Advanced Data Analysis and Machine Learning (2023-2024)
Computational Engineering
LUT University

Google Stock Dataset: Final Submission

Alejandro López Vargas
Sina Saeedi
Pau Ventura Rodríguez

December 26, 2023

1. Introduction

Visit our GitHub to gain access to the script.
In this project, we aim to analyze the historical data of Google's (GOOGL) stock price and create a predictive model for it. The dataset includes the following variables: Date, Open, High, Low, Close, Volume, and Name. The 'Name' variable is constant and represents the company's name, which is Google in this case, making it unnecessary for our analysis.
Here are brief explanations about each of the variables:

1. Date: This variable represents the specific dates corresponding to each data point in the dataset. It serves as the timeline for the stock's historical performance.

2. Open: The opening price of the Google stock on a particular date. This is the price at which the stock begins trading at the beginning of the trading day.

3. High: The highest price at which Google stock traded during a given day. It provides insights into the highest level the stock reached within a trading session.

4. Low: The lowest price at which Google stock traded during a given day. It indicates the lowest level the stock reached within a trading session.

5. Close: The closing price of Google stock on a specific date. It is the final price at which the stock is traded for the day, representing the market sentiment at the close of the trading session.

6. Volume: The total number of shares of Google stock traded on a given day. Volume is a crucial indicator of market activity, providing insights into the level of interest and participation in the stock.

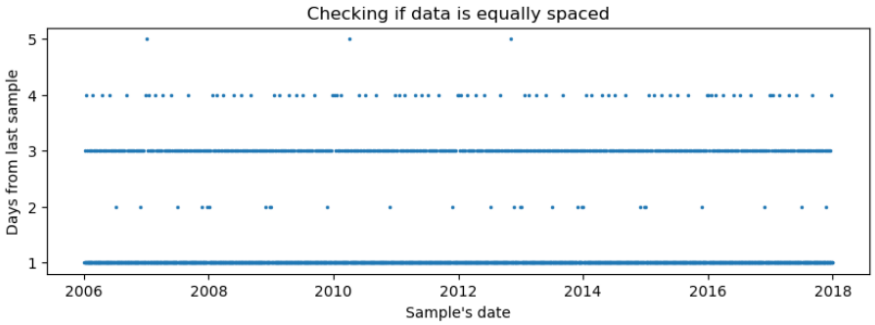
Project's Goal:
The main objective of this project is to develop a predictive model that can anticipate future closing prices (Close) of Google stock based on historical patterns and trends. By drawing insights from the information contained in the Open, High, Low, and Volume variables and previous values for the Close variable, we aim to create a reliable model that can assist investors and traders in making reasonable decisions about Google stock. The investor will then check at the beginning of the day if the closing price will be higher than the opening price. If this is the case, the stock will be bought in the morning and then sold when the day finishes.

2. Materials and methods

2.1. Data Visualization

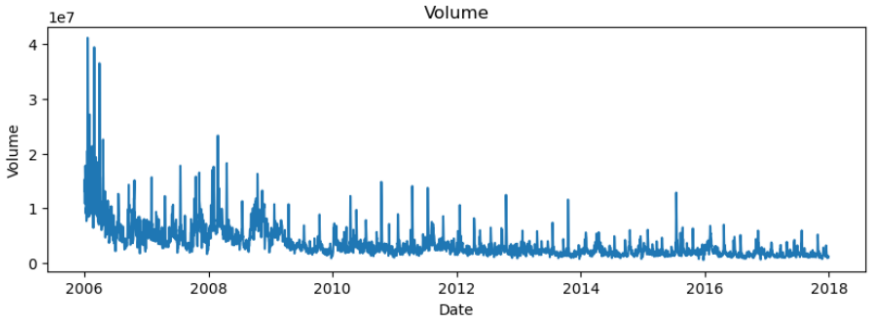
2.1.1. Date Distribution

Another important step is to check if all the data points are equally distanced between them.



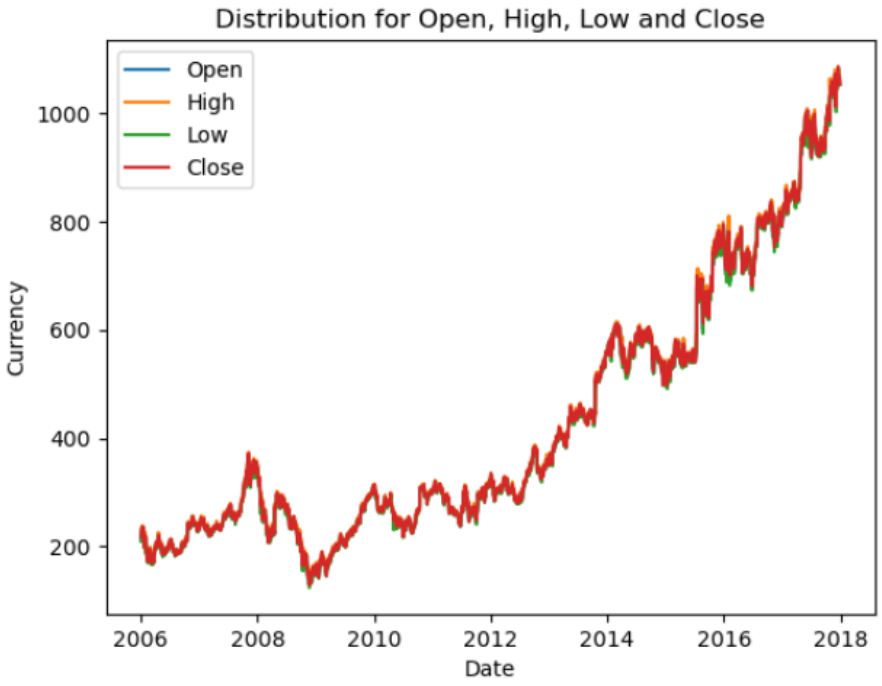
The general case is that two days consecutively have samples assigned. Observing our samples we can see that in general we have samples for 5 days of a week, and then there are two days missing (days from the last sample= 3). This makes sense as the stock market is only open on weekdays, so we do not have data on weekends. However, this is not the only case, as there have been times when the stock market has been closed for 1 day, 3 days, or even 4 days at maximum. This might occur on holidays or dates like Christmas. These anomalies can alter our analysis and we will focus on them lately.

2.1.2. Volume Distribution

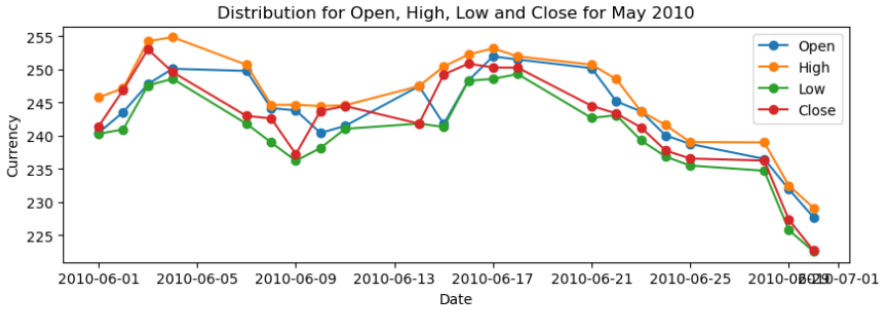


It seems like Google's stock market has lowered its popularity over the years, with some peaks throughout time, but the general tendency seems to have decreased.

