Team Matrix -

Stock Price Prediction

December 2020

HARSH DESAI
PARASURAM VISWANATH JASTY
POOJITA TADEPALLI

ABSTRACT:

Our report outlines the business use case, architecture, data sources, modelling techniques used for our project. Our project helps in predicting the closing value of stock using Linear Regression, Decision tree and deep learning neural network LSTM. This helps our users to make informed decision making while buying or selling a stock. The process flow of the project is discussed in this report.

Table of Contents

Introduction:

There are thousands of stocks traded on the market. We focused of S&P 500 to make our prediction. Because these are top 500 companies and there is a less chance the buyers will substantially lose money. The risk is less compared to others and everyone in the stock market will aim for higher profits People want to buy stocks and earn money. This is very risky because stock prices fluctuate up and down. Selling and buying the stock at right time will substantially increase profit. Our aim to predict the closing price of the stock so buyer will have a clarity if the stock is going up or down. Different variables are considered in predicting a stock price these variables will help us to increase accuracy of the model.

Matrix:

Our team leverages various data science techniques to predict the closing price of stocks for the companies listed on S&P 500. We predict the closing price of the stock by using historical price of the stock. The different machine learning techniques we used are Linear Regression, Decision Tree and Long Short-Term Memory.

Data cleaning:

	Total	Percent
date	0	0.000000
close	0	0.000000
volume	0	0.000000
Name	0	0.000000
high	8	0.001292
low	8	0.001292
open	11	0.001777

The above image projects the number of null values in each column. As we can see Total number of columns with missing values is 8-27 which is less than 0.005% of the dataset. We could see missing values in high, low, open values in the data. To avoid any issues since it is a time series analysis, we will forward fill the null values instead of dropping them. This is if the values of the day are null, we are taking the previous day values.

Data Sources:

The data sources which we are primarily using are the historical stock prices data (5 years past data) of all companies currently found on S&P 500 index

The S&P 500 or simply the S&P is a stock market index that measures the stock performance of 500 large companies listed on stock exchanges in the United States. It is a capitalization-weighted index (is a stock market index whose components are weighted according to the total market value of their outstanding shares.) and the 10 largest companies in the index account for 26% of the market capitalization of the index.

1. It contains the historical stock price data of all the companies found on S&P 500 index in the file all_stocks_5yr.csv in a merged form.
2. This data file contains 62k rows of data with the columns: date, open, high, low, close, volume, Name of the company from dated Aug 2013 to July 2018.
3. It contains the individual stock ticker historical data (5 years past data) of the companies found on S&P 500 index.
4. It has data from Aug 2013 to June 2019 with the columns: date, open, high, low, close, volume, Name of the stock ticker.
5. All the files have the following columns:
   Date - in format: yy-mm-dd
6. Open - price of the stock at market open (this is NYSE data so all in USD)
7. High - Highest price reached in the day
8. 7.Low Close - Lowest price reached in the day
9. Volume - Number of shares traded
10. Name - the stock's ticker name

