Stock Price Indicator

Capstone project for the Udacity Data Science Nanodegree

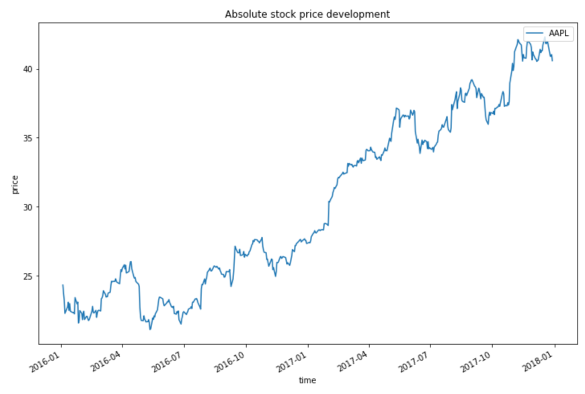
Jochen Ruland, May 2021

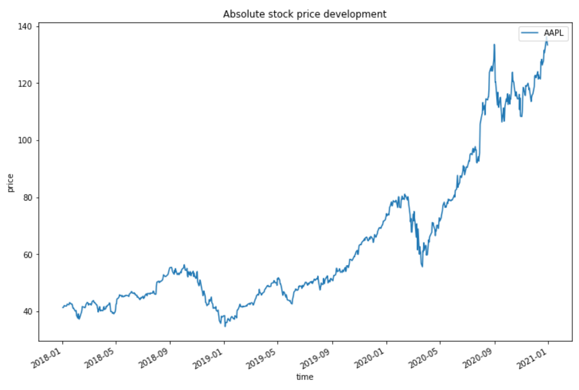
Problem Introduction

My university professor always said: “Prediction means to know where to scratch yourself before feeling itchy.” That was around 2000 when there was by far less data available nor was there the software or computing power of today.

With this project I want to prove him wrong…ok, maybe at least prove him partially wrong when it comes to the prediction of future stock price development. So, the question is: are stock prices completely random and therefore unpredictable or are there patterns which can be used to predict future prices with a certain level of confidence.

That this will not be an easy task is shown by comparing the stock price development of Apple for different periods in time.





While looking at the first graph you might think that it could somehow be possible to draw a linear line, this is not possible in the second graph. And it will become even more difficult when looking at months, weeks or single days.

In this project we will use machine learning, specifically regression learners, to predict future stock prices.

Strategy to solve the problem

To setup a model for predicting future stock prices we must identify and gather the relevant data. The data must be checked for anomalies and eventually it must be cleaned.

To fit a model, we need one or more independent variables describing the dependent variable (stock price). If we cannot gather these features directly from the data source, we should evaluate if they can be calculated based on the data available.

When it comes to modeling, we will use regression learners as we want to make precise predictions on price. Other approaches would be to use classifiers in order to make predictions whether a stock will go up or down or reinforcement learners if we wanted to determine if it is best to buy, to sell or to do nothing. Once a model is setup and fitted, we must evaluate the prediction results. Therefore, appropriate metrics must be defined and analyzed.

By starting two Python scripts the data will be downloaded, processed and predictions will be made.

For non-technical users, there are different possibilities to publish the results. In this case, a WebApp has been implemented.

Metrics

To measure the quality of predicted stock prices the following metrics have been applied:

Mean Absolute Error (MAE): this metric gives us an overview how far predictions deviate from the actual values.

Mean Absolute Percentage Error (MAPE): this metric gives us an overview how far predictions deviate from the actual values and shows the result as relative values.

Pearson Correlation (CorrCoef): gives an indication how tight the predicted values are along the actual values.

R2 Score: it shows the relevance of our features by indicating the proportion of the variance in the dependent variable that is predictable from the independent variable(s). So, it also shows correlation.

In a prior step also Mean Squared Error (MSE) and Root Mean Squared error (RMSE) have been calculated but as RMSE comes to relatively similar results as MAE and MSE always returns quite large deviations, these metrics have been skipped.

Analysis

Exploratory Data Analysis

To read in historical stock price information we use the ALPHA VANTAGE API. When starting the project an API called “yfinance” has been used first and it has worked quite well. But after reading the details about this API it has become clear, that it is not officially supported by Yahoo. It scrapes data from the Yahoo website. As scraping data from the website is still legally a grey zone and the data consistency can be questioned, we have switched to the official ALPHA VANTAGE API. The service is for free unless you do not start more than 5 requests per minute. You just need to register to get an API Key (cf. https://www.alphavantage.co/support/#api-key).

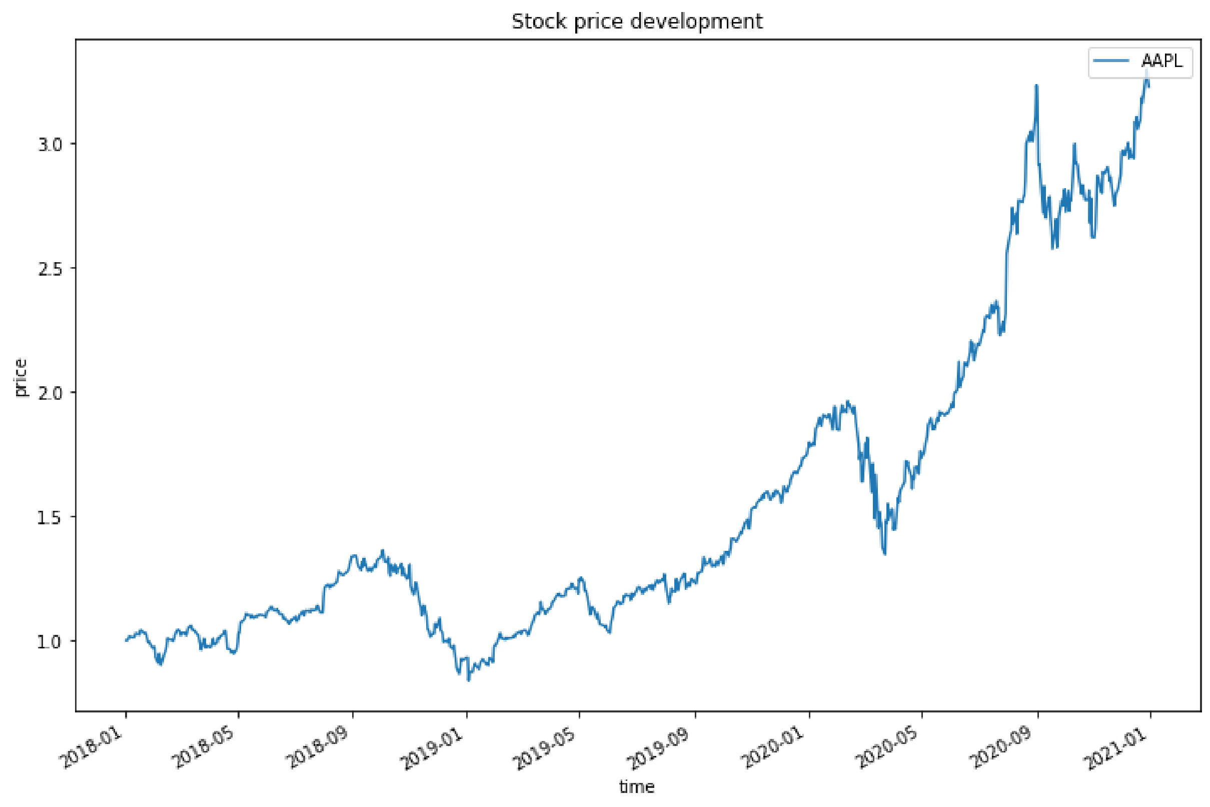
ALPHA VANTAGE provides daily financial data as well as for weeks, months, or inter-day periods etc. And it provides a wide range of indicators. For this project we want to predict daily stock price development therefore the software selects the adjusted closing price daily. Data can be downloaded in a Pandas format.