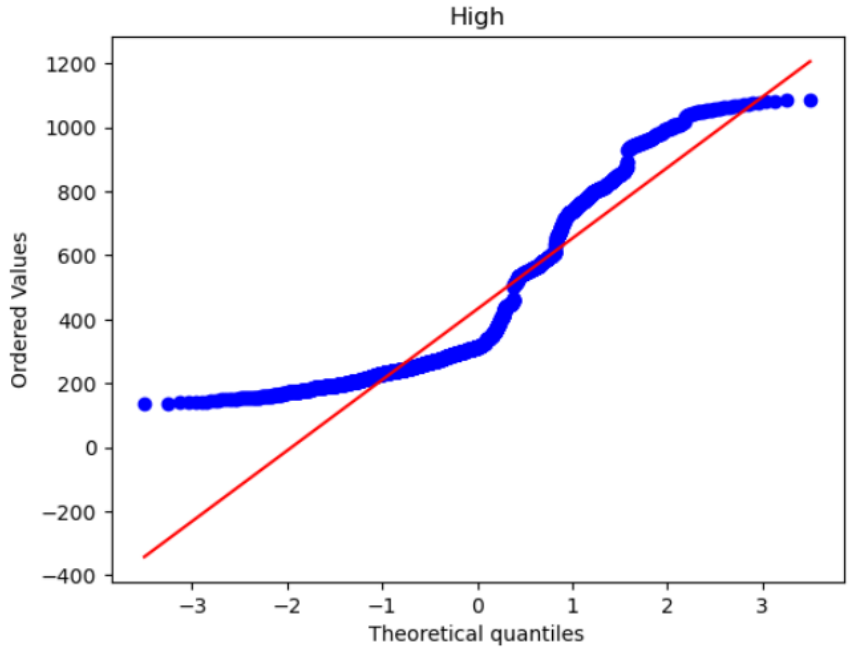
2.1.3. Prices Distribution



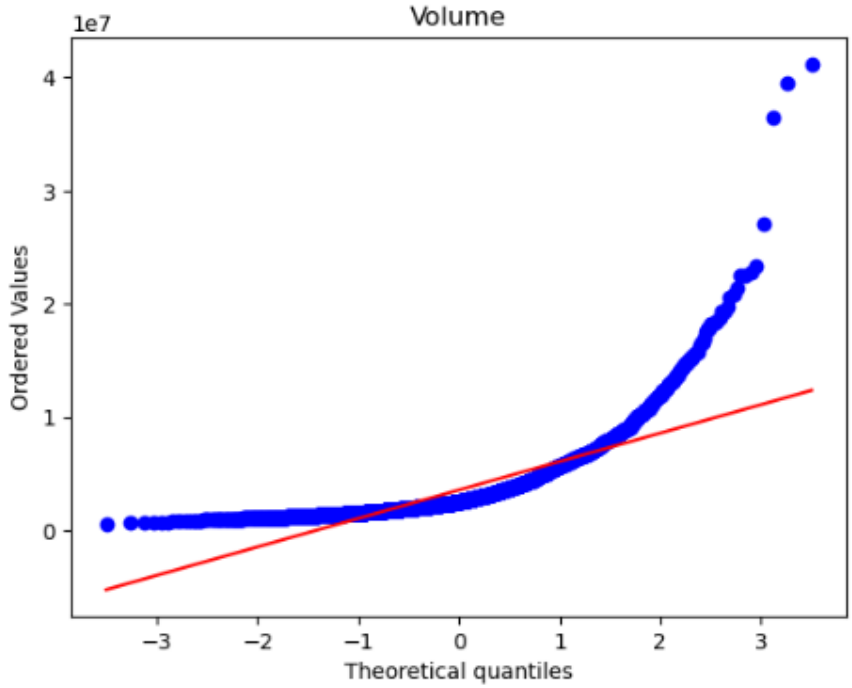
Although the measures are overlapping, we can see the shape and behavior of the prices during the years. It seems like until 2010 it remained on average constant, but after that it started increasing drastically, going from 200 in 2010 to surpassing 1000 in 2018. If we take a deeper look into a short period of time, we can further understand the variables.



As we can see, the High variable works as an upper bound for all other variables, whereas the Low variable works as a lower bound for them.
For further analysis, we will check if our features follow any normal distribution, which is not usually the case for time series. For doing so, we will use a probplot.



All prices variables follow this same probplot, which indicates that for values higher than 600 the distribution is more close to a normal than for values under 600. Anyways, these features do definitely not follow a normal distribution.

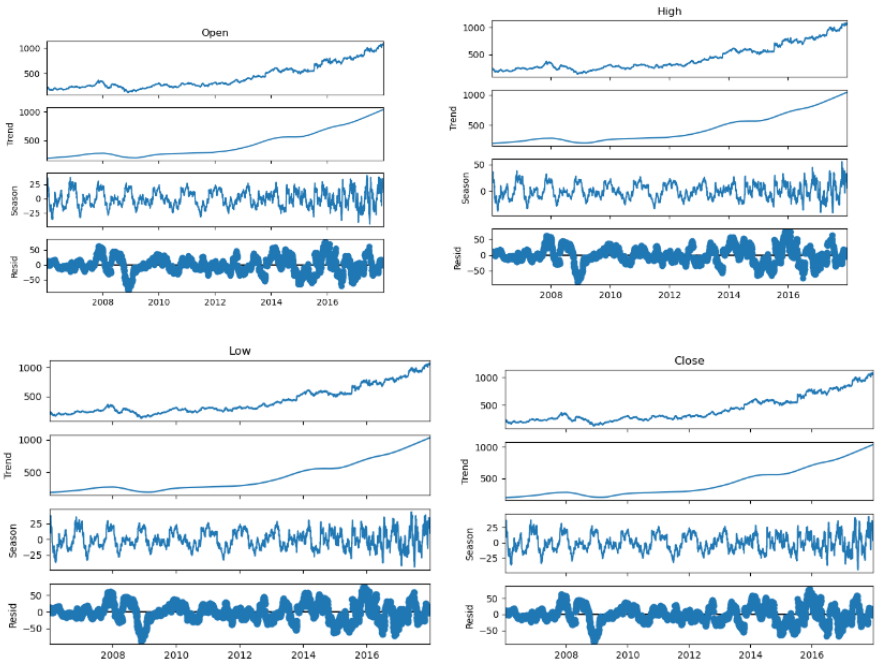


The Volume variable does not as well follow a normal distribution, as most values are

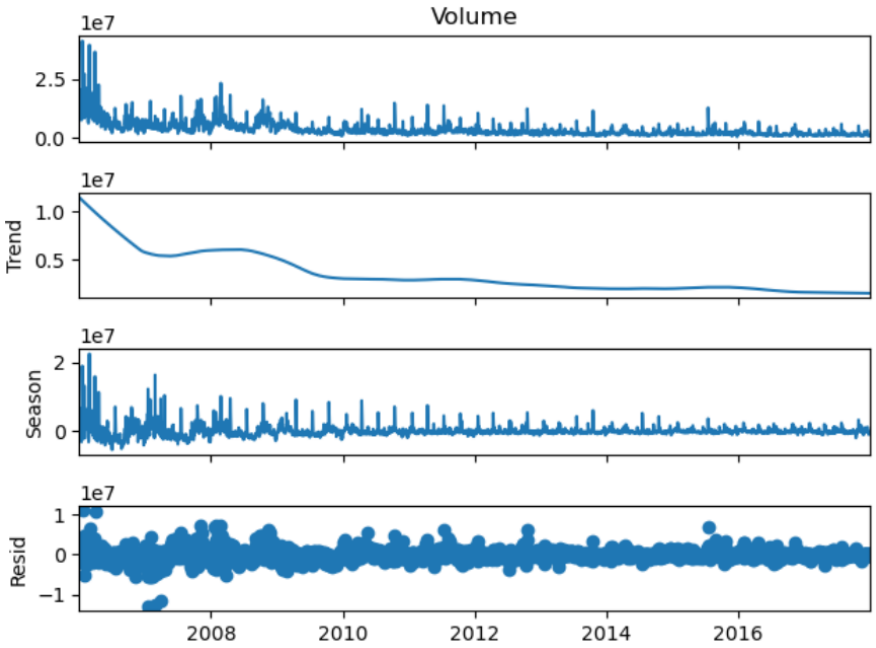
very far from the red line.

2.1.4. Trends

The features open, close, high and low have very similar distributions, so they will also have similar trends and seasonalities as we can see below:



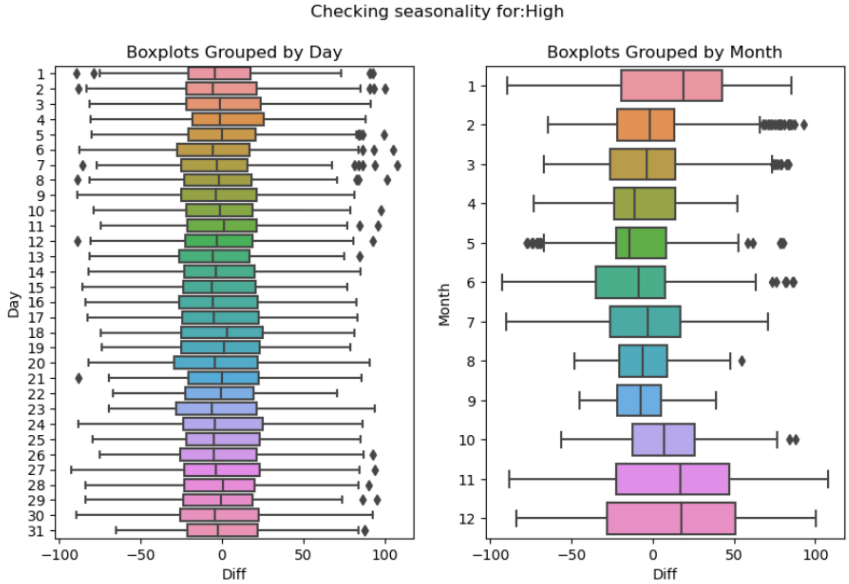
The trend is clearly increasing throughout the years. For the seasonalities, at least until 2014, there seems to be a small peaks followed by a large peak and a big decrease for approximately each year.



