Predict DAX30 Stocks Prices

Ana Isabel Casado Gomez

2nd February 2021

Abstract

This is the capstone project report for the Machine Learning Nanodegree from Udacity. We focus on predicting the stock prices for the companies in the DAX30, the blue-chip stock marketing index with the 30 major German companies trading on the Frankfurt Stock Exchange. To provide the next best action for investing in stocks, in this project we compare the performance of different models and techniques. The best model is used in a dashboard to predict based on some parameters that the user should input.

1. Definition

1.1 Project Overview

The year 2020 has been a very special year for all the world: the first worldwide pandemic was announced. It did not only affect the health of the population all around the world with 2.13 Million deaths[1], but also the worldwide economy. The US stock market did record in March 2020 the worst performance since Black Monday in 1987 with €2 trillion lost, while Germany's DAX index suffered a historic drop[2]. The DAX[3] is a blue chip[4] stock market index consisting of the 30 major German companies trading on the Frankfurt Stock Exchange.

Stock modelling is a hot topic, already studied for decades, that won't be off-fashion any time soon due to the big volatility, the unforeseen factors that affect the markets and the high amount of money that it moves. Investors need a tool that helps them to make better decisions and to keep the risk low. Just with a fast search in google scholar under the keywords 'stocks forecasting modeling', we obtain over 18.000 results just from 2020. If we change the search to 'stocks machine learning' we obtain similar result numbers.

Time series[5] is a sequence of data points indexed in time order, generally equally spaced in time. Stock prices are one of the examples of time series. Time series forecasting used to be dominated by linear methods because they were easier to understand, however, with the explosion of Machine Learning and more specifically, the Deep Learning neural networks, time series forecasting problems have been used as supervised learning challenges for these methods[6] [Nelson et al. 2017].

In this project, we focus on the stock modelling for the German's market, DAX 30 using LSTM neural networks as an improved method versus the new facebook library, Prophet, to model time series data and the common well used ARIMA model.

1.2 Problem Statement

The main idea of this project is to understand what is the next best action for a given symbol or company. To know the next best action, the stocks must be predicted. The challenge of this proposal is to build a stock price predictor that takes daily trading data over a certain data range as input and outputs the projected estimates for the next k days. This is a regression problem as it will predict the price value, and not just if the stock price goes up or down, that would be the classification problem. After the predictions

are done, one recommendation to buy, hold, or sell will be suggested too. To achieve this, we will also compare predictions with time series versus neural network methods and prophet library from Facebook, to evaluate what is best for the given problem.

One of the challenges in this problem is to identify a unique method that performs satisfactorily for the different types of companies belonging to the DAX30.

1.3 Metrics

The metrics used to evaluate the performance of this project will be RMSE and Sharpe Ratio. RMSE will be the main metric to compare the results of the different models against the benchmark one.

- RMSE[7] is the root mean square errors and measures the differences between predicted values from a model and the actual observed values:

$$RMSE(\hat{y}) = \sqrt{MSE(\hat{y})} = \sqrt{E[(\hat{y} - y)^2]}$$

- Sharpe Ratio[8,9] was developed in 1966 by William F. Sharpe and measures the performance of the return of an investment compared to its risk.

$$Sharpe\ Ratio = \frac{E[R_p - R_f]}{\sigma_p} = \frac{E[R_p - R_f]}{\sqrt{var[R_p - R_f]}}$$

where:

- R_p is the return of the portfolio
- R_f is the risk-free rate
- σ_p is the standard deviation of the portfolio's excess return

The higher the Sharpe Ratio the better the company. We could say that a good Sharpe ratio would be [1,2), the interval [2,3) would be very good and +3 would be excellent. It means, any ratio <1 would not be considered good. However, it's not that simple to set the thresholds in order to define what is a good sharpe ratio for a unique symbol. To have a better judgement, if you decide to invest, the best way to decide between two or more companies in which one to put your savings will be the one with a higher Sharpe ratio. That is a good metric for comparison[10] between symbols, but not to judge a unique symbol.

Note, we will assume the risk-free for this analysis as 0 because nowadays the interest rates are so low (we would get very little money if we don't invest, and just leave it in a saving account), so the simplified formula is:

$$Sharpe\ Ratio = \mu_p / \sigma_p$$

2. Analysis

2.1 Datasets and Sources

The dataset used for this project comes from Yahoo! Finance. As Yahoo! finance stopped their historical data API, and it's not accessible anymore, the python *yfinance* library will be used to download historical market data. We will call symbols to the alias used in Yahoo! Finance to refer to a company.

The library retrieves as pandas data frame the historical dataset of the symbol provided and the following columns:

1. **Date**: day of the specified symbol as the index.
2. **Open**: the open price of the symbol on the given frequency time.
3. **High**: the highest price for the given frequency time.

4. **Low**: the lowest price for the given frequency time.
5. **Close**: the price that the symbol closed for the given frequency-time adjusted for splits.
6. **Adj Close**: the adjusted close price adjusted for both dividends and splits for the given frequency time.
7. **Volume**: the physical number of shares traded of that stock for the given frequency time.

The frequency of the data needed is daily, however, it has to be applied after the data is downloaded due to missing value like weekends. The target for our forecasting is **adj close.**

Table 1 is one example of the 5 first rows of the sample downloaded from *yfinance* library. We can observe that from the first row to the second we are missing 2 days (weekend). We will impute the data with interpolation methods.

Date	Open	High	Low	Close	Adj Close	Volume
2017-06-30	26.90	27.799999	25.260000	27.799999	27.799999	11407039
2017-07-03	28.25	28.750000	27.250000	27.500000	27.500000	496331
2017-07-04	27.50	27.770000	26.805000	26.950001	26.950001	172959
2017-07-05	26.98	27.280001	25.424999	25.549999	25.549999	251146
2017-07-06	25.65	26.225000	25.594999	25.900000	25.900000	164755

Table 1. Example of the 5 first rows of the dataset for the symbol DHER.DE - Delivery Hero AG.

As we are dealing with time series, the index of the dataset is the DateTime of the stocks with daily frequency.