	1. open	2. high	3. low	4. close	5. adjusted close	6. volume	7. dividend amount	8. split coefficient
date								
26.05.2021	126.955	1.273.900	126.420	126.85	126.850.000	56575920.0	0.0	1.0
25.05.2021	127.820	1.283.200	126.320	126.90	126.900.000	72009482.0	0.0	1.0
24.05.2021	126.010	1.279.400	125.940	127.10	127.100.000	63092945.0	0.0	1.0
21.05.2021	127.820	1.280.000	125.210	125.43	125.430.000	79295436.0	0.0	1.0
20.05.2021	125.230	1.277.200	125.100	127.31	127.310.000	76857123.0	0.0	1.0
...	...	...	...	...	...	...	...	...
08.01.2021	132.430	1.326.300	130.230	132.05	131.629.957	105158245.0	0.0	1.0
07.01.2021	128.360	1.316.300	127.860	130.92	130.503.552	109578157.0	0.0	1.0
06.01.2021	127.720	1.310.499	126.382	126.60	126.197.293	155087970.0	0.0	1.0
05.01.2021	128.890	1.317.400	128.430	131.01	130.593.265	97664898.0	0.0	1.0
04.01.2021	133.520	1.336.116	126.760	129.41	128.998.355	143301887.0	0.0	1.0

Example of an Alpha-vantage data download in Pandas format for Apple (AAPL).

We only select the “date” column and the “adjusted close” column, sort the data by ascending dates and save it to a .csv file for further use.

In a next step the price data is normalized to not only show absolute stock prices but also relative stock price development. This is not such relevant if you only show the development of one stock, but it helps a lot if you want to compare different stocks and visualize the development. The information then has been plotted as shown below and checked for inconsistencies.



Based on the adjusted close price a bunch of features have been calculated which can be used as independent variables for training and prediction. The application calculates the following features:

Simple moving average: it provides an indication of the average rolling mean over a 10-days period. This also means that the first datapoint can be calculated after a period of 10 days. Therefore, there is no previous data available, and those observations will be skipped.

Bollinger bands ®: it provides a band 2 standard deviations above and below the mean which can help to identify significant price deviations.

Momentum: it indicates a price change over some days, in this case the change over 5 days. Thus, momentum could be an indicator for short term tendency of price development.

Daily returns: they describe the relative change in price per day.

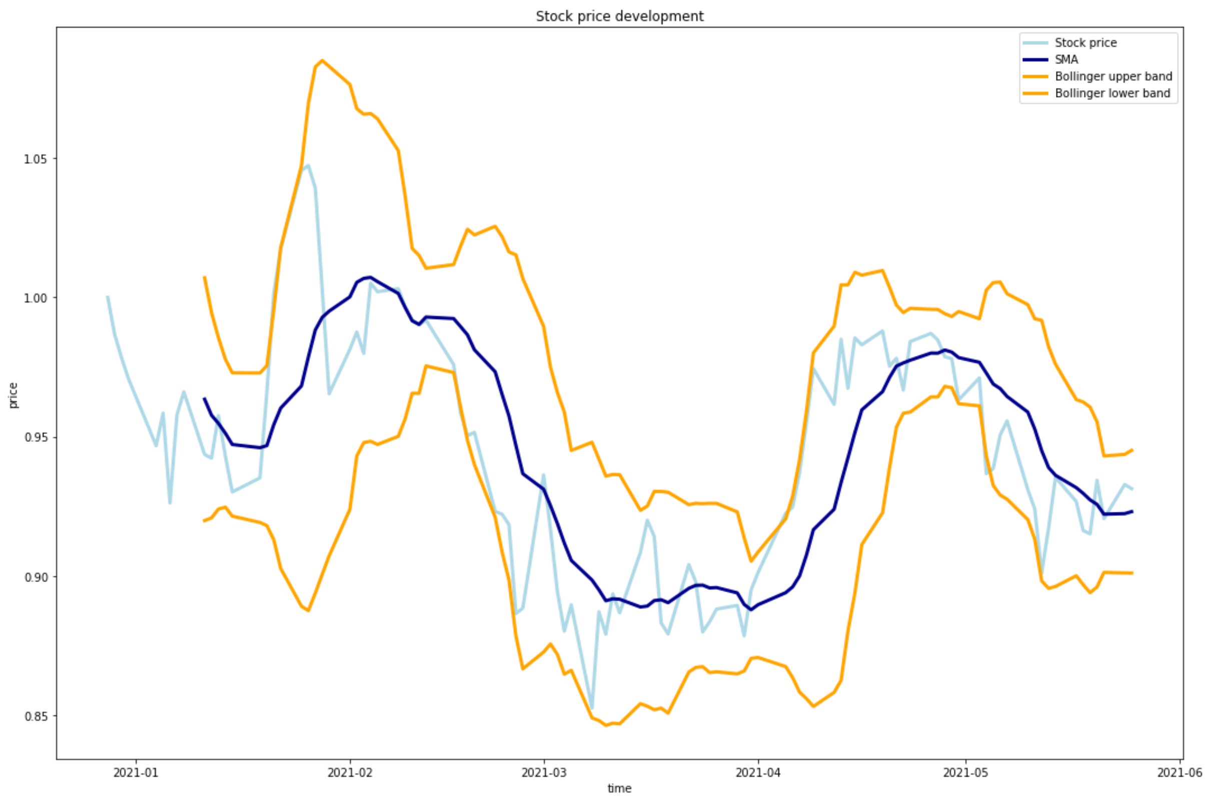
Cumulative returns: they indicate the relative change in price from the first day of the period selected until the last day of that period.

Further a **market index** which best represents the overall market development has been selected.