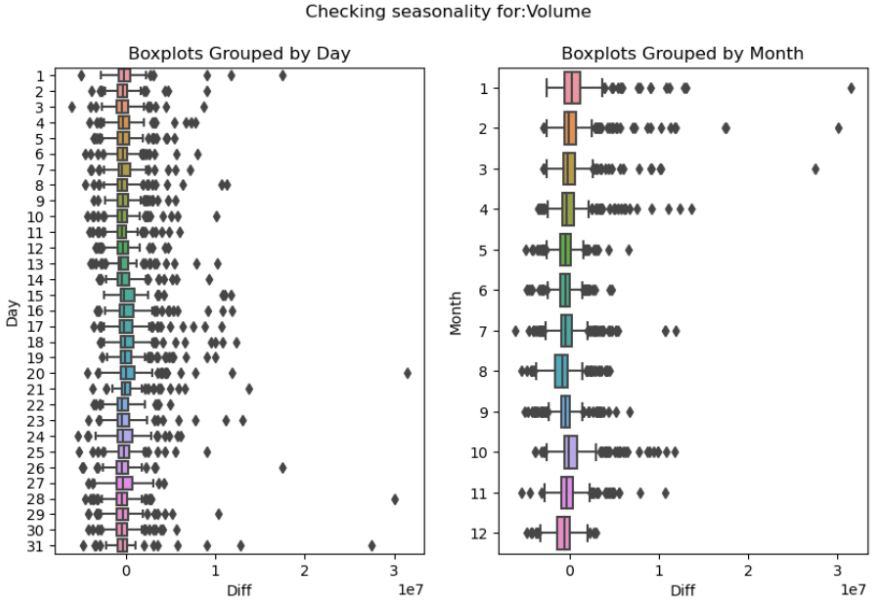
The volume however, looks very noisy, specially on the first years (probably with the boom) the season captures a lot of variance, which we were not able to detect any pattern.

2.1.5. Looking for seasonality

To look for seasonality, we take out the trends for each variable and then group by month and days. The following boxplots then represent the samples distribution for each group.



The days seem to have not a lot of variance, whereas although the months are not completely different between each other, the distributions seem to be different. In winter months (November, December and January) the mean is a bit higher, which indicates that the price tends to be higher on those months overall (then probably a good strategy would be to buy in summer and sell in winter). However, winter months tend to have a higher variance as well.
Other price variables (low, open, close) have a very similar distribution, so we won't comment on them (you can check the code for that).



For the volume, we know that this variable is very noisy, and this can be seen on the amount of outliers. Overall, we cannot take that many insights from their boxplots, as the distributions are very similar among months and days.

2.2. Data pretreatment

1. Data Formatting: Ensure that the date format is the correct one and build an appropriate dataframe indexed by date. Ensure as well that data has the same frequency between observations (in our case we need to have all business days). We have converted the Date column on the dataframe to format Datetime in order to give it an ordinal value. The stock market is only open on business days, which means we do not have data for all days of the year. Furthermore, some holidays did also not register data like Christmas, which we will create in order to have an equally spaced dataframe. Pandas allows us to create these extra samples by using the command: df.asfreq('b'), which will fill their values with NaN.

2. Data Cleaning: Dropping unnecessary columns. In our case the column "Name" only has one unique value, which does not give us any information on the target

and will then be dropped. Check for missing values in the dataset and handle them appropriately. We have missing values as we created them when we added the extra days to keep the frequency. Unfortunately, we have 3 samples that have consecutive missing values. As all those samples consist of only clusters of 2 consecutive samples, we will use Frontfilling on the first instance and Backfilling on the last instance. To fill up NaN values, we can use Backfilling (setting up the values with the next datapoint's values), Frontfilling (setting up the values with the previous datapoint's values), Meanfilling (assigning the mean value of that feature, which in this case will not yield good results due to the increasing tendency). In this case, we will use Backfilling. In addition, check for outliers in the data and consider whether to remove or transform them.

3. Feature Engineering: Create lag features for Open, High, Low, Close, and Volume. However, if we want to predict the Closing price, we will not have the High, Low, Volume nor Closing variables. Instead, what we will have is the values for those variables on the previous timestamp, so we can use on each sample the lagged version of Closing, Low, High, and Volume instead of the original one. This means including previous time steps as features. The number of lags is a hyperparameter that can be tuned. The samples will then be, given current day (cd) and previous day (pd): Open(cd), close(pd), Low(pd), High(pd), Volume(pd). So we will try to predict next sample's Close price (which will be the current day's closing price).

4. Train, Test, and Validation Split: Divide the dataset into training, testing, and validation sets, ensuring that the test and validation data capture the trend and seasonality patterns present in the training data. Employ a split ratio of (80-10-10) for the dataset allocation. The training split will be used to train our model. The validation will be used to know when to stop training the model and avoiding overfitting. Finally, the test split will be used to measure the performance of the model on unseen data. We can also perform cross validation to ensure that our data partitions are not biased and our performance results are as rigorous as possible. The type of cross validation used in that case would be Time Series Cross-Validation, which divides the training set into two folds at each iteration on condition that the validation set is always ahead of the training set

5. Normalization: It's common to normalize the data, especially when using neural networks, to improve convergence and training speed. For our analysis, we will use the Z-score normalization method on train data and with the mean and standard deviation that we found from the train data, we will normalize the test and validation data.

2.3. Evaluating Strategy

For evaluation, several steps can be done which are explained briefly:

2.3.1. Performance Metrics

In the following equations, Y is the real value and hatY is the predicted value while "n" is the number of total samples.

- Mean Squared Error (MSE): measures the average squared difference between the predicted values and the actual values giving us the overall accuracy of the model.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Mean Absolute Error (MAE): the average absolute difference between predicted and actual values. It is less sensitive to outliers.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

2.3.2. Visual Inspection

- Time Series Plot: Plot the actual vs. predicted values over time to visually inspect the model's performance.

2.3.3. Hyperparameter Tuning

- Assess the impact of changes in hyperparameters on model performance.
- Conduct sensitivity analysis to identify the most influential hyperparameters.

2.3.4. Comparing the Models (RNN, LSTM, and Transformer) with Each Other

- Evaluating the performance metrics of all models in comparison with each other to determine the most suitable model for predicting our stock time series.

3. Models and results

3.1. Model 1: Simple RNN

Recurrent Neural Networks (RNNs) are a type of neural network architecture designed for sequence data, making them well-suited for tasks such as natural language proces-

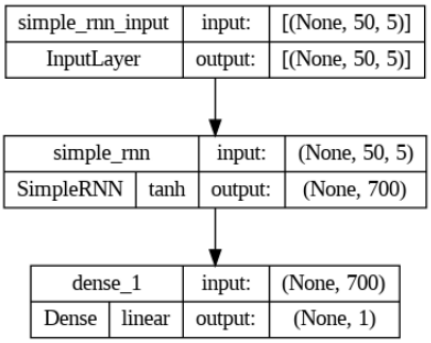
sing, speech recognition, and time series analysis. Unlike traditional feedforward neural networks, RNNs have connections that form a directed cycle, allowing them to maintain a hidden state that captures information about previous inputs in the sequence. Here's a brief overview of the main components of a simple RNN:

1. Hidden State: The hidden state at time step "tïs a representation of the network's memory or context, capturing information from previous time steps.

