830014230203714
Accuracy: 0.025193798449612403


LSTM
33/33 [==============================] - 1s 4ms/step - loss: 0.0055 - mse: 0.0055
LSTM R2: 0.7936873469632911
LSTM MSE: 29.282965354868814
LSTM MAE: 3.773492747831714
Test loss: 0.005465036258101463
Test accuracy: 0.005465036258101463
Accuracy: 0.55%
```

However, even with a number of 30 days, the R2 is still at 0.8 with LR and MLP, whereby LSTM drops to 0.69.

```
30 days out:
------------
Linear Regression
Linear Regression R2: 0.8023495414619239
Linear Regression MSE: 26.06617012652934
Linear Regression MAE: 3.840367674747195
Accuracy: 0.014634146341463415


Muli-layer Perceptron
Muli-layer Perceptron R2: 0.8056262562367411
Muli-layer Perceptron MSE: 25.634036523560532
Muli-layer Perceptron MAE: 3.8282281762928236
Accuracy: 0.007804878048780488


LSTM
33/33 [==============================] - 1s 4ms/step - loss: 0.0075 - mse: 0.0075
LSTM R2: 0.6966687847866634
LSTM MSE: 40.003363103326905
LSTM MAE: 4.687066224335461
Test loss: 0.007465765345841646
Test accuracy: 0.007465765345841646
Accuracy: 0.75%
```

Now, I want to take a quick look of the results of the Nvidia prediction. From our previous observation of the trend data, we know, that Nvidia is not that volatile than the Daimler stock.

```
7 days out:
------------
Linear Regression
Linear Regression R2: 0.9840980476812651
Linear Regression MSE: 477.8649860102321
Linear Regression MAE: 15.751104514724204
Accuracy: 0.0019305019305019305


Muli-layer Perceptron
Muli-layer Perceptron R2: 0.982952112618063
Muli-layer Perceptron MSE: 512.3011503232476
Muli-layer Perceptron MAE: 16.427553295187025
Accuracy: 0.0019305019305019305


LSTM
33/33 [==============================] - 1s 4ms/step - loss: 0.0030 - mse: 0.0030
LSTM R2: 0.9357662233186765
LSTM MSE: 1930.2707101593764
LSTM MAE: 33.1983335169494
Test loss: 0.002950117690488696
Test accuracy: 0.002950117690488696
Accuracy: 0.30%
```

It seems like that the lower volatility is also reflected by the numbers of R2, which are 0.98 in LR and MLP. Even with LSTM the R2 is at 0.92 for a seven days prediction. With the forecast up to 30 days, there is a decrease of ~0.06 for all ML model.
To most difference compared to the first observation with Daimler is, that in all results, MSE and MAE are much higher.

```
30 days out:

------------

Linear Regression

Linear Regression R2: 0.9260988131636365

Linear Regression MSE: 1967.4077523867877

Linear Regression MAE: 32.34013531018811

Accuracy: 0.002926829268292683


Muli-layer Perceptron

Muli-layer Perceptron R2: 0.9314364892804224

Muli-layer Perceptron MSE: 1825.3073907897913

Muli-layer Perceptron MAE: 31.672436833157796

Accuracy: 0.001951219512195122
```

```
LSTM
33/33 [==============================] - 1s 4ms/step - loss: 0.0072 - mse: 0.0072
LSTM R2: 0.8687900250209328
LSTM MSE: 3493.090341510891
LSTM MAE: 44.56669153963414
Test loss: 0.007249835878610611
Test accuracy: 0.007249835878610611
Accuracy: 0.72%
```

By taking a look onto the next two charts, we can see a high increase of the stock price within the test dataset.

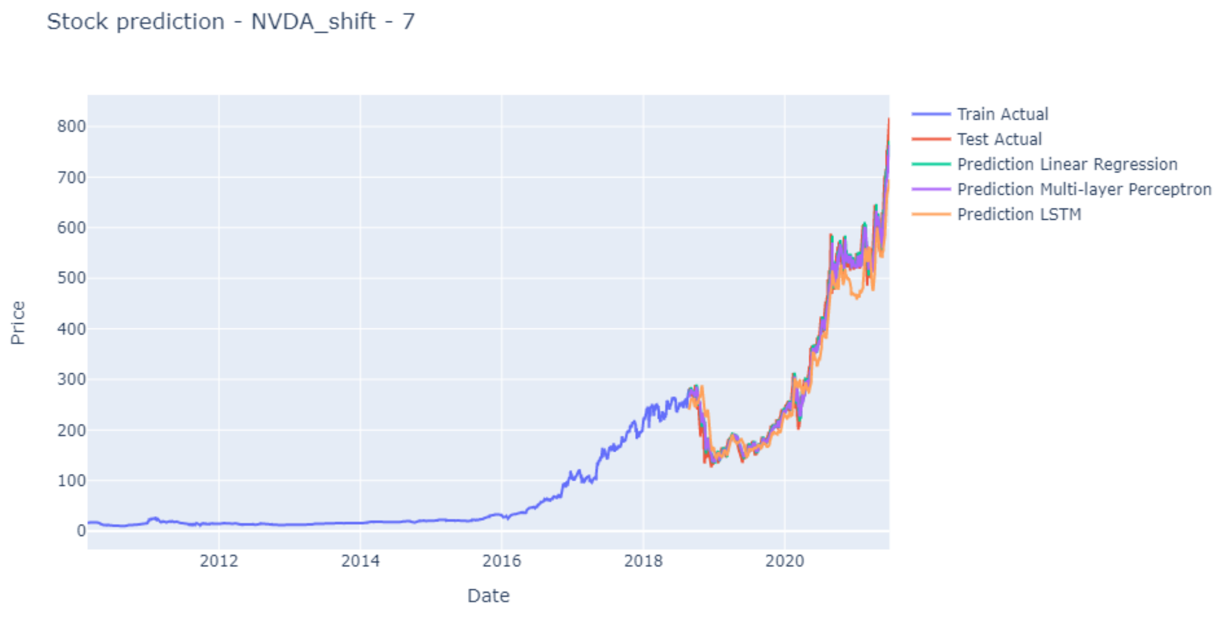


Figure 17: Nvidia prediction - total

This development could explain why the errors have increased that much.

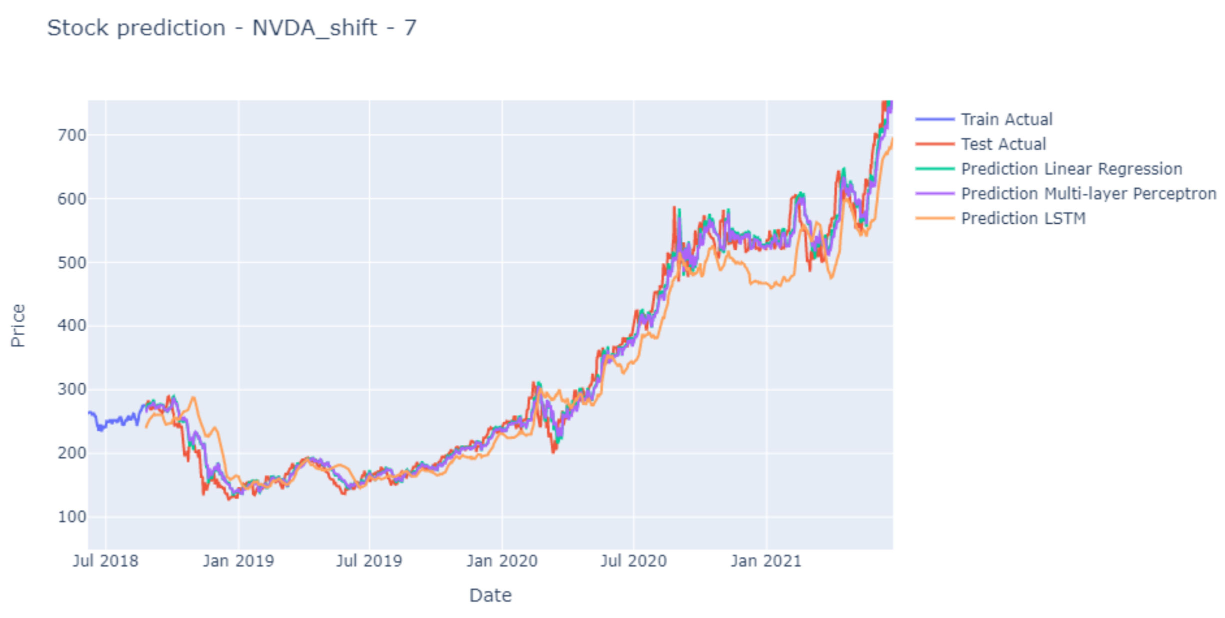


Figure 18: Nvidia prediction - drilled

Reflection:

With this project, I have conducted predictions of two different stocks by the use of three different AI models over certain time windows. During the observation of the results, it seems like that with LR and MLP the results were always better than with LSTM.
By using two different datasets, I can also make the conclusion that the results are also dependent on the volatility of the stock trend. In order to predict a stock price, I have only used the price trend of past days and some statistical number. For that, all results looks well.