2.2 Data Exploration and Visualization

Firstly, we are going to show an overview of the time series data we will be dealing with. Figure 1 shows the times series for all the 30 symbols we are going to be modelling. We can see clearly here, how there are some points missing in the series and the drop in all the stocks in march 2020, corresponding to the impact of the COVID19.

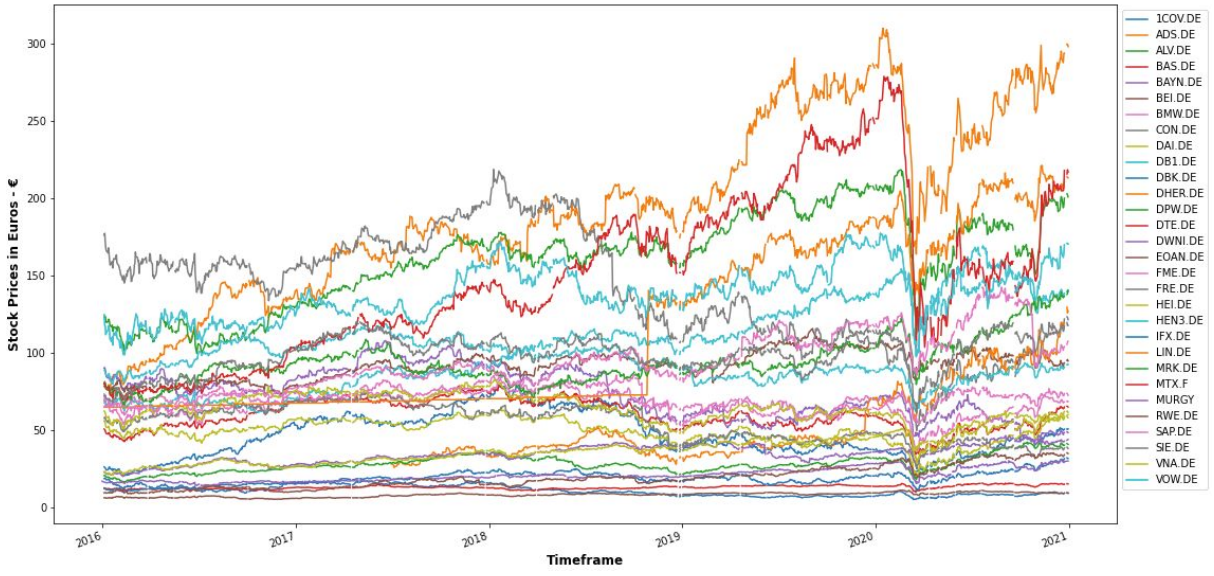


Figure 1. Time Series of all DAX30 companies during the last 5 years.

Secondly, after visualizing the stock time series for all the companies, we are going to study a bit more the distribution of each one. Figure 2 shows an overview of the distribution of the stock prices of DAX30 companies. We easily can see that some of them are more spread than others and some of them do not contain any big change during the timeframe analyzed (5 years). The symbol with a lower value for the stocks is *EOAN.DE*, the spread is very low too, therefore the range of evolution was very small. The symbol that higher price has had at any point in the 5 years was *ADS.DE* as its maximum value is higher than 300€. Outliers are also observed for some symbols. *BMW.DE* or *DAI.DE* have them under the minimum value, which could be due to a big crash suffered during this time. Other companies like DHER.DE and *MRK.DE* have outliers over the maximum, which leads to higher peaks of the prices.

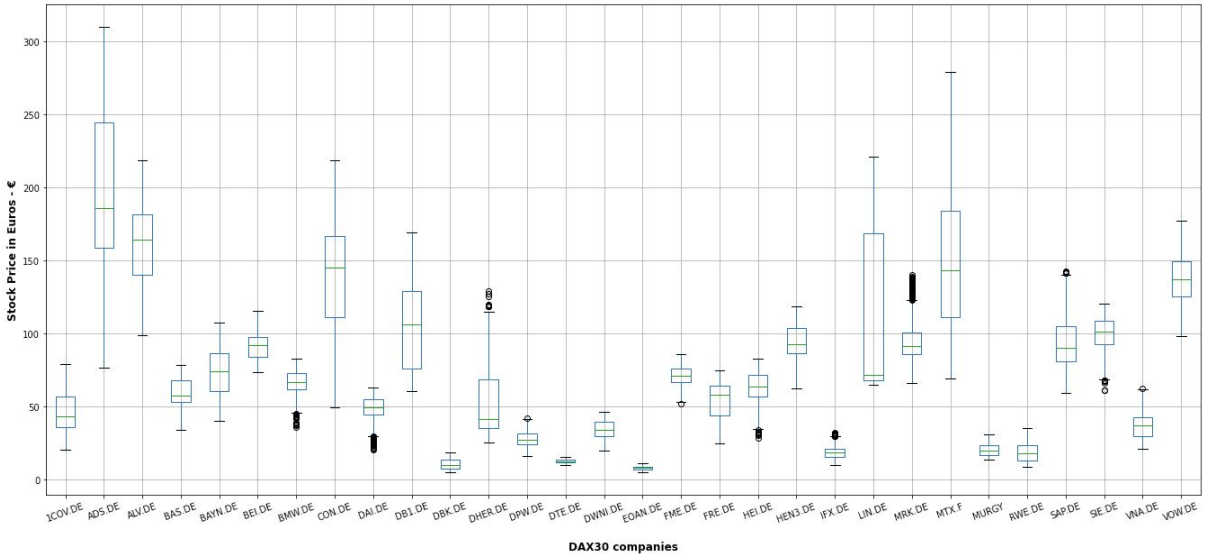


Figure 2. Description of the variation of the dataset through their quartiles and outliers.

The shape of our data is (1291 rows, 180 columns). All the companies have a count of 1265 values except DHER.DE that are 884 because it entered into the stock market after 2016, exactly in July 2017. The difference between the shape and the non-values count shows missing values in our sample. After an exploration of the index, we see that the weekends are not in the dataset, but the holidays are presented with null values. Also, some extra dates out of the range we submit are presented. Those rows are removed or imputed. The methodology followed is exposed and discussed in the preprocessing part 3.1.