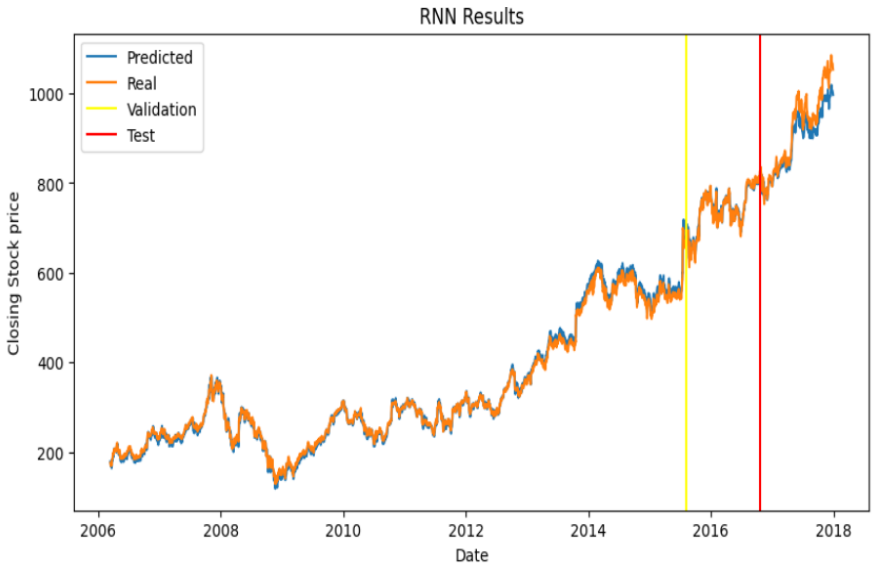
2. Input: The input at time step "trepresents the current element in the input sequence.

3. Output: The output at time step "tïs the prediction or representation generated by the network based on the current input and the hidden state.

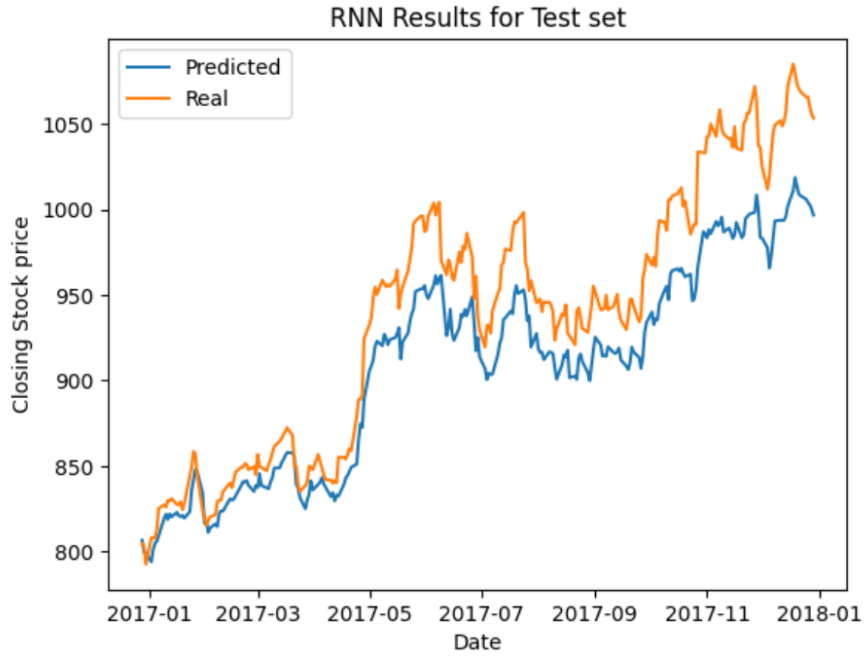
In our scenario, our goal is to forecast the Close price of Google stock using the RNN model. Following the required preprocessing steps and the addition of lagged features, we establish our simple RNN model consisting of only one simple recurrent layer. The number of neurons in the hidden layer is set at 700 which is a parameter that can significantly impact the model's accuracy. Another crucial parameter is the learning rate, which is chosen as 0.0005. The final architecture for our RNN model is:



A rather simple architecture yielding decent results, following the Ockham's razor principle. Our neural network only contains a simple RNN cell that takes 50 lagged samples of 5 features each, and outputs 700 neurons using an hyperbolic tangent activation function. Those 700 neurons then go through a final dense layer and using a linear function they yield a single value corresponding to our model's prediction. In the following figure, we can observe the performance of our RNN model for train, test, and validation data.



The model exhibited satisfactory performance on the training and validation datasets. In addition, Initially, it performed well on the test data, but over time, it struggled to closely track the target Close price. The following plot provides a more detailed visualization of the test data predictions.



Furthermore, for the performance matrices, we have chosen RMSE and MAE for this model.

```
Test RMSE: 34.73758067021133
Test MAE: 29.6833082075808
Validation RMSE: 8.957153526603104
Validation MAE: 6.573888052553414
Train RMSE: 8.79016334167736
Train MAE: 6.854271679404635
```

Examining the matrices, it's clear that the model performed well during both training and validation. However, the model faced some difficulties when attempting to predict the test data. Yet, the satisfaction level with the Root Mean Square Error (RMSE) could vary depending on our particular criteria for accuracy and precision in the model's predictions.

3.1.1. Sensitivity Analysis for RNN

For the sensitivity analysis of the RNN model, the number of neurons in the hidden layer is considered. The following table shows the result of the RMSE of the model for different numbers of neurons:

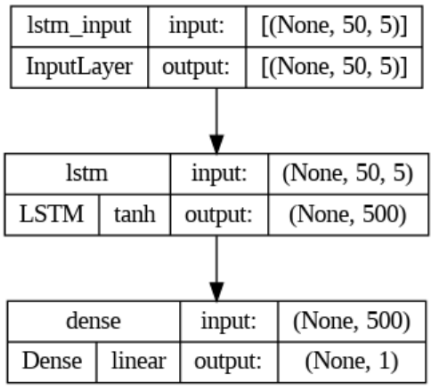
	Hidden Neurons	Test RMSE	Val RMSE	Train RMSE
0	500	46.741974	9.228915	11.037960
1	600	40.924871	9.135053	10.340989
2	700	49.583507	9.418348	10.040602
3	800	41.108717	9.582819	11.534445

We can observe that the best RMSE result for the test set is achieved for the number of neurons equal to 600 even though the results were quite close to each other for 600 and 800.
Also, regarding the execution time, it can be stated that the RNN model is approximately 4 to 5 times faster than the LSTM model, as would be naturally anticipated.

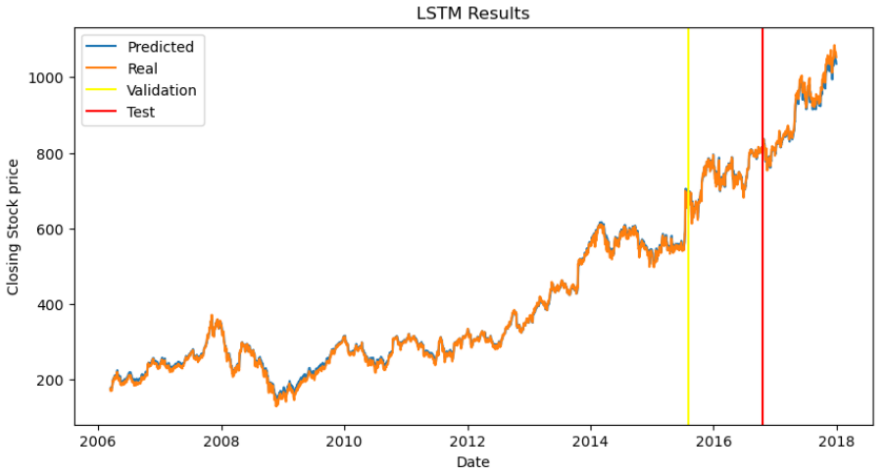
3.2. Model 2: LSTM

To train an LSTM model, the data is required to be Normalized and properly processed. Normalization has been done by training the scaler with the training set (80 % of the dataset), which was then applied to the validation set (10 %) and the testing set (10 %). LSTM needs the data to be fed in the shape [n_training_samples, n_timesteps_per_sample, n_features]. In our case, after trying different timesteps, we have decided to add 50 timesteps into our model, which results in information about 50 days before the current day. Also, keep in mind that we had to include the current day's Open feature, so we had to drop last day's Opening feature. Although including 50 lagged variables drastically increases execution time, it adds a good amount of information to reach good predictions and, considering that each sample only has 5 features, adding 50 lagged variables only