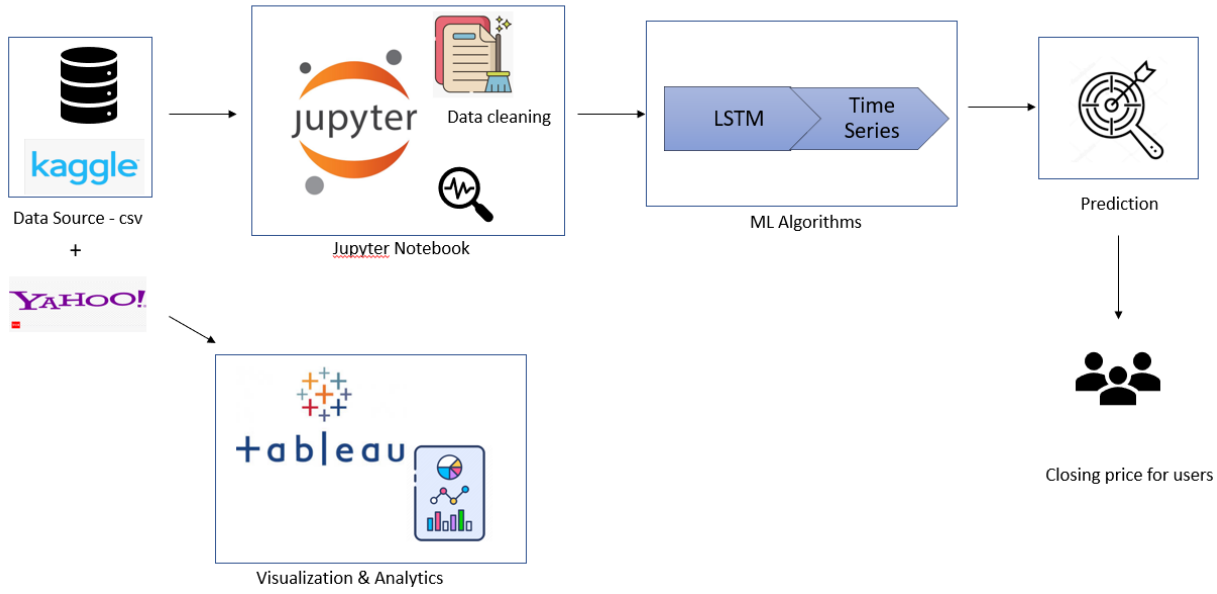
We have captured real-time data using Yahoo API to better understand the variations in the stock at present. This gave us a clear understanding on how the price of a stock varies.

Solution:

Here, we are predicting the closing price of stocks. It also helps us to understand the risk of investing in a particular stock. The model is trained on the stock prices of companies which is then tested on other companies.

As this is a time series prediction, we are considering the prices of 5 years to predict the future price. Once the model is trained, we will test the performance of our model by plotting the predicted stock prices and actual stock prices of other companies.

Architecture:



The data was collected from 2 sources:

1. Kaggle: The data from Kaggle was primarily in csv format. A common dataset that consisted data for all the companies and separate csv files for all the companies individually.
2. Yahoo: We have used data reader to extract live data from yahoo.com. The data is in same format as that obtained from Kaggle. This data has only been used for analysis and visualization, not for modelling.

We have primarily used Jupyter notebook for analysis and modelling. Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to

combine software code, computational output, explanatory text and multimedia resources in a single document. A Jupyter Notebook provides you with an easy-to-use, interactive data science environment that doesn't only work as an integrated development environment (IDE), but also as a presentation or educational tool. Jupyter is a way of working with Python inside a virtual "notebook" and is growing in popularity with data scientists in large part due to its flexibility. It gives you a way to combine code, images, plots, comments, etc., in alignment with the step of the "data science process."

We have used Python libraries such as pandas, matplotlib, numpy and seaborn to make visualizations. The NumPy arrays takes significantly less amount of memory as compared to python lists. It also provides a mechanism of specifying the data types of the contents, which allows further optimization of the code.

The following are some of the advantages of the Pandas library:

1. It can present data in a way that is suitable for data analysis via its Series and DataFrame data structures.
2. The package contains multiple methods for convenient data filtering.
3. Pandas has a variety of utilities to perform Input/Output operations in a seamless manner. It can read data from a variety of formats such as CSV, TSV, MS Excel, etc.