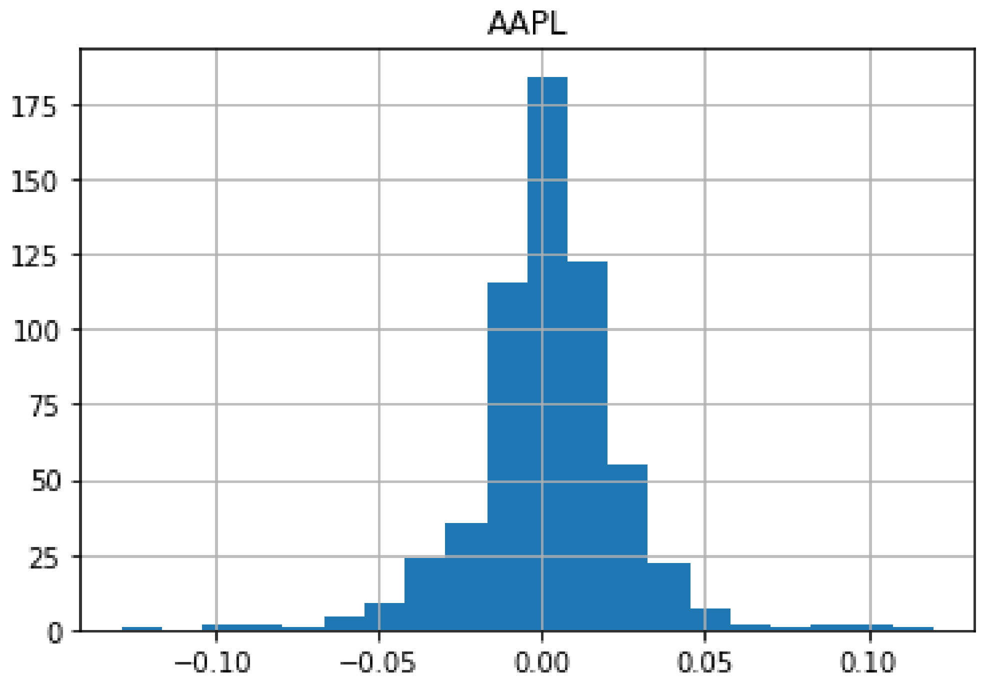
The calculation of the different features will be further discussed in chapter "Data processing". If there is missing data, this problem has been addressed in a later step when merging all indicators together by dropping null-values and using backward and forward fill methods.

Data Visualization

The development of the simple moving average and Bollinger bands for Apple (AAPL) is shown in the following plot.



Plotting the distribution of daily returns shows that they are quiet normally distributed.



Methodology

Data Processing

Data import an normalizing

When downloading the data for one stock ticker symbol from ALPHA VANTAGE there are two options available. The default option is "compact" which means that Alpha-Vantage returns the last 100 entries. So, for example the last 100 trading days. Using the parameter "outputsize=full" will return the data of the last 20 years. We download the full dataset, select only the date and the adjusted close value, sort the data by ascending dates and save the data to an .csv file with the name of the stock symbol.

To match the data with the selected timeframe for analysis and training, we create a new dataframe containing only the dates of the selected timeframe as index. Then we read the .csv file and we merge the dataframe with the corresponding information from the .csv file.

In the next step we normalize the stock price data by dividing each value by the first value which serves as reference. Thus, we can show the relative price development. As already mentioned above this is more useful when comparing different stocks.

Calculation of indicators

As the indicators all have been derived from the adjusted close this somehow also represents a step of data processing. The indicators have been calculated as follows:

Simple moving average: *SMA[t] = price[t] / price[t-n : t].mean() - 1*

Bollinger Bands ®: *BB[t] = (price[t] – SMA[t]) / 2 std()[t]*

Momentum: *momentum[t] = price[t] / price[t-n] – 1*

Daily returns: *daily_returns[t] = (price[t] / price[t-1]) - 1*

Cumulative returns: *cum_returns[t] = (price[t] / price[0]) - 1*

In this case we have chosen S&P 500 ("SPY") as **market index**. The market index data will be downloaded and imported in the same way the stock data has been imported just specifying the index ticker symbol.

After calculating the indicators and selecting a relevant market index these information have all been merged into one dataframe called "indicator_df". To create this dataframe we add a column called "symbol" holding the stock ticker symbol and further columns for the adjusted closing price, each indicator, and the market index. Then we drop null-values and fill in missing values using backward and forward fill methods. Now we have got the dependent variable (daily stock price) and all describing features together in one dataframe.

	Date	Symbol	Adj Close	Daily Returns	Cumulative Returns	SMA	Momentum	Upper Band	Lower Band	Market Index
0	02.01.2019	AAPL	38.439.735	0.000000	0.000000	0.955876	0.000000	1.009.707	0.902045	240.242.630
1	03.01.2019	AAPL	34.610.851	-0.099607	-0.099607	0.955876	0.000000	1.009.707	0.902045	234.509.766
2	04.01.2019	AAPL	36.088.364	0.042689	-0.061170	0.955876	0.000000	1.009.707	0.902045	242.364.868
3	07.01.2019	AAPL	36.008.041	-0.002226	-0.063260	0.955876	0.000000	1.009.707	0.902045	244.275.818

Example of "indicator_df"

In order to train a model, we need to generate the labels. Therefore, we split the indicator dataframe into a 1-dimensional array of Y labels and a n-dimensional array X. X contains the training features. Now it comes to selecting the right dates. The software provides two ways of predicting future prices:

a) To make the user interface of the webapp a little easier, always 6 days after the end date of the selected period will be predicted.
b) When running the scripts, you can predict a period by entering a start date and an end date for prediction.

In both cases we receive n days to be predicted. To generate the training and test datasets we take the whole X-array of features except those for the last n days. For the Y array we take the complete dataset except the first n days. Thus, we attribute to each value of Y a set of features n days before. In a next step we split the X and Y datasets each into a set of training and test data. We split up the first 80% of the datasets as training data and the last 20% as test data. It this case it is important to not randomly split up the data into train and test data as that could mean that we would try to train the model with data from a future date to predict a stock price in the past relative to that date. We further take the date column from the Y data as this allows us to later plot the data on that time scale. In a last step we standardize the X feature set. As the values of X have different dimensions, this is important to have each value being evaluated with the same weight in the model. Standardization scales each value of the X feature set by subtracting the mean and dividing by the standard deviation to shift the distribution. Thus, you have a mean of zero and a standard deviation of one.

Once the data is split up, we can fit different kind of models and evaluate their performance in predicting future stock prices.

The following models have been implemented in the Jupyter Notebook, fitted and tested against one another.

Lasso LARS Linear Regression:

Lassso linear regression with least angle regression (LARS) is an algorithm that uses a special way to change the loss function during training to include additional costs for a model that has large coefficients and is therefore instable.

"Linear regression refers to a model that assumes a linear relationship between input variables and the target variable.
With a single input variable, this relationship is a line, and with higher dimensions, this relationship can be thought of as a hyperplane that connects the input variables to the target variable. The coefficients of the model are found via an optimization process that seeks to minimize the sum squared error between the predictions (yhat) and the expected target values (y).

- loss = sum i=0 to n (y_i – yhat_i)^2

A problem with linear regression is that estimated coefficients of the model can become large, making the model sensitive to inputs and possibly unstable. This is particularly true for problems with few observations (samples) or more input predictors (*p*) than variables than samples (*n*) (so-called *p >> n problems*).
One approach to address the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients. Linear regression models that use these modified loss functions during training are referred to collectively as penalized linear regression.

A popular penalty is to penalize a model based on the sum of the absolute coefficient values. This is called the L1 penalty. An L1 penalty minimizes the size of all coefficients and allows some coefficients to be minimized to the value zero, which removes the predictor from the model.

- l1_penalty = sum j=0 to p abs(beta_j)

An L1 penalty minimizes the size of all coefficients and allows any coefficient to go to the value of zero, effectively removing input features from the model. This acts as a type of automatic feature selection method.

… a consequence of penalizing the absolute values is that some parameters are actually set to 0 for some value of lambda. Thus, the lasso yields models that simultaneously use regularization to improve the model and to conduct feature selection. — Page 125, Applied Predictive Modeling, 2013.