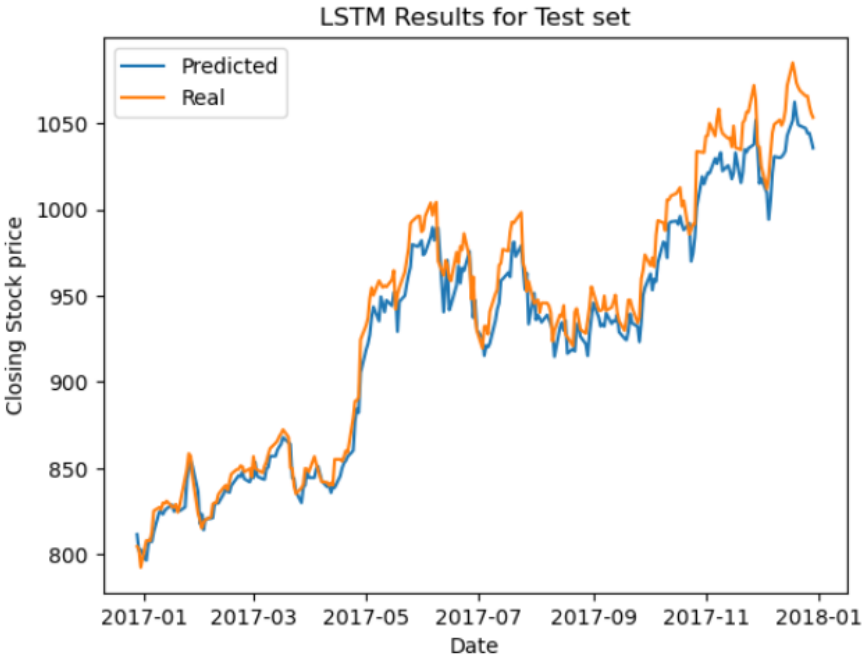
results in having 250 more features (which for a Machine Learning model is not that much). The chosen prediction window was 1 day, as we will simulate a short-period trader who wants to know if it is worth buying stock when the day opens. After the samples were reshaped, they were ready to be fed into the model. The parameters that needed to be considered are: the number of neurons on the Hidden layer inside LSTM, the batch size, the number of epochs and the learning rate. The number of epochs was set into an arbitrary number (100) and then EarlyStopping was applied. This would allow the model to avoid overfitting and not rely on the number of epochs trained. After trying different combinations of neurons in the Hidden layer, we decided to set the number to 500, as our data has many underlying patterns and it's hard to model. Same for the learning rate, after different trials and errors, we decided to set it up to 0.0005 to ensure a slow but converging phase. The batch size was set to 32, as larger batch sizes result in faster training times, but less accuracy and even overfitting. In the model, we also added a Dense layer at the end to combine the output of the LSTM into the final output without any activation function. When compiling the model, we have used MSE as the loss function and the Adam Optimizer. The final architecture for LSTM is:



This very simple architecture yields very interesting results, as we only have a LSTM cell that inputs 50 lagged variables of 5 features each and yields 500 cells using a hyperbolic tangent activation function. Then those neurons go through a dense layer of a linear activation ending up in 1 only cell containing the regression prediction. Training the LSTM model took almost 1 minute, which is a decent value considering that we have 2500 training samples and 250 features. The results of training, validation, and testing data are the following:



The model seems to have learned properly the trends and patterns of our data. On the training data specially, the prediction closely fits the real values, achieving a MAE of 5.70 and a RMSE of 7.34, which means that on average, we will fail the closing prediction in 5.7 units. On the validation set, the prediction is not as good as in the training set, but still closely matches the real value, achieving a MAE of 6.00 and a RMSE of 8.17, which means that on average we will fail the closing prediction in 6 units. On the testing set, the prediction is the worst of the three, as the model did not see the data and was not trained depending on it. Still, as we can see on the chart it performed quite good, with a MAE of 11.63 and a RMSE of 14.10. Here we can see a zoomed version of the testing set to avoid being mislead by the large scale of the dataset:



The model seems to have a good understanding of the patterns, as the predicted and real values have a very similar behavior. Although the predictions seem to overestimate the real values most of the time.

3.2.1. Sensitivity Analysis for LSTM

For sensitivity analysis, two parameters of learning rate and number of hidden layer neurons are chosen. For each of these parameters 3 different values have been considered and the RMSE result of the model is evaluated as follows:

	Learning Rate	Hidden Neurons	Test RMSE	Val RMSE	Train RMSE
0	0.0010	300	24.278524	8.443466	7.311486
1	0.0010	500	21.478543	8.418149	6.451053
2	0.0010	700	8.660410	9.315286	7.872874
3	0.0005	300	11.230527	8.120198	7.989817
4	0.0005	500	14.706670	8.090913	5.701423
5	0.0005	700	16.600249	8.143188	6.486614
6	0.0001	300	18.606965	12.565943	9.622439
7	0.0001	500	11.566314	12.517367	8.576187

As we can see, the best performance (based on the RMSE of the Test) is achieved for a learning rate of 0.001 and hidden layer neurons of 700. However, by increasing the number of hidden layer neurons, the run time of the code increases as well.

3.3. Model 3: Transformer

Transformers, a groundbreaking machine learning architecture, revolutionized time series analysis by introducing parallel processing through self-attention mechanisms. Originally designed for natural language tasks, Transformers excel in capturing temporal dependencies, making them ideal for modeling complex patterns in sequential data. This architecture, introduced by Vaswani et al. in 2017, has transformed the landscape of sequential data processing, offering remarkable capabilities in tasks such as forecasting, anomaly detection, and pattern recognition. To train a Transformer model and ensure that it converges, the data requires to be Normalized and properly processed. Normalization has been done by training the scaler with the training set (80 % of the dataset), which was then applied to the validation set (10 %) and the testing set (10 %). The transformer model used has been adapted from the template provided in the Transformer's exercise, as some changes have been made. For example, in this case, we do not want to predict all the features but only the Closing price. Also, the normalization has been fixed to make sure that the statistics are only calculated using the training set. In terms of features, we are using 50 lagged variables to make sure that the model is able to capture the most amount of information (and as well because that was the optimal number for LSTM). After the samples were reshaped and sequenced, they were ready to be fed into the model. The parameters that needed to be considered are:

1. Learning Rate: Dictates the phase at which the model will learn. A learning rate too large might cause the model to not converge, and a Learning rate too small might cause the model to take too much time to converge.

2. Number of epochs: Backpropagation uses Gradient Descent algorithm to find the optimum value of the weights. This involves making a certain number of steps, which might not be reached only going through all the data once. Thus, the number of epoch is the number of times the data will be sent to train the model.

3. Batch size: The data might be too large to be handled all at the same time. For this reason, DataLoaders are used, which contains a certain amount of data on each batch.

4. Number of Layers: Determines the number of layers of type transformer inside the model.

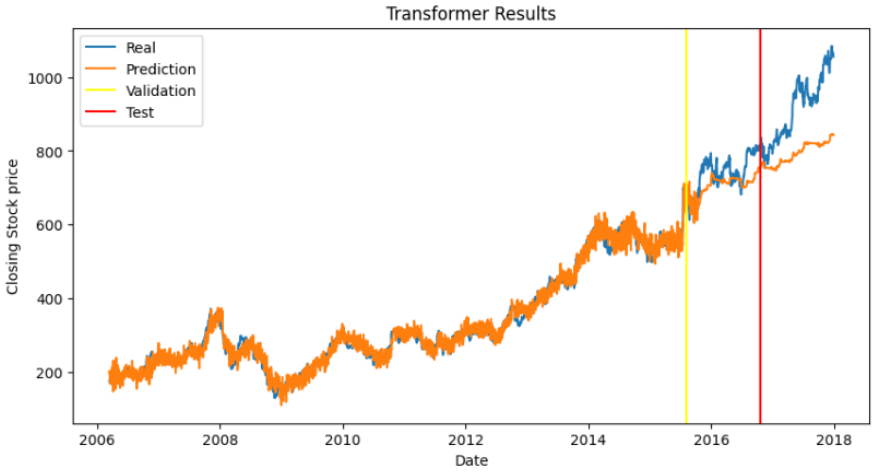
5. D: Represents the dimensionality of the model, also known as the dimensionality of the hidden state.