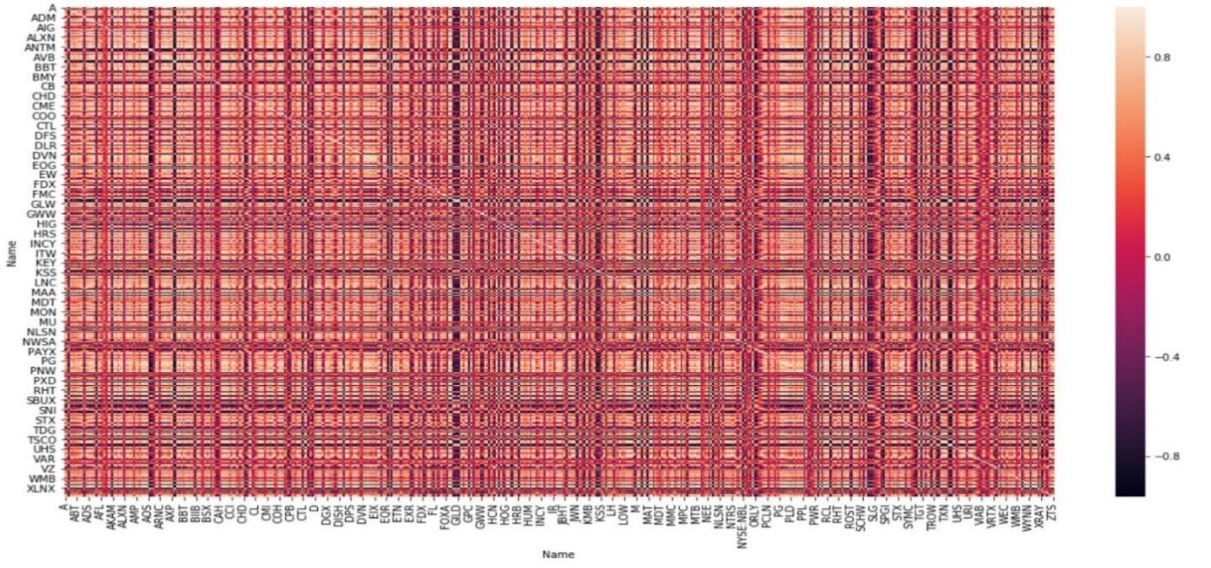
One of Matplotlib's most important features is its ability to play well with many operating systems and graphics backends. Matplotlib supports dozens of backends and output types, which means you can count on it to work regardless of which operating system you are using or which output format you wish. This cross-platform, everything-to-everyone approach has been one of the great strengths of Matplotlib. It has led to a large user base, which in turn has led to an active developer base and Matplotlib's powerful tools and ubiquity within the scientific Python world

Apart from Python, we have implemented visualizations using Tableau. Tableau is the fastest growing data visualization and data analytics tool that aims to help people see and understand data. In order to transform the way people, use data to solve problems, tableau software ensures to meet strict requirements. In other words, it simply converts raw data into a very easily understandable format.

Data analysis is great, as it is a powerful visualization tool in the business intelligence industry. Data that is created using this software becomes so easy that it allows even a non-technical user

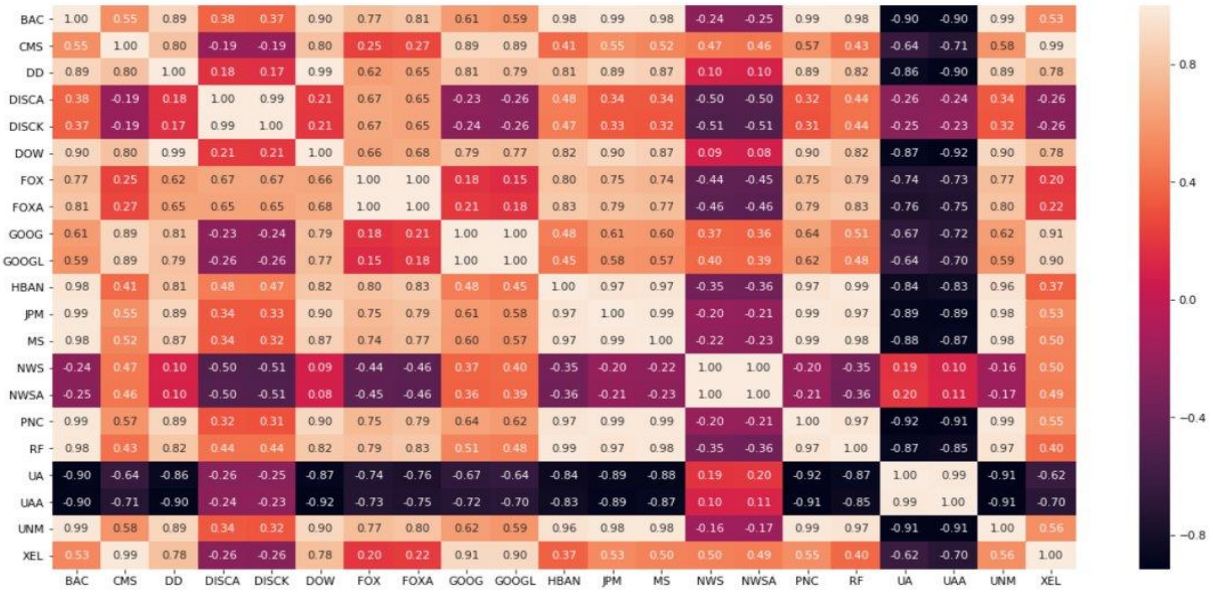
to create a customized dashboard. It provides top class interactive data visualization with the purpose to help organizations solve their data problems.