After cleaning the dataset and imputing missing values, the total amount of our input data is 1823 days per company. It means, our input set contains 533 imputed data from weekends and holidays and 1290 trading days. In other words, we have imputed the 29% of the total sample. Except for DHER.DE that we had 884 data points and after imputation 1280.

2.1.1 Relationship between observation and other previous observation

Time series modelling assumes the relationship between an observation with the previous one (lags). Figure 3, shows a positive correlation relationship in all the companies for a lag of 15. After 15, some of the symbols become more spread. With less than 15, the correlation is stronger. This is a good sign for choosing the parameters for our benchmark model.

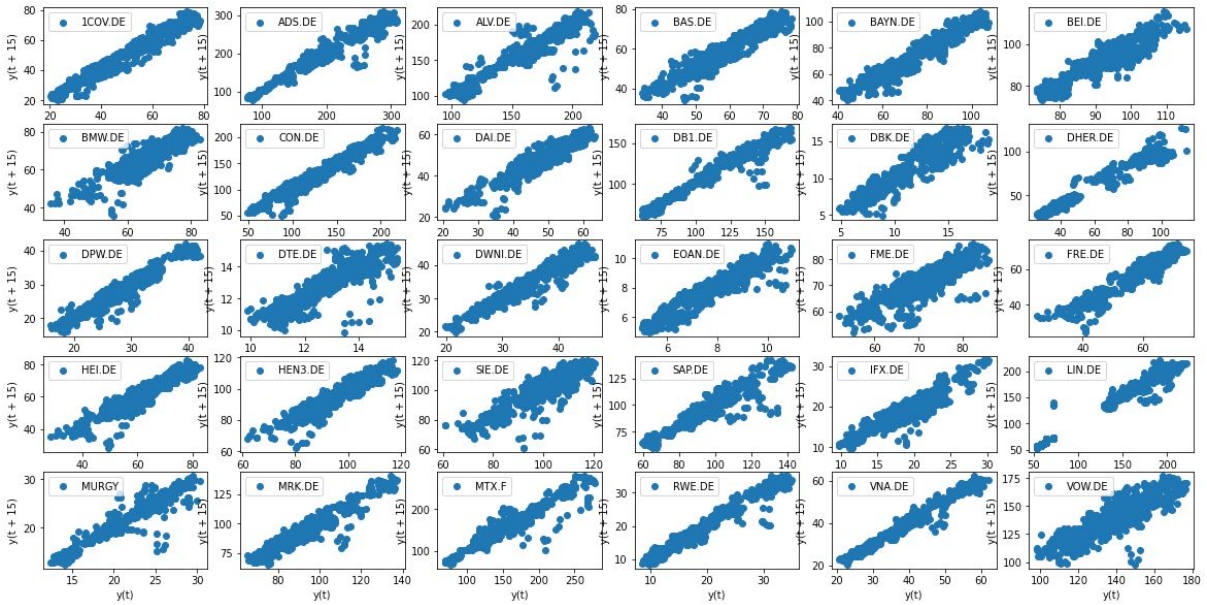


Figure 3. Time Series Lag = 15 Scatter Plot

In time series, histograms are also useful to get insights into the underlying distribution of the observations. However, the density plot will show better data distribution. In Figure 4, it is not observed for any of the 30 analyzed companies any distribution similar to Gaussian distribution.

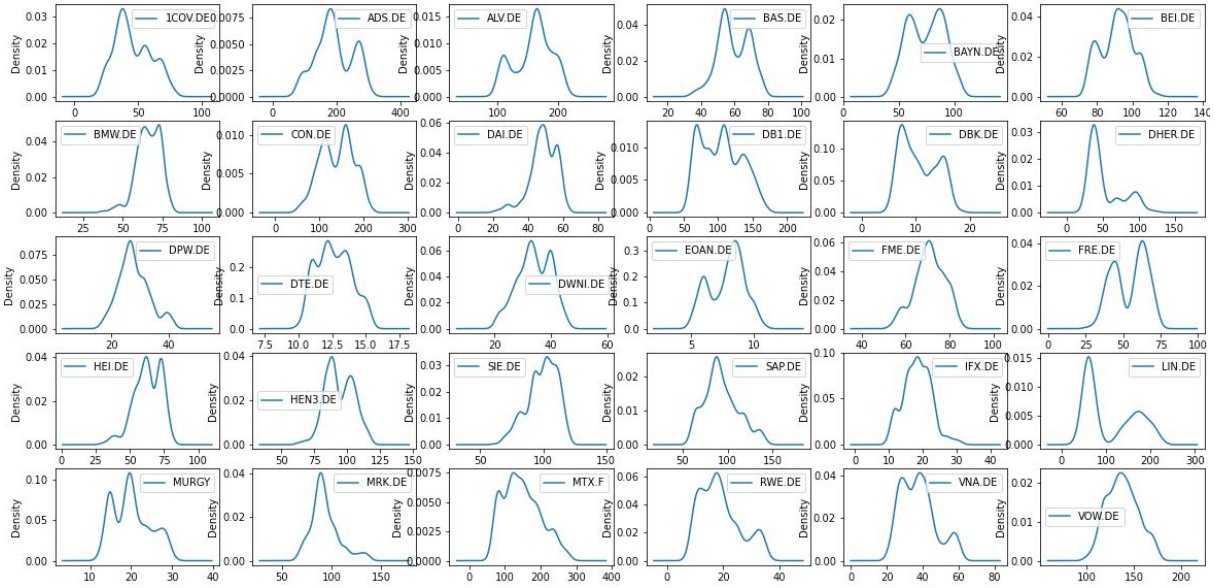


Figure 4. Density Plot for all the companies in DAX.

2.1.2 Sharpe Ratio in DAX from lifetime, 2019 and 2020

The Sharpe Ratio (SR) we will use in this part are defined as follows:

- SP Lifetime is computed from the 1st January 2016 until 31st December 2020, counting only trading days. Weekends and holidays are not counting for the computation of this metric.
- So accordingly, the Sharpe ratio from 2019 would include trading days for 2019 and 2020.
- The SR 2020 is computed for the trading days belonging just to the year 2020.