This penalty can be added to the cost function for linear regression and is referred to as Least Absolute Shrinkage And Selection Operator (LASSO), or more commonly, "*Lasso*" (with title case) for short.
The Lasso trains the model using a least-squares loss training procedure.

Least Angle Regression, LAR or LARS for short, is an alternative approach to solving the optimization problem of fitting the penalized model. Technically, LARS is a forward stepwise version of feature selection for regression that can be adapted for the Lasso model.
Unlike the Lasso, it does not require a hyperparameter that controls the weighting of the penalty in the loss function. Instead, the weighting is discovered automatically by LARS.

… least angle regression (LARS), is a broad framework that encompasses the lasso and similar models. The LARS model can be used to fit lasso models more efficiently, especially in high-dimensional problems. — Page 126, Applied Predictive Modeling, 2013. “ (https://machinelearningmastery.com/lars-regression-with-python, 23.05.2021).

K Nearest Neighbor (KNN):

K nearest neighbor is an instant based approach where we use the data to identify the k nearest datapoints to our query to predict Y for the requested value X. We then use the mean to these values (cf. Udacity, machine learning for trading, 03-03 Regression, 2021). The basic regression model uses uniform weights. That means that each point in the local neighborhood will contribute in the same way to the classification of a query point.

Adaboost (meta-estimator) with KNN as base estimator:

AdaBoost is a boosting algorithm which scikit-learn describes as follows: "The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights w1, w2, …, wN to each of the training samples." (https://scikit-learn.org/stable/modules/ensemble.html#adaboost, 1.11.3. AdaBoost, 23.05.2021)

Random Forest (decision trees):

Random forest is also a meta-estimator which scikit-learn describes as follows: "A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size max_features.

The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant, hence yielding an overall better model.

In contrast to the original publication, the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class." (https://scikit-learn.org/stable/modules/ensemble.html#forest, 1.11.2.1. Random Forest, 23.05.2021)

Testing the different algorithms in the Jupyter notebook, the results using Lasso LARS linear regression and Random Forest have always been close to each other. As Lasso LARS linear regression tends to overfit easier, especially when selecting short timeframes, Random Forest has finally been implemented in the WebApp and the Python script (cf. "Implementation" and "Results").

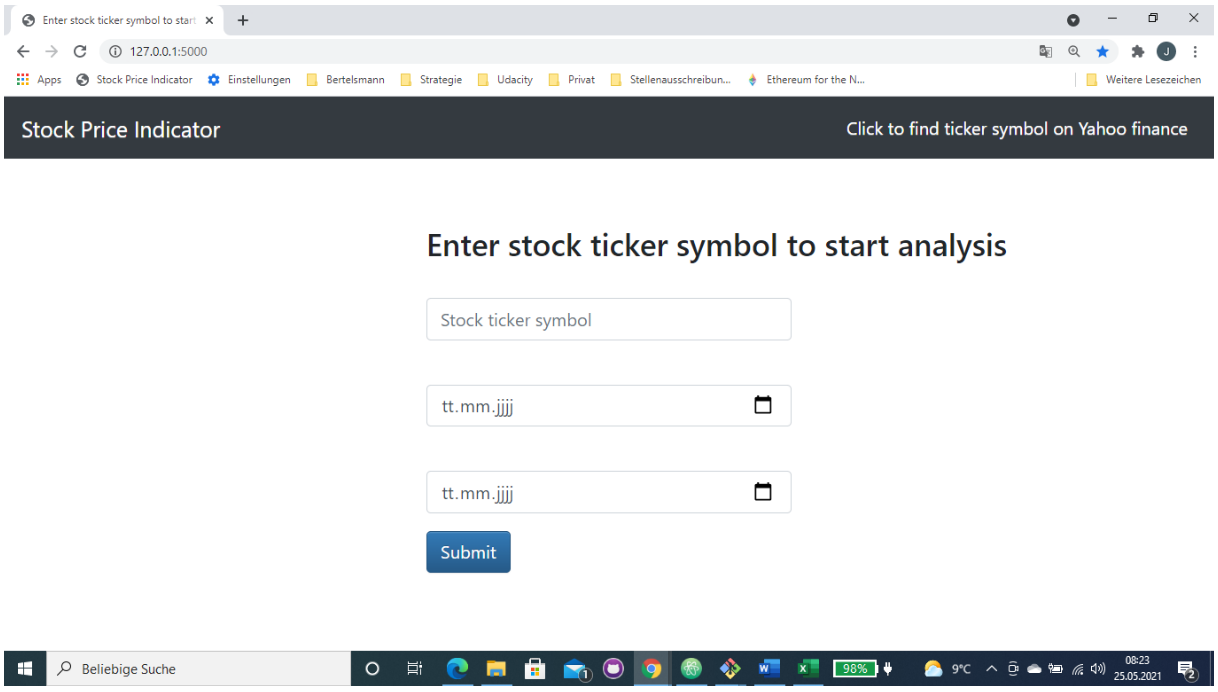
Implementation

Starting the project, the different steps described in "Data processing" have been implemented in a Jupyter Notebook named `Stock_price_indicator.ipynb`. Mainly each step has been implemented in a separate function in order to easily test the results. Once the whole process of data processing, modeling, fitting, predicting and evaluating has been implemented and tested, everything has been transferred into two main classes and some helper functions. The implementation in the two classes called “StockDataAnalysis” and “ModelStockPrice” can also be found in the Jupyter Notebook where they have been tested before being copied into two Python modules called “process_data.py” and “train_classifier.py”. StockDataAnalysis contains all steps from data download until generation of the joint dataframe “indicator_df which contains the stock price per date and all indicators described above. The second class “ModeStockPrice” instantiates a model and contains functions to create the train and test datasets, fits the model, predicts, and evaluates the results. The reason for organizing the code in these two classes is to allow different applications independently to access this code.