6. H: Number of attention heads insidethe transformer.

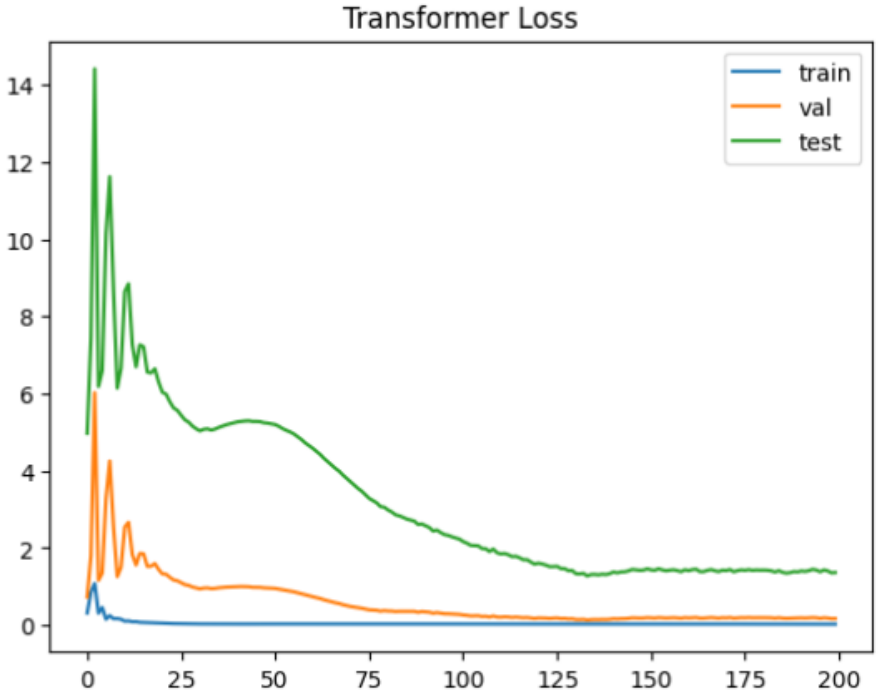
7. Hidden mlp dim: Dimensionality of the Hidden Multilayer Perceptron inside the transformer layer.

8. Dropout: Defines the percentage of neurons that are removed from the architecture to ensure the network does not overfit.

Epoch, Learning Rate, Batch size and Dropout have shown to have the least impact on the model, so we have chosen to determine a default value of 300 epochs, 0.00001 learning rate, 32 batch size and 0.1 dropout. This was made to focus on the most impactful parameters. Those parameters then have been deeply analyzed along different ranges of values. Trying all the possible combinations of these parameters and hyperparameters on the model would be a time-consuming task. To solve this problem, we will try random combinations of the values, the so-called Random Grid Search and compare the performances of the models. As mentioned, we have tried many ranges of values for our hyperparameters to check which ones yielded the best model's performance. The possible values include:
num_layers = [1,2,3,4]
D = [32, 32*2, 32*4, 32*8]
H = [1, 2, 4, 6]
hidden_mlp_dim = [32*3, 32*6, 32 * 8, 32*10]
The metric used to check which hyperparameter combinations were better was the validation loss. That model that yielded the lowest loss on the last epoch was set to have the best hyperparameter combination. The random grid search was performed using 50 iterations. More iterations and broader searching space was not possible due to execution time limitations (this hyperparameter search already took several hours). The best hyperparameter found are: Number of layers: 1, D: 256, H:4, Hidden layer dim: 96. Training the transformer model took almost 2 minutes. The results on training, validation and testing data are the following:

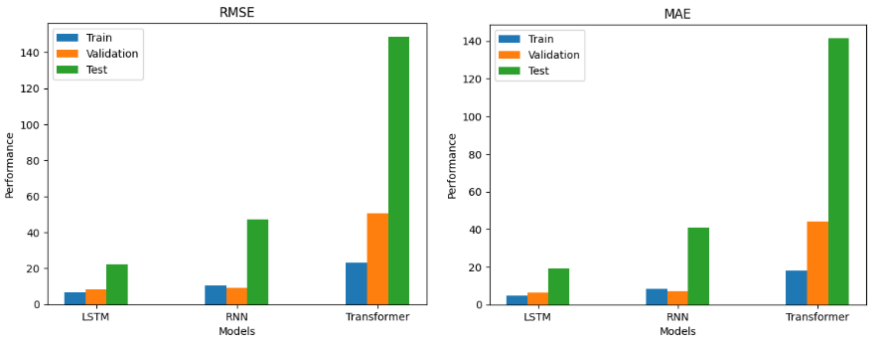


As it can be seen, the Transformer model did not perform properly, as the predictions are really far away from the real values. Even in the training data the predictions have a lot of variation and do not fully fit to the real values. The validation and testing split also do not properly converge to the real value, and mostly yield really bad underestimations.

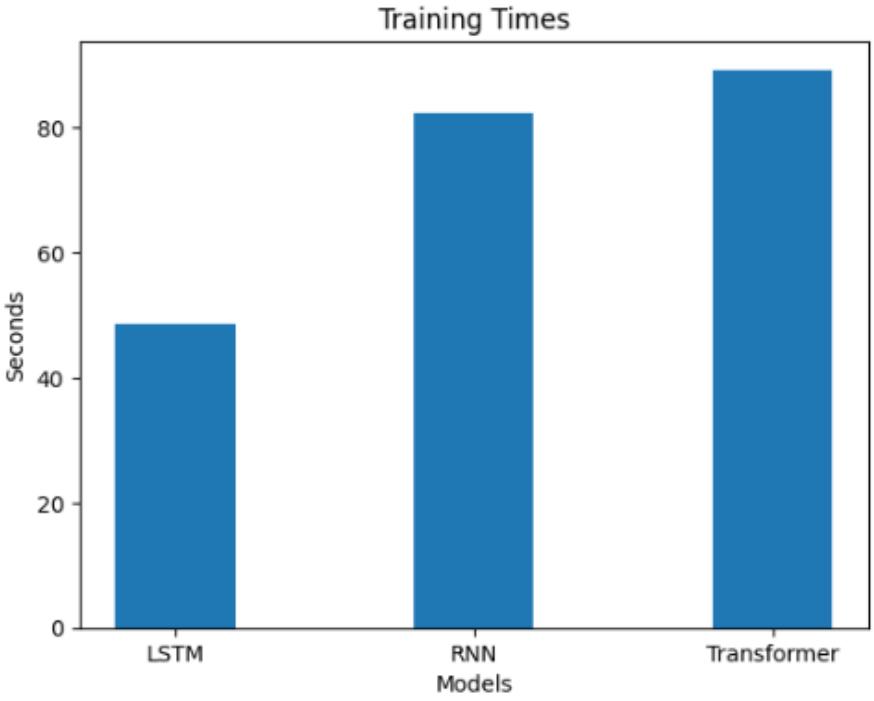


It can be seen from the loss that the test partition is very far away from the validation and training losses. This might be an indicator of overfitting to the data. However, increasing the dropout rate does not help improving this effect. Why did our transformer not perform properly? Although we have made a decent hyperparameter search, none of them seem to have yielded a good convergence. We have many hypotheses about that fact. First, our model took so long to train, which has extremely limited our capability to test more hyperparameters combinations. This high execution time can probably be attributed to the fact that the code is almost all programmed in python instead of being an implementation that calls to a function executed in low level programming language (such as C). Although on a single iteration this might not have a big impact, on multiple iterations the difference grows higher. Another possible explanation is that we are blindly trusting this provided architecture, but it might have a slight error that completely avoids convergence although the code runs completely properly. Another option is that we might have some misunderstanding of how the transformer works and how the inputs must be given, and thus the model is not able to perform as it should. Lastly, maybe a transformer is just not optimal to forecast our dataset and is outperformed by other sequential models.

3.4. Models comparison



Using both metrics to compare the models, we can conclude that the best model is LSTM, followed closely by RNN. The worst model has shown to be Transformer by far. Although all models have much higher testing error than training and validation, the LSTM seems to be the one that overfits the less and thus achieving the minimum error on unseen data. Note that, given the results, using LSTM ensures an average absolute error of around 20 on each prediction.



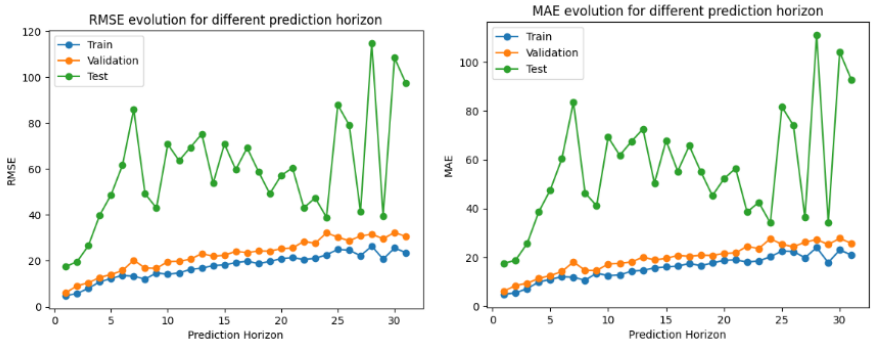
If we compare the training time, LSTM seems to still yield the best results, with a training time of 50 seconds, being the lowest of the three models, followed by RNN and Transformer.