Figure 5 with bar charts shows the comparison for Sharpe Ratio for the 3 different periods studied.

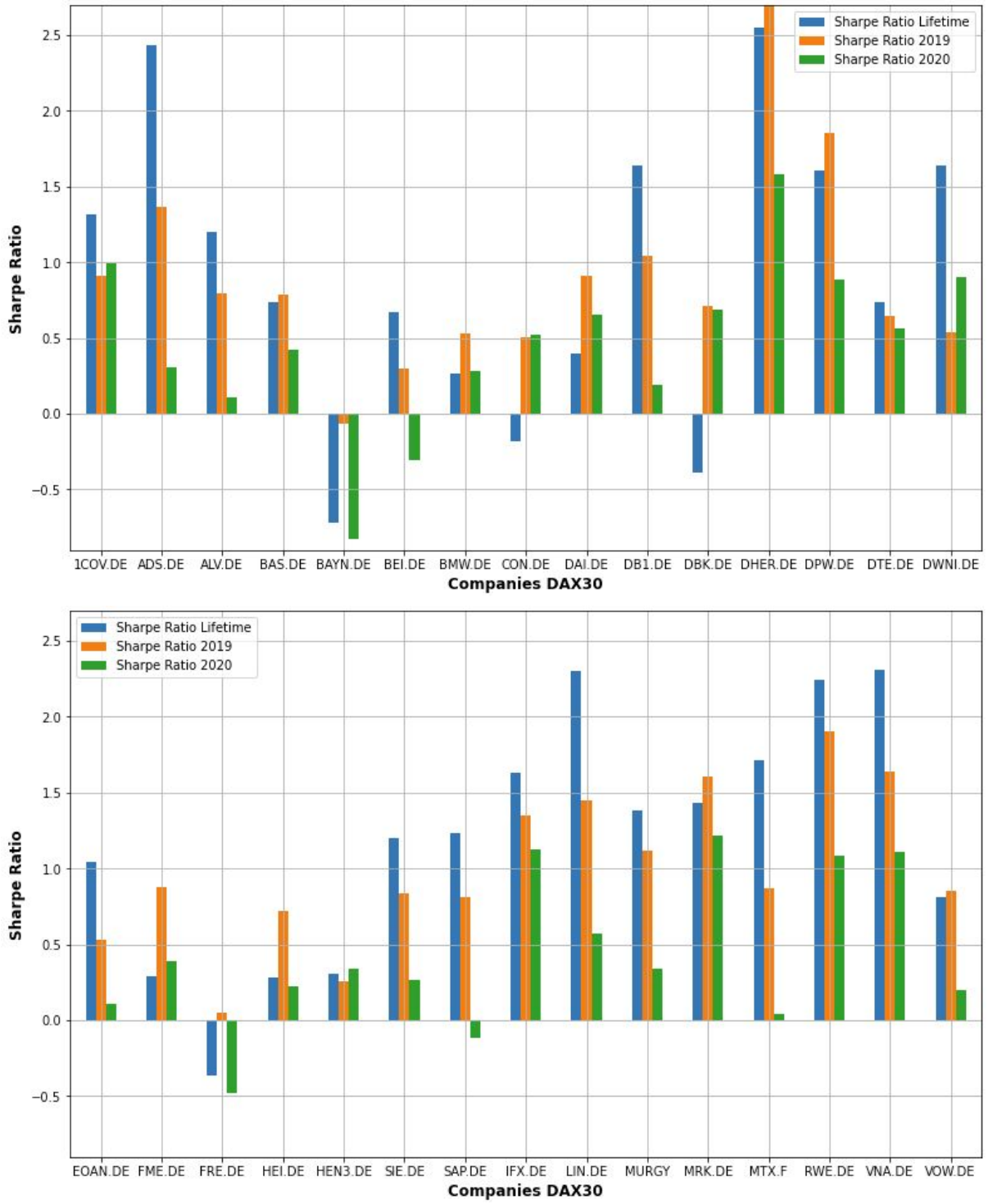


Figure 5. Comparison of the sharpe ratios from lifetime, 2019 and 2020 for DAX30.

From the DAX30 companies, we can see how the best one to invest would be *DHER.DE*, the newest in the group as we have also fewer data points for it (only in the market since June 2017). The SR for all the companies in 2020 is lower than for the other 2 timeframes studied, this is obviously due to the COVID-19.

Figure 5 compares evolution of stock prices for symbols that are in different Sharpe Ratio range from each other in their lifetime. From the 30 symbols, we visualize the symbols with the best and worst Sharpe symbol to analyze the volatility and price change. *DHER.DE* and *RWE.DE* are the ones with higher

SR continuously in the 3 periods, *1COV.DE* has a mid-performance and *BEI.DE* and *FRE.DE* are too close to 0 or even with negative values.

Figure 6. Evolution of Stock Prices for 5 selected symbols from DAX30.

The ones with better SP have a higher return of investment as they see a continuously increasing tendency. *1COV.DE* has a very good performance in the 3 first years and worst in the last 2 years. *FRE.DE* continuously decreases.

If we were investors that decided to invest just based on Sharpe Ratio, from all the DAX-30 we would choose Delivery Hero AG as the next company from DAX to include in our portfolio.

2.3 Benchmark

The baseline model used to compare our Prophet and NN models will be one of the most traditional and well-known time series models: ARIMA. The reason is because it is a generalization of an autoregressive moving average model and easily can be applied to data where there is no evidence of non-stationarity[13].

ARIMA(p,d,q) model has 3 key aspects[14]:

- **Autoregression** (AR). It uses the relationship between an observation and some number of lagged observations.
    - **p** parameter (lag order) defines the number of lag observations.
- **Integrated** (I). It uses the differencing of raw observations to make the time series stationary.
    - **d** (degree of differencing) defines the times the raw observations are differenced.
- **Moving Average** (MA). It uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.
    - **q** parameter (order of moving average) defines the size of the moving average window.