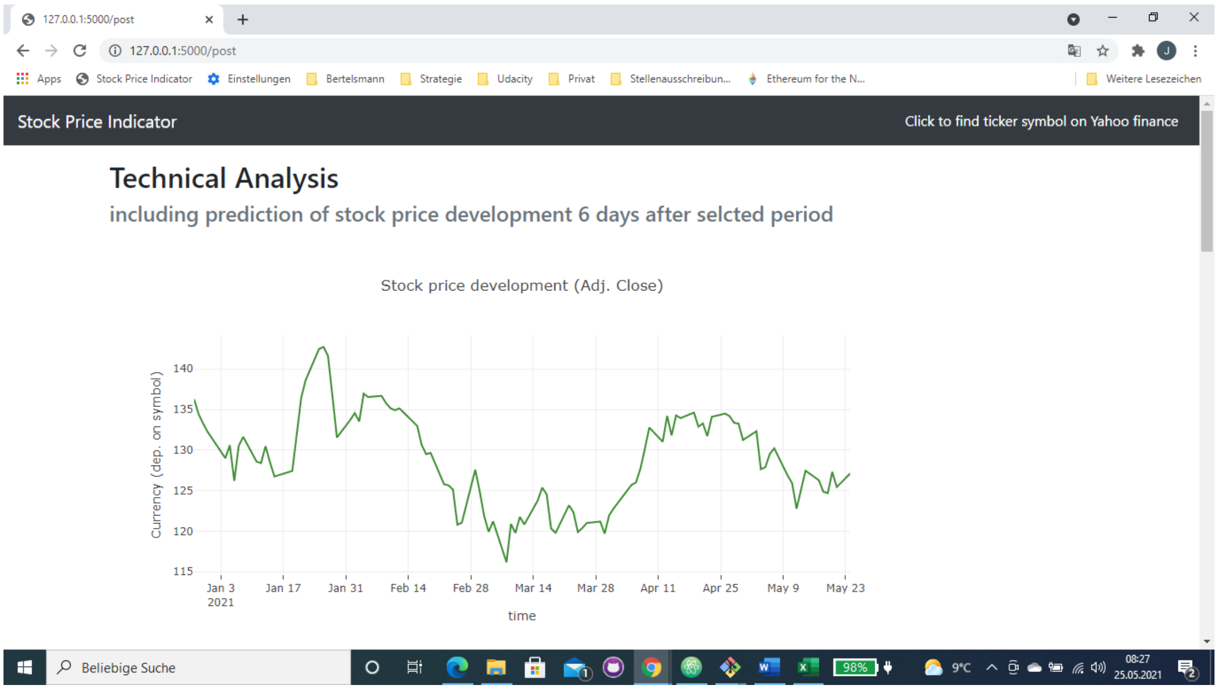
The program can be run by either starting a WebApp called “web_app.py” or two python scripts called “ETL.py” and “ML.py”. The first script downloads the data for a list of ticker symbols in a given timeframe, creates all indicators and saves the result into a database file. The second script also takes a list of ticker symbols which must be a subset of the data selected with the first script. It further takes a timeframe for prediction and then trains a model, predicts the price for the given timeframe and evaluates the model performance. It took some time to find out that the best way to organize the process is by having two scripts and to save the result of the first script into a database file.

The WebApp uses the Flask library to setup the backend and Bootstrap to create the frontend. It is a little simpler than the Python scripts and takes only one ticker symbol per request as well as the timeframe for downloading the data and training the model. The WebApp will then always predict the next 6 days after the timeframe of data which is selected to train the model.

Frontend to enter the ticker symbol and the timeframe:



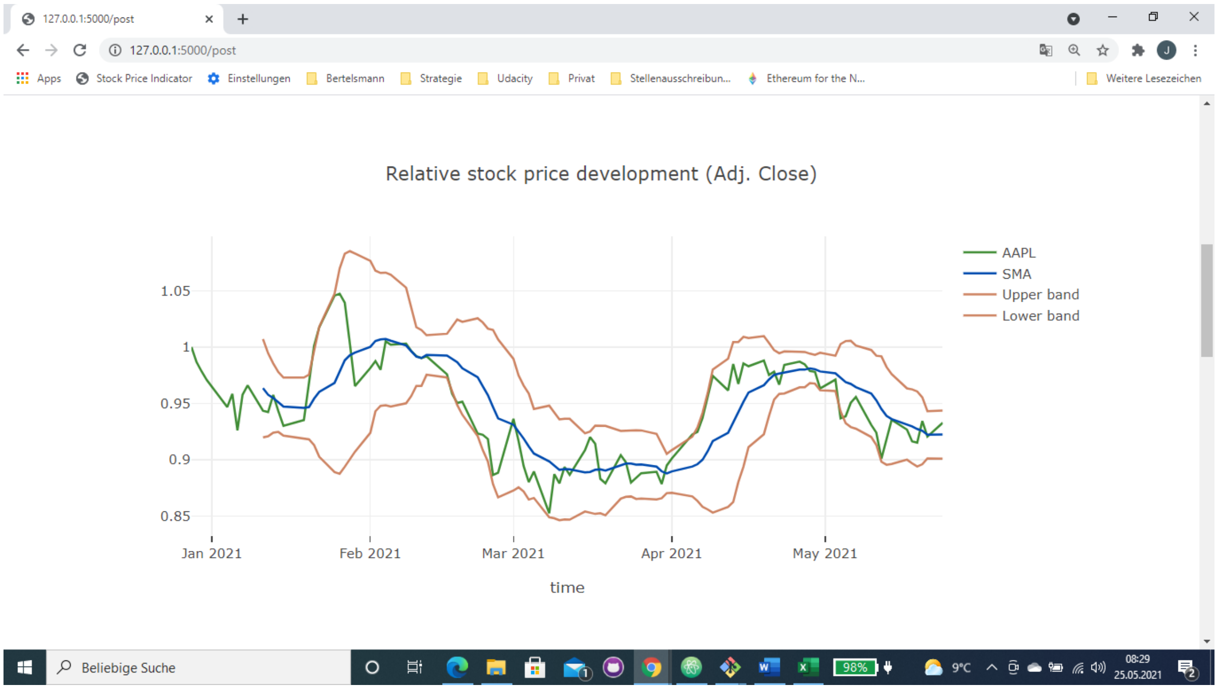
After submitting the entry data, the WebApp shows a technical analysis for that stock, the predicted stock prices for the next 6 days as well as the metrics of model performance.

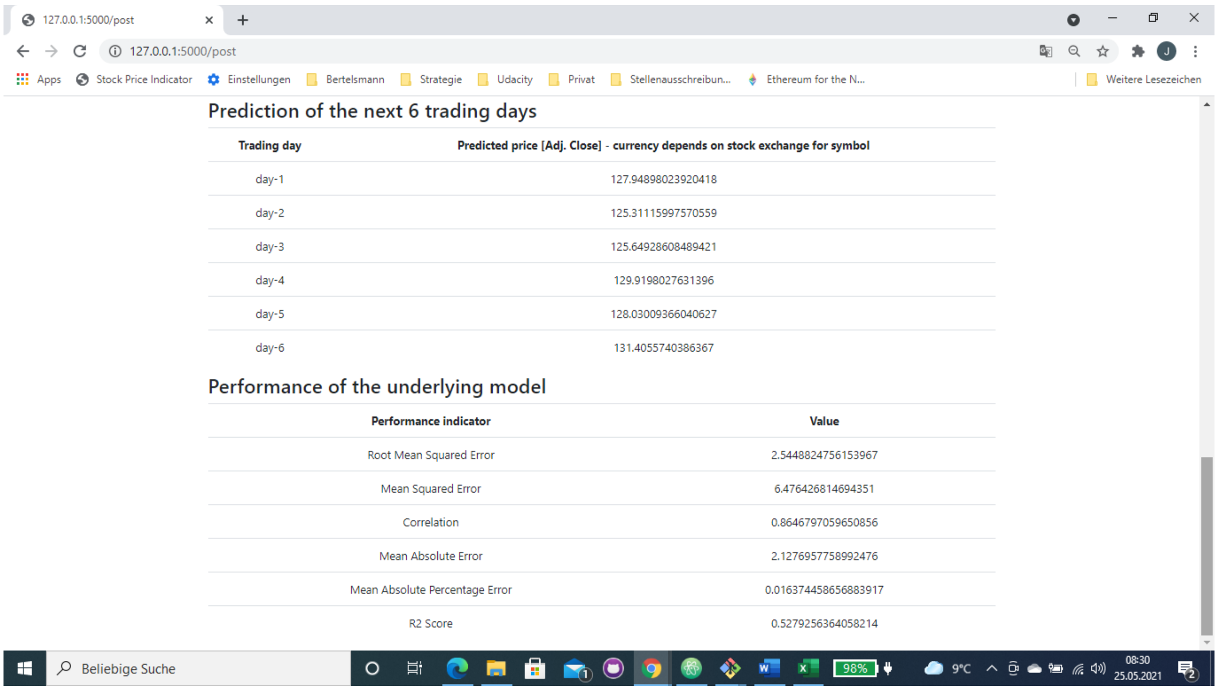

Stock Price Indicator

Click to find ticker symbol on Yahoo finance

Technical Analysis

including prediction of stock price development 6 days after selcted period




Prediction of the next 6 trading days

Trading day	Predicted price [Adj. Close] - currency depends on stock exchange for symbol
day-1	127.94898023920418
day-2	125.31115997570559
day-3	125.64928608489421
day-4	129.9198027631396
day-5	128.03009366040627
day-6	131.4055740386367