Based on the metrics and different tests done, the best performing model seems to be LSTM by far, achieving an average error of around 20 dollars on the stock price, while keeping the lowest training time among all other models. Although the transformer is the most complex model and the one that takes the longest training time, it is the one that yielded the worst results. Probably a demonstration of the Ockham's Razor principle, sometimes the simplest models yield the best results and we do not have to overcomplicate the architectures. Overall, the predictions from LSTM are really good, but is it enough? Well, considering that in the validation set the average difference between the Opening and Closing values is 0.28, we could set Closing = Opening + 0.28 and we get a MAE of 5.83, which is much lower than our current MAE. So a mean regressor with only one variable (Opening) performs much better than our model. Getting really close to our model's performance on the training set.

3.5. Forecasting power

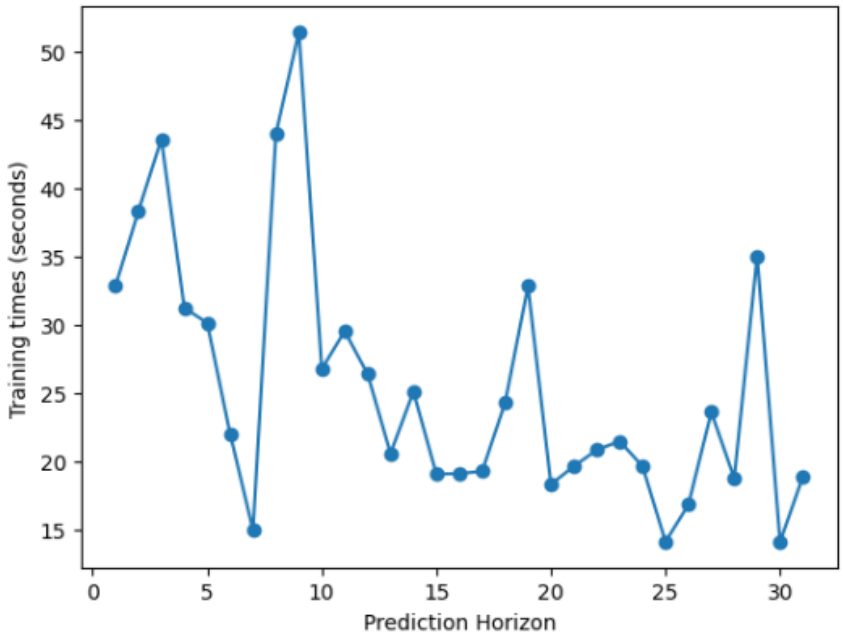
3.5.1. LSTM

We have tried building different LSTM models varying the predicting horizon, which means predicting multiple time steps ahead instead of the following closing value. This can be very useful in our practice case, where we want to know what the closing value will be in a future to know if we should buy or not stock. The modifications applied to the LSTM architecture have consisted in increasing the number of output cells on the last Dense layer of the architecture to match the prediction horizon. As plotting the single sample results is harder in this scenario, we will be only showing the MAE and RMSE metrics calculated when we made the predicting horizon vary.



As expected, as the prediction horizon grows higher, the errors increase. Although validation and testing splits' errors have smooth growth (probably because training was based on those samples), testing splits' errors have very volatile error curves, which in most of the cases are far from the validation and testing errors. First the testing curve grows until reaching a peak when the prediction horizon is of an exact week. However, it drastically falls and maintains a slow decrease after growing again. The next minimum is found when we are predicting 24 days ahead. Then it has a considerably high variance, as the error varies between 35 and 110 MAE. The best value then to ensure a consistent

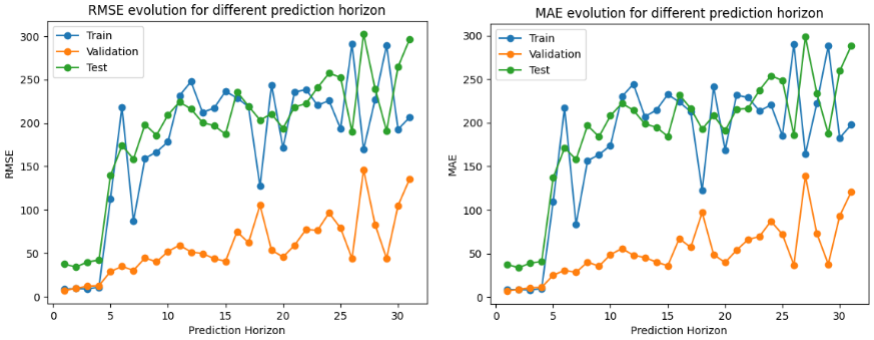
LSTM estimator would be around 24, as it has an estimated error of 40 units on average. If we compare the execution times of all models:



We can see that as the prediction horizon grows higher, the execution time decreases. This might be because the loss value is higher and thus the converging point is easier to find. Using this graph we wanted to check if the model's training time increased as the predicting horizon increased, to include it in the equation when considering if it is worth it to build a model that predicts a far time step in the future.

3.5.2. RNN

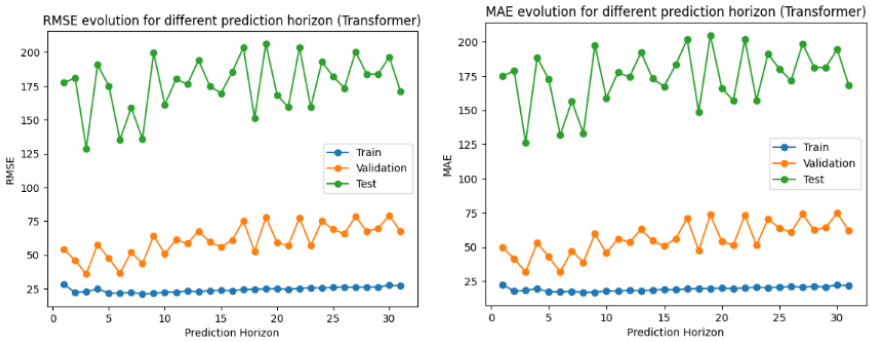
For the RNN we have followed the same process, and the charts yielded some interesting results:



A very strange behavior can be seen here, as the training error on the first 4th datapoints is very low, matching the validation error, but after that, it drastically grows to be at the testing split's error scale. This might be explained by the fact that the model is required to be too complex to represent that many output variables and struggles to converge, although the validation split keeps the metrics much lower. Taking a look only at the testing split, the errors are much worse than the ones seen at LSTM. In this case we can see that the error stays more or less constant after the 5th days' prediction around the 200 MAE , which is really high compared with the 40 of the LSTM.

3.5.3. Transformer

For the Transformer we have followed the same process, and as the model that we built before highly unperformed, we did not have high hopes on increasing the prediction horizon.

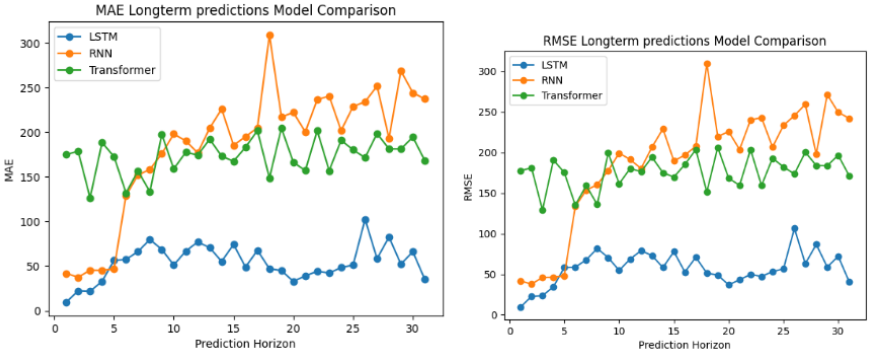


As expected, although the error does not increase and maintains constant as the prediction horizon increases, which is a good feature, the error is really high on all sets, specially on the testing set, having mean absolute errors of around 150, which is very high compared with the 40 of the LSTM.

Unlike other models, training a transformer with higher prediction horizon significantly decreases the model's execution time, going from 63 to 50.

3.5.4. Comparisons

If we compare all the models on the same graph for the testing split, we have the following results:



Comparing the models, on the first 5 values for the prediction horizon the RNN and LSTM have similar results, with the LSTM slightly outperforming RNN on the first values by almost 40 units, and ending up being surpassed by RNN only at the value 5. Then RNN highly decreases the performance, reaching 150 MAE which is the same as

the Transformer model, and then it keeps increasing the error as the prediction horizon increases, while transformer and LSTM keep their error more or less constant around 150 and 40 respectively. If we want to make long term predictions we should in general be leaning towards the LSTM model, except in the case of a prediction horizon valued 5, in which case we would use RNN.