However, if we take a very close look to the predicted data for a seven days prediction, we can see that there is a trend shift between actual and the predicted data for exactly seven days. I explain this behavior by the given dataset the algorithms are training with.
It seems like that all algorithms are following the actual stock price given the dataset.

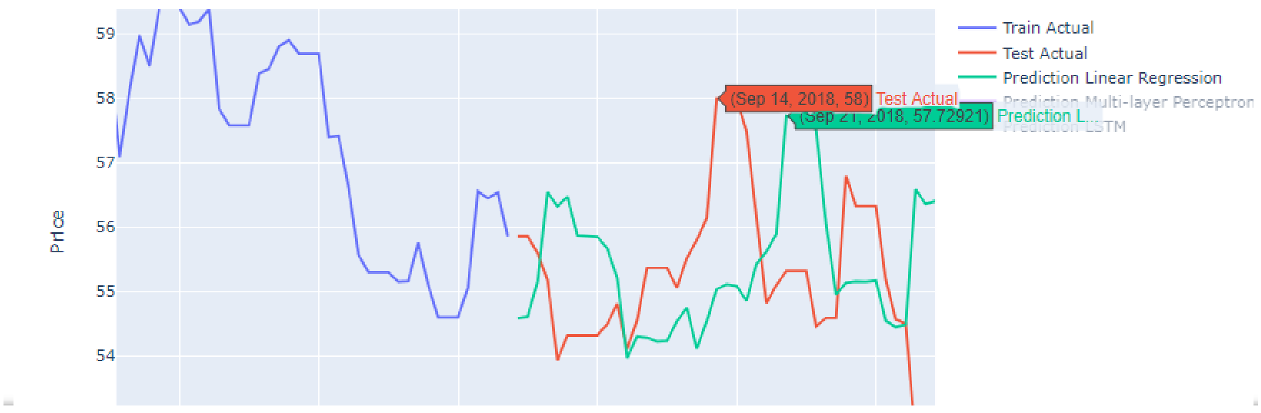


Figure 19: Actual Price and predicted price

In other words, the results I got, should not be used to make some decisions, if it comes to stock trading. For that, an additional set of features is mandatory. Also switching form a supervised approach to a deep learning ml algorithm could bring some better results. For now, the predictions are too unspecific.

Improvement:

Although the performance of each model has seem to be quite impressive, I have yet not faced a real prediction yet. Thus, a new set of features is necessary. The yfinance API could help here a lot since it provides a lot more of market data/ metadata and information about the companies that can be used to define new features. By using neuronal networks like MLP and LSTM, I see a lot of potential to improve the results by changing the model parameters.

During my creation of the LSTM model, I could detect a dramatic change of the performance while playing with the number of units or the batch size. Thus, a grid search or another randomized approach could help here a lot.

I hope I could give a good contribution with my project and would be happy to get some feedback.

Thanks a lot!