Performance of the underlying model

Performance indicator	Value
Root Mean Squared Error	2.5448824756153967
Mean Squared Error	6.476426814694351
Correlation	0.8646797059650856
Mean Absolute Error	2.1276957758992476
Mean Absolute Percentage Error	0.016374458656883917
R2 Score	0.5279256364058214

The creation of the WebApp from scratch has been quiet challenging. Therefore, it contains only basic functionalities. But it was a good experience having implemented the entire process from the download of the data to the publication of the results via the WebApp.

Hyperparameter tuning/Refinement

As described in chapter "data processing" different algorithms have been tested and compared in a first step. Starting with Lasso LARS linear regression, then trying K nearest neighbor and AdaBoost using KNN as base estimator. Finally, a Random Forest Regressor has been implemented.

K nearest neighbor has been tuned by raising the parameter k up to 15. For larger values the model results became worse. Boosting KNN using AdaBoost has not been more efficient than using KNN alone.

The random forest regression has been optimized by reducing the n_estimators from 100 down to 10. The criterion to measure the quality of a split nearly had no impact on the results. With regards to the min_samples_split, best results have been achieved around 10. Lower and larger values deoptimized the model.

Throughout all tests, prediction results using Lasso LARS linear regression and Random Forest have always been close to each other. This is true for the MAE, MAPE as well as for correlation and the R2 score. Finally Random Forest has been implemented in the class "ModelStockPrice" and thus is used in the WebApp as well as in the Python script "ML.py" as this algorithm does not overfit.

Prediction on Apple stock prices (AAPL) based on data selected 2020-12-25 to 2021-05-25	

linear_model - LassoLars (alpha = 0.1)
 	
`MAE 4.024405722480176`
`MAPE 0.030471534056191938`
`r2 -3.2235401510659374`
`CORRCOEF 0.7396411437309108`
	KNeighborsRegressor (n_neighbors=15)
	
	`MAE 5.548907661437991`
	`MAPE 0.0418930364944374`
	`r2 -6.2174682290762195`
	`CORRCOEF 0.5984261479585979`

AdaBoostRegressor (base_estimator=knn, random_state=42)	RandomForestRegressor (random_state=42, criterion='mse', n_estimators=10, min_samples_split=10)

`MAE 5.628074455261229`
`MAPE 0.042587832045289174`
`r2 -6.909284940860663`
`CORRCOEF 0.6413255652517044`
	`MAE 4.086759474104925`
	`MAPE 0.030903365209605235`
	`r2 -3.4192605404071212`
	`CORRCOEF 0.6703990894961048`

Results

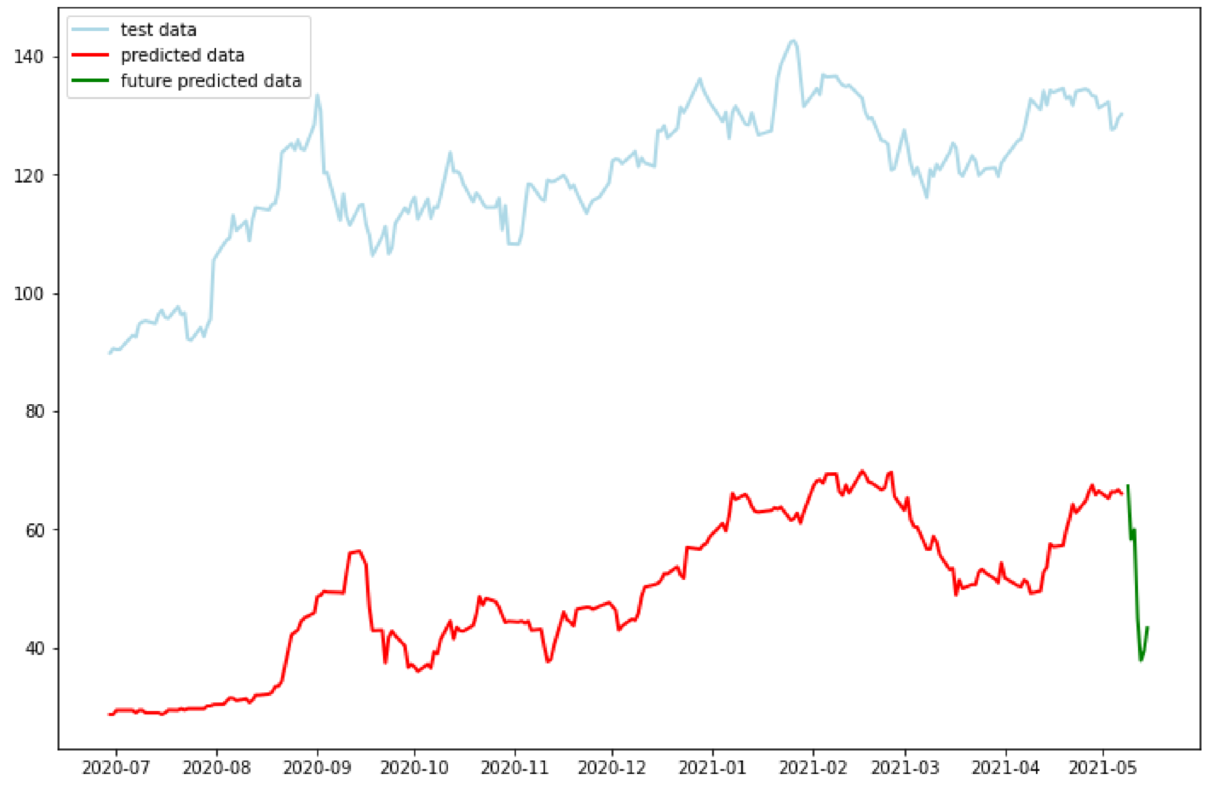
Model Evaluation and Validation

Although we could achieve a good performance with an MAPE of less than 3% in some cases and a correlation coefficient of more than 0.75, the robustness of the model depends a lot on the selected timeframe and the historical variance in stock price.

Looking at the impact of the timeframe first and taking the example of Apple, the prediction results differ a lot when selecting a longer timeframe.