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	1821		**coef**	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Model:	ARIMA(4, 1, 0)	**Log Likelihood**	-4525.850	**const**	0.1204	0.077	1.554	0.120	-0.031	0.272
Method:	css-mle	**S.D. of innovations**	2.905	**ar.L1.D.y**	0.0961	0.023	4.102	0.000	0.050	0.142
Date:	Sat, 30 Jan 2021	**AIC**	9063.699	**ar.L2.D.y**	0.0372	0.024	1.580	0.114	-0.009	0.083
Time:	20:55:30	**BIC**	9096.742	**ar.L3.D.y**	-0.0091	0.024	-0.387	0.699	-0.055	0.037
Sample:	1	**HQIC**	9075.889	**ar.L4.D.y**	-0.0029	0.023	-0.125	0.901	-0.049	0.043

Table 2. ARIMA model summary for the symbol ADS.DE. More in *data_exploration.ipynb*

2.4. Algorithms and Techniques

The 2 models that we will develop in this project for the given problem are Prophet model from prophet library and Neural Network models. Their RMSE performance will be compared against the benchmark model we described in the previous section, ARIMA. In this section we will explain further the type of models we will work with.

The type of Neural Network for this project is LSTM[11] (long short term memory) recurrent neural Network as it handles the order between observations when learning a mapping function from inputs to output as well as temporal dependence. A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**.

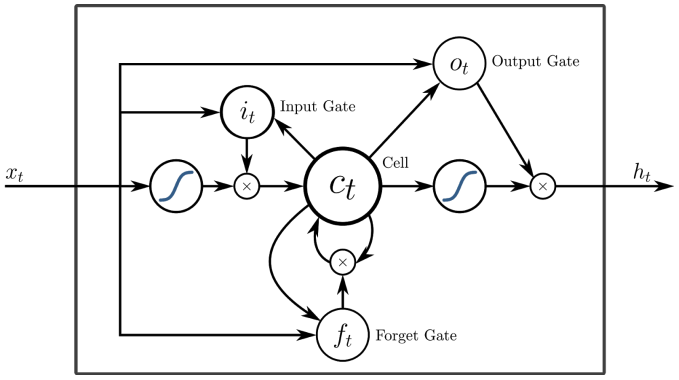


Image 1. The Long Short-Term Memory (LSTM) cell. Source Wikipedia/long_short-term_memory

Different structures are analyzed in this project from Vanilla LSTM to 4 stack layers (see Diagram 1) with one exceptional method to learn the input sequence from both forward and backwards: Bidirectional LSTM (Diagram 2). Dropout will be applied to each layer to avoid overfitting. We will use the sklearn library to create the models with Dense, LSTM and Bidirectional layers.

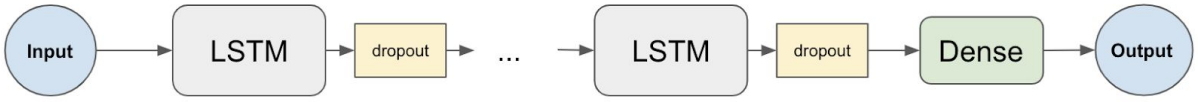


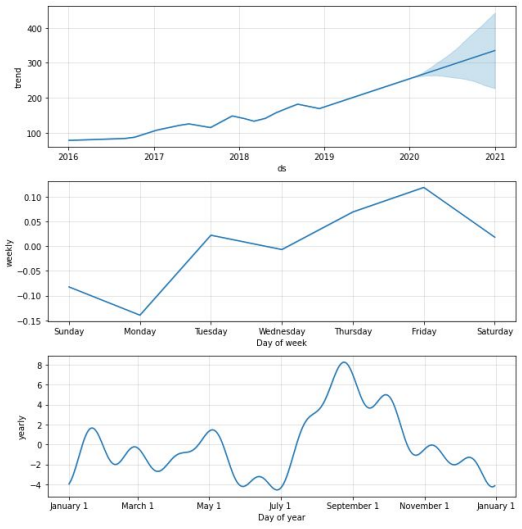
Diagram 1. LSTM neural network structure for multiple stack layers (up to 4).

Diagram 2. Bidirectional LSTM neural network.