4. Limitations and Strengths of the Models

After running the parts of the evaluation, now we can talk more about the pros and cons of each method.

4.1. RNN

Limitations:

- Vanishing and Exploding Gradients: Difficulty capturing long-term dependencies due to gradient issues.
- Short-Term Memory: Struggles to retain information over extended sequences.

Strengths:

- Flexibility in Sequence Modeling: Perfect for tasks where the order of input elements matters.
- Ease of Interpretation: Relatively simpler and easier to interpret.
- Run Time: Compared to other 2 methods, it is much faster.

4.2. LSTM

Limitations:

- Computational Complexity: More computationally expensive compared to standard RNNs.
- Potential Overfitting: Prone to overfitting, especially with smaller datasets.

Strengths:

- Long-Term Dependencies: Effectively captures long-term dependencies in sequences.
- Effective Handling of Sequential Data: Excellent for tasks involving sequential data.
- Gating Mechanisms: Uses gating mechanisms to selectively update and forget information

4.3. Transformer

Limitations:

- Complexity and High Run Time: Can be complex and challenging to implement with usually higher execution time.
- Interpretability: Challenging to interpret, impacting model explainability.
- Data Efficiency: It requires substantial data. Performance may suffer on smaller datasets.

Strengths:

- Attention Mechanism: Captures long-range dependencies for effective global context understanding.
- Parallelization: Enables efficient parallel processing, leading to faster training.
- Flexibility: Adaptable to diverse tasks beyond its original design for NLP

5. What can be improved?

To improve the performance of the prediction, we can consider several options as follows:

1. Using different models such as SARIMA(Seasonal Autoregressive Integrated Moving Average), Xgboost, and pre-trained models for transfer learning.
2. Optimizing the hyperparameters and finding the optimal sequence length.
3. Train the model using a sliding window approach where the model is trained on multiple subsets of data to capture different market conditions.
4. Scheduling the learning rate or control it some other way.
5. Considering different architectures of LSTM and RNN.
6. Use different activation functions if possible to change the type of nonlinearity of the model.

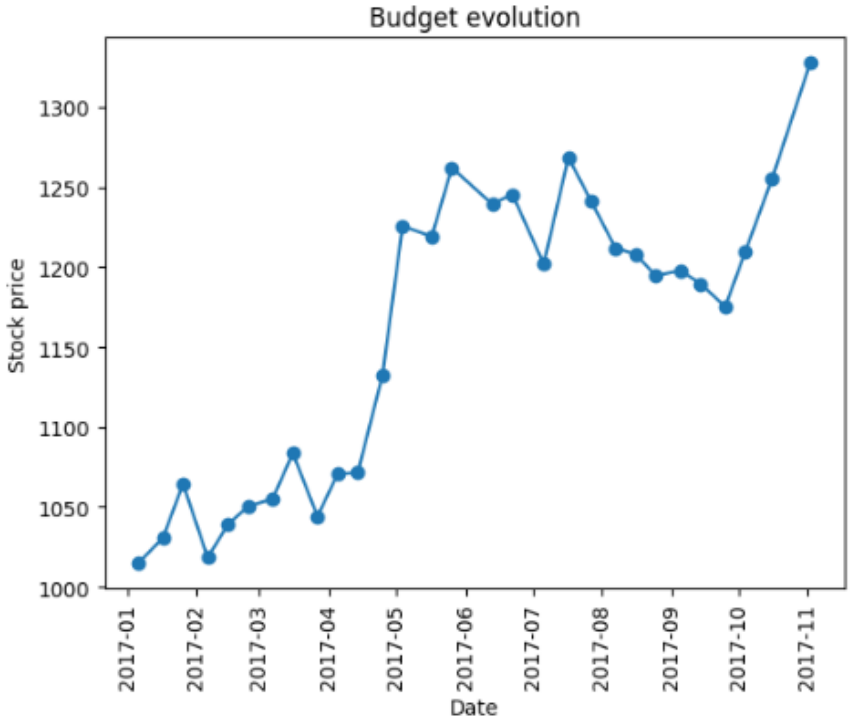
6. Are we becoming millionaires?

To further test our models' performance, we have decided to make a simulation of investment using our best model, the LSTM, and comparing it with raw investment. We have trained our model to predict 23 business days (i.e. a full month) into the future with a training split, and validated it with a validation split. The leftover data is from January 1st 2017 to January 1st 2018, so this period of time is where the simulation will take part. To check if our model will help us gain extra return, we will compare it with

a raw investment technique. The initial budget for both cases will be 1000€. The two compared techniques will then be:
Raw investment technique: We will buy the stock the first day at the opening (1st January 2017) and we will only sell it the last day (1st January 2018).
LSTM supported technique: With the built model, at current time-step we will make a prediction of the closing price for the next 23 days. Then, we will check if the maximum closing value on those 23 days is higher than the current opening value. If this is not the case, we will update the time-step to the following day and start again the process. Otherwise, we will buy stock at the current time-step and sell it at that time-step that has the highest predicted value regardless of the real price on that day. Then, we will update the current time-step to that day where we sold the stock and repeat the process.

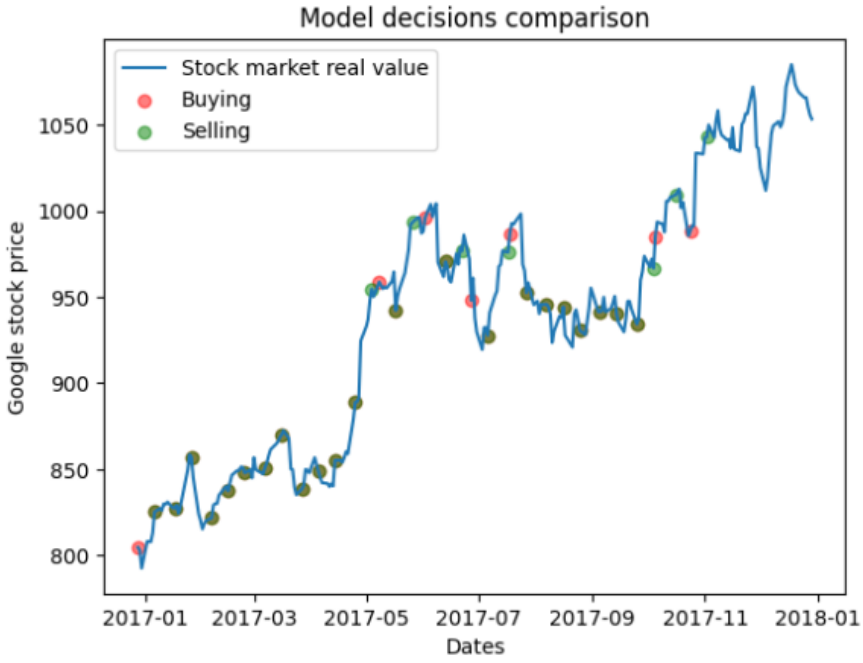
6.1. Results

The results are as follows: For the raw investment technique, the final budget was 1298.60, which means an increase of 298.6€. For the LSTM supported technique the final budget was of 1327.69€, an increase of 327.69€, which means that the decisions made based on our model helped us earn an extra 29.09€. In the following graph we can see the progression of our budget through time:



Most of the time the budget grows, although there is a period of around 2 months in which the model makes wrong decisions. On the following graph we can see the decisions

made by the model, when it decided to buy and when it decided to sell. We can also see that around that period of time where we lost money, the model seems to have predicted that we would have a good increase, but we didn't. Note as well that in some points the selling and buying point do not match. This is because when the model sold the stock, it predicted that in the next 23 days it would not be higher than in that time step.



The success rate of our decisions, which we have defined as the amount of times that our model sold stock when the price was higher than the current price, is 62.06 %, which is higher than 50 %.
In conclusion, even with a really basic investing strategy we have achieved better results than with a raw investing strategy. We could improve our strategy to update our model so that as we get real data from the current time, the predictions for the predicted stock price get updated and we can change our decision based on that. We could also test different predicting horizons instead of 23, ... But we have shown that building a model helps improve the investments done.

7. Conclusions

We have been able to build consistent models that yielded decent predictions for our data although it was very hard to model due to the volatility that it has. The best model has been shown to be LSTM, reaching a highly outperforming all other tested models, which tends to have an error of around 11 units. Also LSTM ensures a decent estimation over a long period of time, with an error of around 35 in predicting horizons of over 20 days.