Prediction on Apple stock prices (AAPL) based on data selected 2016-12-25 to 2021-05-25

RandomForestRegressor(random_state=42, criterion='mse', n_estimators=10, min_samples_split=10)
 	
`MAE 70.72237711970425`
`MAPE 0.5946331179500424`
`r2 -34.071015316901715`
`CORRCOEF 0.8265100820197979`

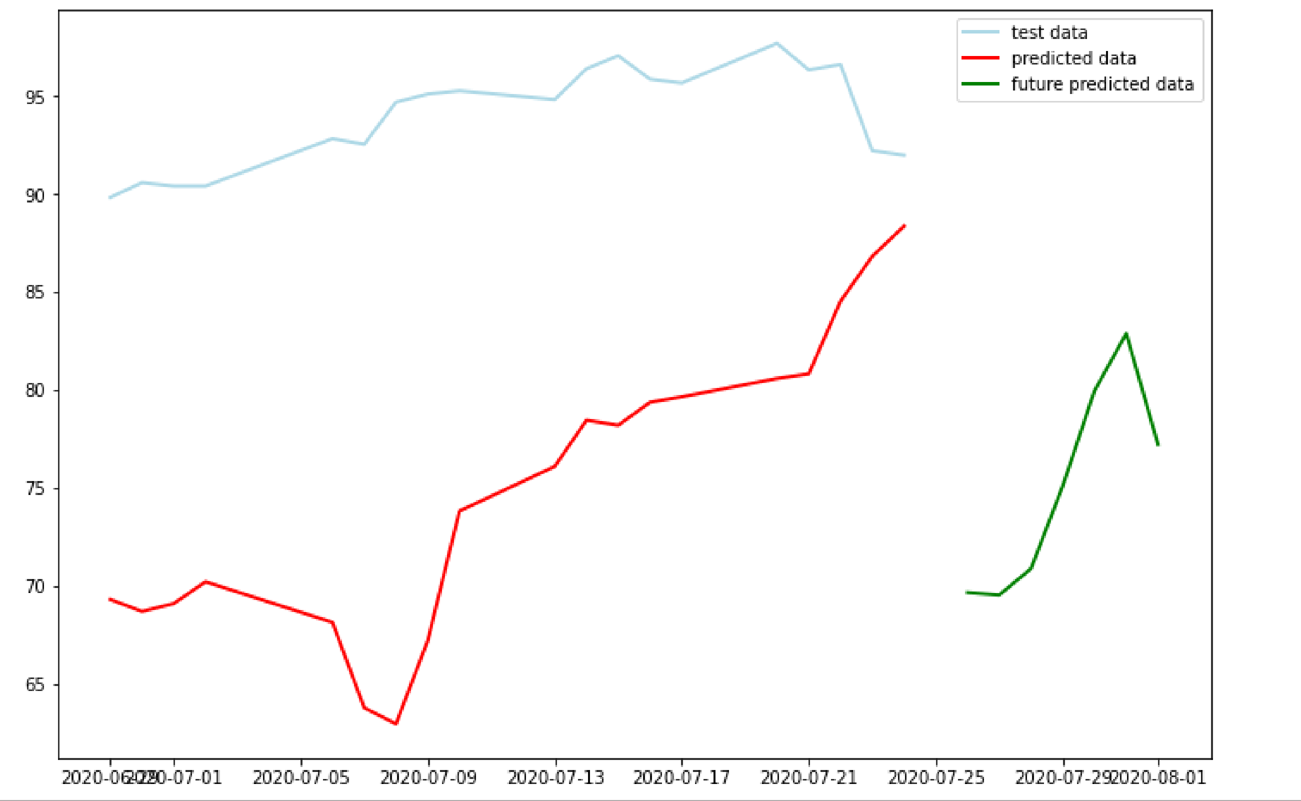


In another case we have analyzed a timeframe which contains more variance in stock prices as this was especially the case with the beginning of the corona pandemic in March 2020. Also, in this case the model only achieves a low performance.

Prediction on Apple stock prices (AAPL) based on data selected 2020-02-25 to 2020-07-25
(including impact of corona pandemic in
March 2020)

RandomForestRegressor(random_state=42, criterion='mse', n_estimators=10, min_samples_split=10)

```
MAE 18.96592857365407
MAPE 0.20196540680612018
r2 -65.09214814937806
CORRCOEF 0.3875717550901236
```

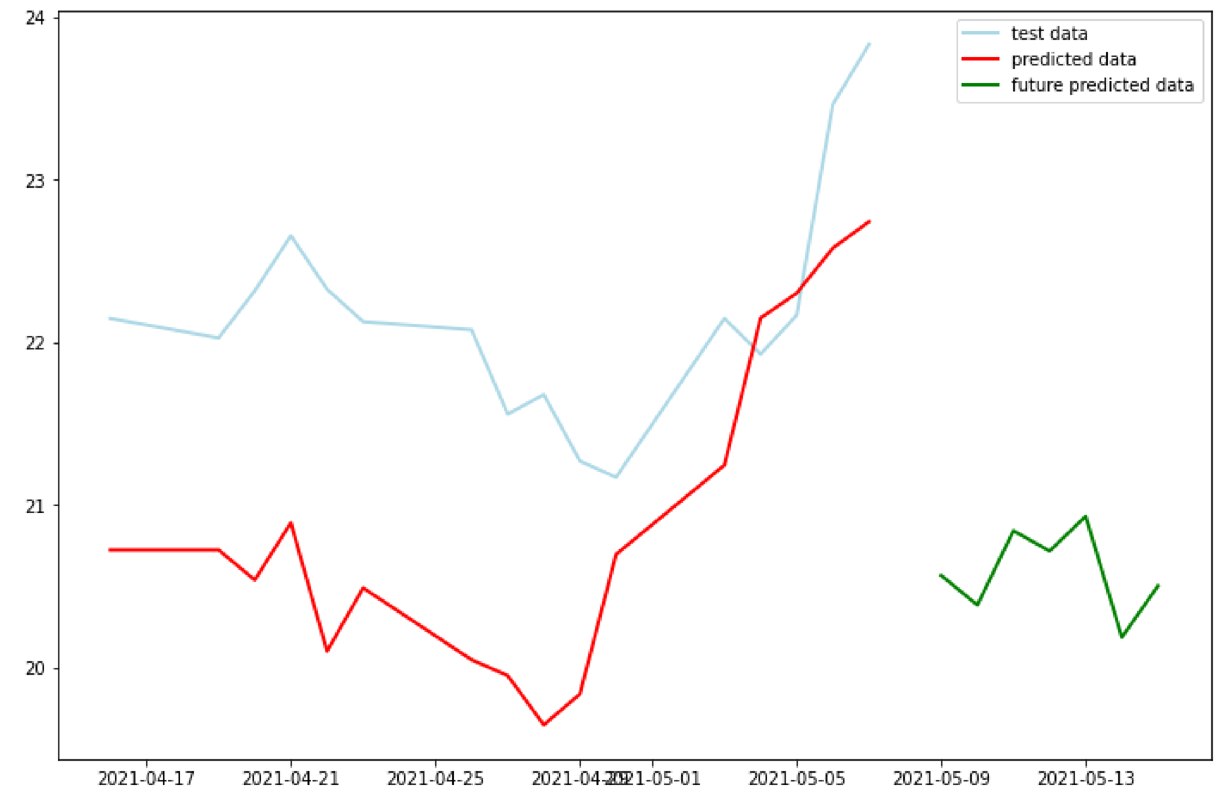


Taking Barrick Gold Corporation (GOLD) as an example for a stock with higher volatility this shows that this reduces the performance of the model too. When analyzing this stock in the same period as we did with Apple in the first example the evaluation returns a MAPE of 5.9 % and a correlation of 0.687 although the prediction results are still good.