A summary of the Vanilla LSTM model is shown in Table 3. It consists of only one layer of LSTM, a dropout and a dense one. In this example we used 50 units for LSTM and 30 for the first Dense layer. The output is 1 as it would predict only 1 day. It's not the model we have developed and optimized, but an example shown in the *data_exploration.ipynb* notebook.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 50)                10400
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 30)                1530
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 31
=================================================================
Total params: 11,961
Trainable params: 11,961
Non-trainable params: 0
```

Table 3. Summary of Vanilla LSTM model for symbol *HEI.DE*



In addition to the LSTM neural networks models, Prophet library from Facebook's Core Data Science team, released as open source[12] , is evaluated. It is a procedure for forecasting time series data based on an additive model where non-linear trends can be fit with yearly, weekly and daily seasonality, plus holiday effect. It is more effective on time series with strong seasonal effects and enough historical data. This model was chosen as a way to contrast a deep learning model with a more traditional one on top of the benchmarking. The mathematical equation behind the model is:

$$y(t) = g(t) + s(t) + h(t) + e(t)$$

with $g(t)$ the trend forecasted with piecewise linear model, $s(t)$ the periodic changes (weekly, monthly, yearly), $h(t)$ the effect of holidays and $e(t)$ the error term. The 3 charts on the right show some output components of modeling with the Prophet. The first one shows the yearly trend. The second the weekly, where we can see that for this example MTX.F symbol, the higher values are on friday while monday has the lowest. With the daily trend we see that July is the month with lowest values and September, on the contrary, the highest.

2.5 Next Best Action

After we have found the best performing model, we will evaluate the best action to recommend the clients regarding to:

- Investing in the company
- Sell or hold the current investment from that symbol.

The decision will be taken based on some parameters that the client should specify, such as the amount of stocks owned from that symbol and the day of acquisition. If the client doesn't have any stocks, 28 days for computation will be used.

We will assume a moderate risk strategy. We sell the stocks before losing more than 10% of the investment done or after we have gained more than 30% of the investment. With this in mind, we create an interval called *game_range* with `[investment-10%*investment, investment+30%*investment]`.

The recommendation about the investment will be based on the forecast provided for the next 5 days:

1. If the forecast investment falls into the *game_range*, the client will be advised to HOLD.
2. If the forecast investment does not fall inside the *game_range* limits, the advice is to SELL.

The recommendation regarding the company will be based on the mean of the norm return for the parameters specified by the client (amount of stocks and day of acquisition) or based on the last 28 days performance.

1. If the mean is bigger than 1.01, the client will be advised to BUY MORE as the company usually performs well.
2. If the mean is lower than 0.95, the advice will be against investing in the company.

If the mean is between 0.95 and 1.01, the client won't receive positive or negative advice, just that the company doesn't have enough volatility. Based on this the client could take the risk of investing or not but the money won't increase as fast as he/she could think.

3. Methodology

3.1 Data Preprocessing

The time frame selected to download the stock data was from 2016-01-01 to 2020-12-31. However, during the exploration phase, the dataset contained the first row 2015-12-31 with all the values set as NA. To consider only the data we framed, we did remove any data out the specified limits.

The dataset was also forced to have a frequency of Days. With *pandas* library, it was easily achieved with the panda's method `.asfreq('D')`.
As the data for all the companies are downloaded together, the output from *yfinance* library contains a multicolumn index. The preprocessing of this part carries out a swap of the column names level to be able to select the data based on the symbol, and then each subset has the same columns as Table 1.

3.1.1 Missing values Imputation

As we are dealing with the Adjusted Close prices in the stock DAX market, we have daily time series with missing values on the weekends and holidays, as the stock market only operates from Monday to Friday. Those values are imputed thought interpolation methods.

The interpolation methods chosen for this project are Linear and Quadratic, see Figure 6. To understand which of the methods leads to a better performance of our models, we have used the benchmark ARIMA model to decide which one we will implement. The results were positive towards the **linear interpolation** method with continuously lower RMSE for 18 out of 30 companies analyzed in DAX. The differences were very small < 0.05.

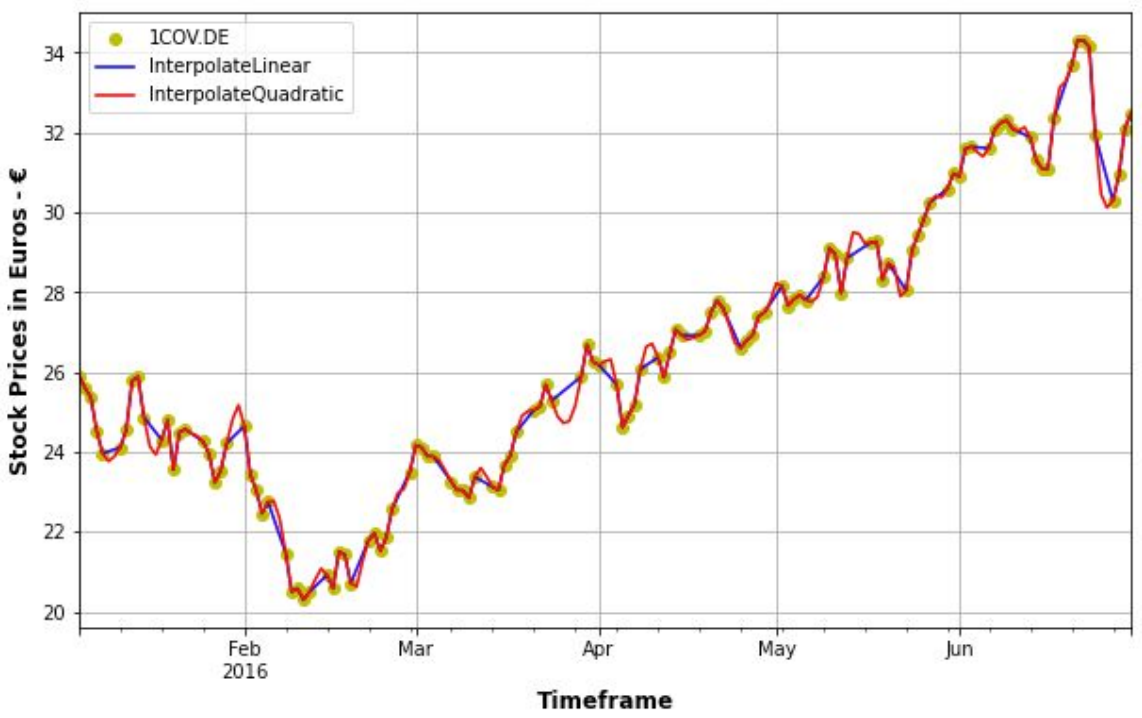


Figure 7. Time series with Linear & Quadratic imputation methods visualized for the first 6 months of 2016 for 1COV.

3.1.2 Normalization

For the neural network model, the data is scaled to avoid the model to learn large weights that are often unstable and could lead to poor performance during the learning stage and sensitivity to input values with a higher generalization error.

The normalization technique is applied to the training dataset with the method `from sklearn.preprocessing import MinMaxScaler` that follows the formula:

$$x_{scaled} = (x - min(x)) / (max(x) - min(x))$$

This also applies to the test set and the inverse transformation is used to visualize the predicted values at the original scale. Normalization technique was not applied to the input data for ARIMA or Prophet model.

3.1.3. Neural Network Input Features

We are going to build a multi-layer LSTM recurrent neural network to predict the last values of a sequence of values. The input data are built with a lag of 1 day and X time-steps. It means, we will be using to train the model, the X previous days to compute the next 5 days. During the development of this project, a time step of 60 days was tried out too with a worse RMSE metric. The value will be also tuned in order to find the best one associated with the best model. Finally, the input data for the LSTM models were reshaped into the format (# values, #time-steps, 1).

3.2. Implementation

The implementation has been divided into 3 notebooks:

1. The notebook *data_exploration.ipynb*covers the exploration shown in 2.Analysis.
2. The notebook *hyperparameter_tuning.ipynb* carries out the tune of the models.

3. The notebook *results.ipynb* show the model comparison and decision function.

The modeling part has been split into different class methods in the script *models_class.py*. Each class holds the training and test data preparation for Cross Validation (CV), model fit and predict, and finally, RMSE metric computation for each of the proposed models: ARIMA, Prophet and LSTM. The class for LSTM allows us to evaluate LSTM from 1 layer till 4 layers and a bidirectional LSTM neural network. In order to compose the methods, we have created functions in *model_utils.py* that build up from the models class, and other functions that would decide which model to run and apply cross validation. In this way, we are able to train, test and tune models faster. Each model uses a different python library:

- ARIMA model: `from statsmodels.tsa.arima_model import ARIMA`
- Prophet model: `from fbprophet import Prophet`
- LSTM NN model: `from keras.models import Sequential`

This project also contains different files to complete the job:

1. data_config.py: configuration with list of symbols to include in the analysis from DAX30.
2. data_preprocessing: script holding the functions to download the data, save in csv and read from it. In addition will also prepare the data and impute missing values.
    a. column_file.py result file from running this script, to save column config of dataset into our local to work offline.
    b. data_30_2016.csv: CSV file as result of running this script with the data points from all the DAX 30 companies without column names.

The rest of the functions used were defined in the notebooks. The exploratory notebook also includes one example of each model fully built and evaluated with graphs to visualize the predictions and the actual values.

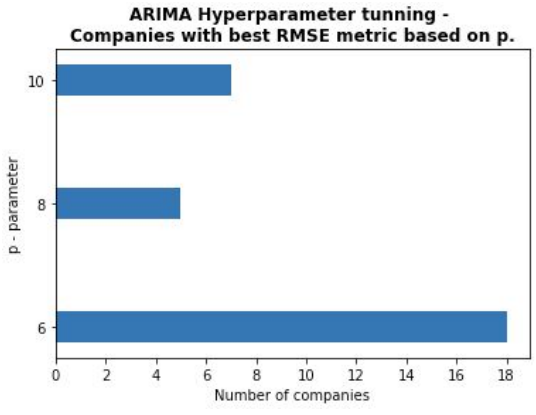
The interface[15] to hold the recommendation of the next best action was implemented using `dash`, an open-source Python framework used for building analytical web applications. All the details in order to run the notebooks and the dashapp is explained in the `README.md`.

3.2.1 Complications

During the implementation of the project, one of the main complications found was the data preparation to feed each model. For ARIMA the preparation was basic, just split in train and test sets but for prophet already includes an specific naming of columns in order to read the data. For the LSTM model, the input data had to be completely modified from the raw series. It broughts more complexity than expected as standardization could not be applied. The horizons of each model by nature were different too, so each one had to be modified in order to be compared under the same circumstances.

The other big complication was to create the interface. Due to lack of experience with the library, a considerable amount of time was invested on developing it. Basically, it was built from 0 knowledge.

3.3 Refinement



The ARIMA model was tuned for the hyperparameter ***p*** (lag order) with a range from 2 to 16 with a step of 2. Hyperparameter tuning was performed for 3 different symbols where each one had a best p parameter different in the range of 6 to 10. The next step was to perform the hyperparameter tuning in

that range for all the symbols. The left table shows what was the best parameter for a higher number of companies based on RMSE. The best ***p*** was 6 for 18 out of 30 companies.

The Prophet model is only tuned forcing the **daily_seasonality** and **yearly_seasonality.** We will tune them to find which one is the best combination of parameters under the current use case. This process was carried out for the 30 companies in DAX and the result is shown in Figure 8. The best combination of parameters is **True-True** because it's the winner for 16 companies out of 30.

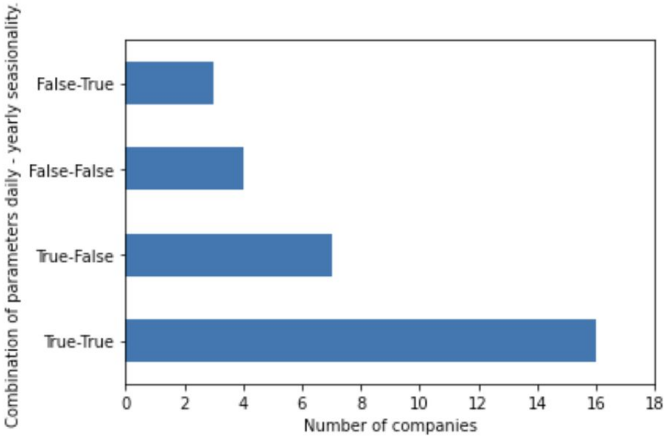


Figure 8. Prophet daily/yearly seasonality tuning comparison of companies with the best RMSE.

For Neural Network model, we tune in 3 stages:

1. Tune Vanilla LSTM (LSTM model with just 1 layer) for *epochs* in *[50, 80, 100]*. The best result for the first 3 symbols is ***100***.

2. Find from the different LSTM structures, which one performs better under parameters:
    a. time_steps = *30*
    b. units = *50*
    c. dropout = *0.2*
    d. epochs = *100*
    e. optimizer = *'adam'*
    f. loss = *'mean_squared_error'*
    g. batch_size = *32*
    h. activation = *'sigmoid'*

3. From best model (**Bidirectional** LSTM) tune the parameters associated to this model for the first 3 symbols:
    a. time_steps in *[15, 30, 60].*
    b. units in *[30, 40, 50].*

The best refined model is Bidirectional LSTM (see Diagram 2) with the parameters:

a. time_steps = *15*
b. units = *30*
c. optimizer = *'adam'*
d. loss = *'mean_squared_error'*
e. batch_size = *32*
f. activation = *'sigmoid'*
g. epochs = 100

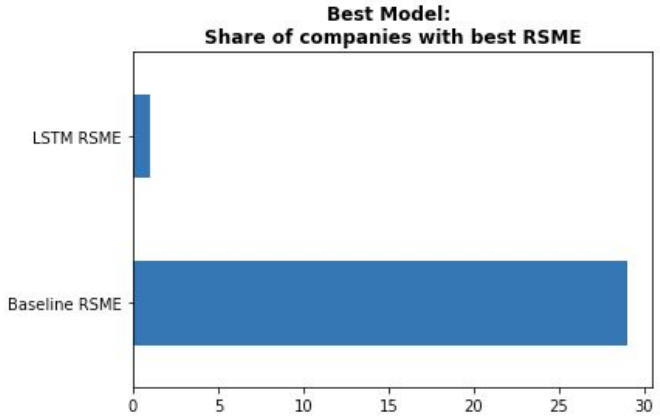
Table 4 shows the minimum RMSE for this model under a cross validation with 8 folds, before and after the hyperparameter tuning. We can see how the changes are quite different with ADS.DE up to almost 3 points of difference.

Table 4. Summary of RMSE for 3 symbols before and after tuning hyperparameters.

Symbol	RMSE after tuning	RMSE before tuning
1COV.DE	1.8903	2.1519
ADS.DE	5.2956	8.1932
ALV.DE	4.7687	5.6137

4. Results

4.2 Model Evaluation and Validation



With the 3 models tuned, we have evaluated them for the 30 companies through cross validation with 8 folds of data and 5 days of horizon to predict each fold. The bar chart in the left shows the comparison. The best model with usually the lowest RMSE metrics is the baseline ARIMA model with p=6, d=1 and 1=0 for 29 companies out of 30. This model will be used for the decision function in *results.py* script and in the dash_app in *dashapp.app*. It is worth mentioning that Prophet did not give the best results for any of the companies analysed. The full notebook with the results and comparison is allocated in *results.ipynb*.

The Best Next Action Decision Function is the last part shown in the results notebook. In the example, it is computed for the symbol DHER.DE under the assumptions that the client owns 50 stocks acquired 1st November 2020. In the left chart of Figure 9 the predictions from the best model (ARIMA) are computed for the next 5 days. On the right, the forecast includes data from the beginning of 2021.

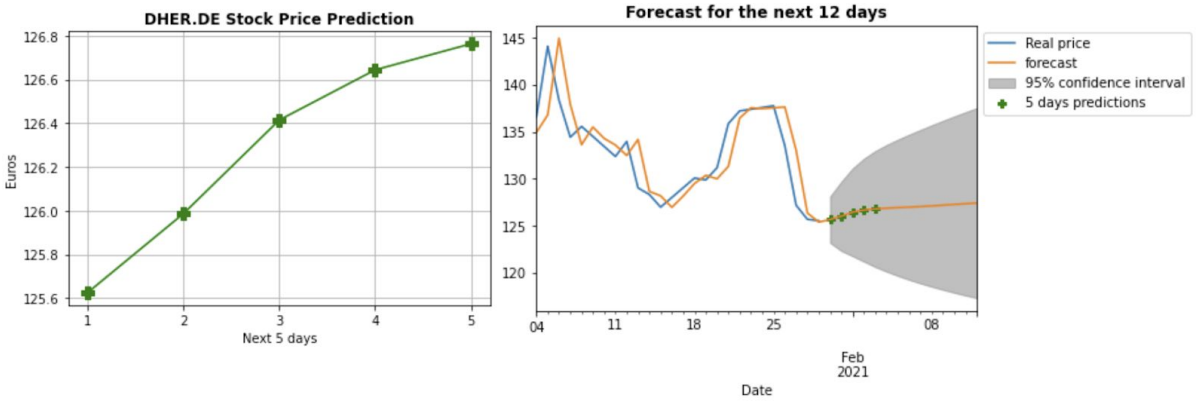


Figure 9. Left chart, stock predictions for the next 5 days. Right chart, those predictions with the 2021 data.

The recommendation of our model under the previous assumptions are:

```
Regarding your investment...
+ Hold On, game is still on.

Regarding the company...
++ Usually this company wins, invest more!
```

This model and advice given is implemented in the dashapp.

4.4 Justification

From the exploratory analysis on our data, we don't see any strong seasonal effect, which may be the reason that the Prophet does not perform better than our benchmark or LSTM model. On the contrary, LSTM doesn't catch as well as ARIMA the seasonality and trends, which could mean that even if it has memory, for this case the LSTM model proposed was not good enough.

At the moment of developing the proposal for the project, the ARIMA model was decided because it is the simpler and more known method available for time series forecasting. It is one of the first models learnt about the subject. However, it was not expected that the other new models like the one from Prophet library that so much effort did put Facebook into the development and the recurrent neural network model would perform as first worst than the baseline chosen method. It made the refinement part longer than expected, removing time from tasks that were initially proposed or time to further tune the Neural Network model to perform better than ARIMA.

4.4 Improvements

List of items that could improve this project but they were not tried out due to lack of time:

1. Different structures of LSTM from the ones tried here.
2. Different types of input data for LSTM, instead of time-steps, try without, using all the data to compute the horizon.

5. References

[1] At 25/01/2021: https://news.google.com/covid19/map?hl=en-US&gl=US&ceid=US%3Aen
[2] https://www.dw.com/en/stock-markets-plunge-on-coronavirus-fears/a-52790114
[3] https://en.wikipedia.org/wiki/DAX
[4] https://en.wikipedia.org/wiki/Blue_chip_(stock_market)
[5] https://en.wikipedia.org/wiki/Time_series
[6] D. M. Q. Nelson, A. C. M. Pereira and R. A. de Oliveira, "Stock market's price movement prediction with LSTM neural networks," *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, 2017, pp. 1419-1426, doi: 10.1109/IJCNN.2017.7966019.
[7] https://en.wikipedia.org/wiki/Root-mean-square_deviation
[8] https://www.investopedia.com/terms/s/sharperatio.asp
[9] https://en.wikipedia.org/wiki/Sharpe_ratio
[10] https://win-vector.com/2015/06/27/what-is-a-good-sharpe-ratio/
[11] https://en.wikipedia.org/wiki/Long_short-term_memory
[12] https://facebook.github.io/prophet/
[13] https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
[14] https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
[15] https://www.statworx.com/de/blog/how-to-build-a-dashboard-in-python-plotly-dash-step-by-step-tutorial