Prediction on Barrick Gold stock prices (GOLD) based on data selected 2020-12-25 to 2021-05-25

RandomForestRegressor(random_state=42, criterion='mse', n_estimators=10, min_samples_split=10)

```
MAE 1.307948536424933
MAPE 0.059025585545846165
r2 -3.6471127649355557
CORRCOEF 0.6877166144458713
```

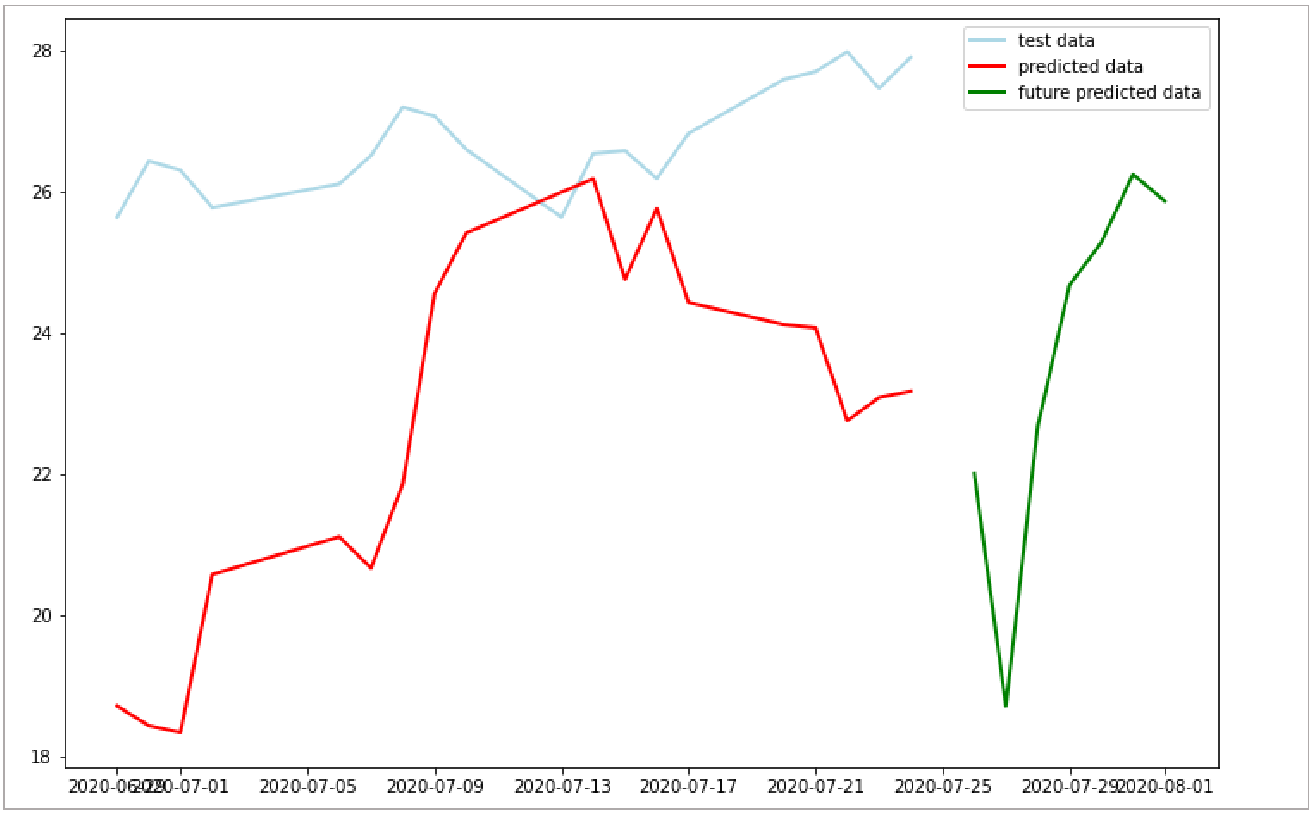


Taking the historical data of Barrick Gold over the time period of the main impacts of the corona pandemic, the model performance becomes also worse, but the impact on the model performance is not as strong as with Apple.

Prediction on Barris stock prices (GOLD) based on data selected 2020-02-25 to 2020-07-25
(including impact of corona pandemic in
March 2020)

RandomForestRegressor(random_state=42, criterion='mse', n_estimators=10, min_samples_split=10)

```
MAE 3.9367577458352456
MAPE 0.14724891780509883
r2 -39.66861153327688
CORRCOEF 0.23850556814512291
```



Justification

Summing it up, the performance of the model depends strongly on the selected period of historical data and the volatility of the stock. Therefore, it is crucial to take the evaluation metrics of the underlying model into account when looking at the predictions. It may be helpful to select a longer or shorter timeframe of historical data if the evaluation of the trained model shows strong deviations and low correlation.

Although we came to slightly better results using Lasso Lars linear regression in many cases, the model seams to overfit easily when it comes to shorter periods of historical data. As the shorter timeframes are more relevant to get good prediction results than longer periods as shown above, the risk of overfitting is an important aspect. Therefore, the Random Forest Regressor has been chosen for implementation in the Python scripts and the WebApp as it is more robust.

Conclusion

Reflection

As shown in this project there is a lot of historical data available as well as a brought range of very performant algorithms which can be trained to make predictions on stock prices. It is surprising that it is possible to setup the entire project including the download and storage of data, the testing of different machine learning algorithms and the publication in a WebApp after only a six month course in data science. It has been challenging but also a good training to deal with all aspects of the CRISP-DM. It makes clear how important each step is and that finding a working approach for each step is not a trivial task. Many iterations had to be made throughout the entire projects. For example, the data download had to be adopted when it became clear how to write the final scripts.

The most difficult part has been setting up the WebApp and splitting up the data into train and test sets so that finally good prediction results could be achieved. Clearly following the data science process helped a lot when deciding what to do next after each step. Implementing everything in a Jupyter Notebook first has made it easy to test everything before writing the scripts and the WebApp.

So finally, I can prove my professor at least partially wrong as it is possible to receive good results when predicting future stock prices. Although it depends strongly on the stock selected and on the chosen timeframe for the historical data. But I assume that there are still some improvements possible to optimize the results.

Improvements

In order to limit the scope of the project not all aspects how to solve this kind of problem have been implemented. To improve the existing way the problem has been addressed, it could make sense to split up the historical data in multiple sequences of training and test data. This might optimize the training process and lead to better results with regards to handling volatility in stock price development.

Further, we have tested different algorithms in modeling. In some cases, Lasso LARS linear regression achieved better results and in other cases Random Forest did. It could make sense always choosing the algorithm with the best results.

Up to now the project only takes technical indicators into account which have been derived from stock prices. There is a brought range of fundamental data available describing the performance of a company. Assuming, that stock prices follow the performance of the company at least in the mid-term or long-term, this kind of data could be helpful to achieve better prediction results for periods of one month or two.

Another kind of improvement could be to use reinforcement learning in order to predict whether to buy, sell or hold a stock. It would be interesting to analyze which problem-solving strategy achieves better results on a portfolio of stocks.