Correlation Matrix:

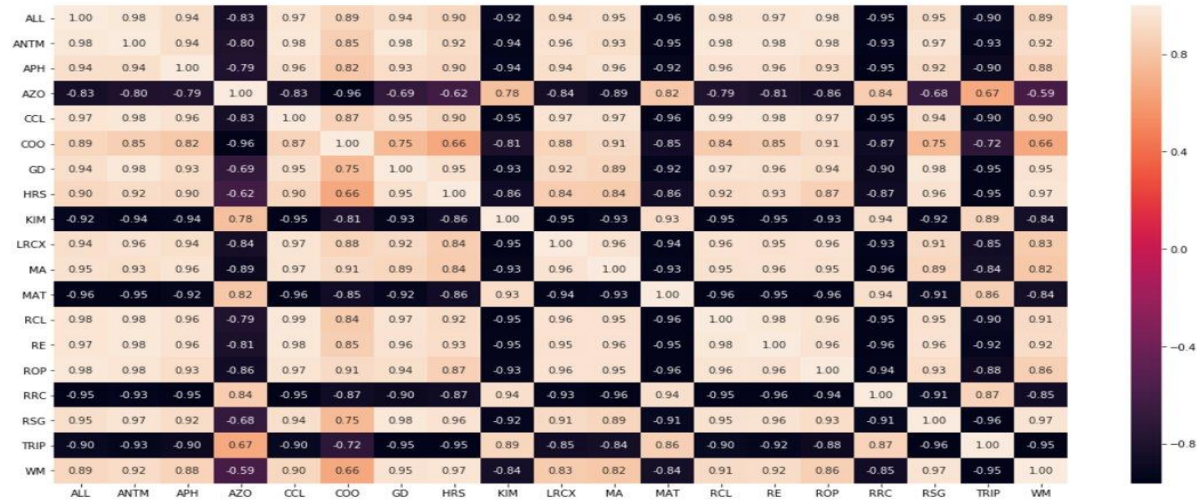



```
cor = data.pivot('Date', 'Name', 'Growth').corr()
sns.heatmap(cor)
plt.show()
```

We have plotted correlation matrix using heatmap of seaborn library. It helps us to better understand the correlation matrix for all S&P 500 companies. Name, date and growth are used to produce the above correlation matrix. The above diagram is too difficult to interpret and write down findings. So, let's focus on strong positive correlation and strong negative correlation between companies in the correlation matrix



The correlation matrix will let the investors know which stocks are highly correlated and so that they can either invest or avoid highly correlated stocks. For example, the above picture highlights strong positive correlation stocks companies like Morgan Stanley and PNC financial services are highly correlated so if someone makes high profit by investing in Morgan Stanley and they will also look for similar stocks. With the help of above correlated diagram investors can find these stocks with ease.

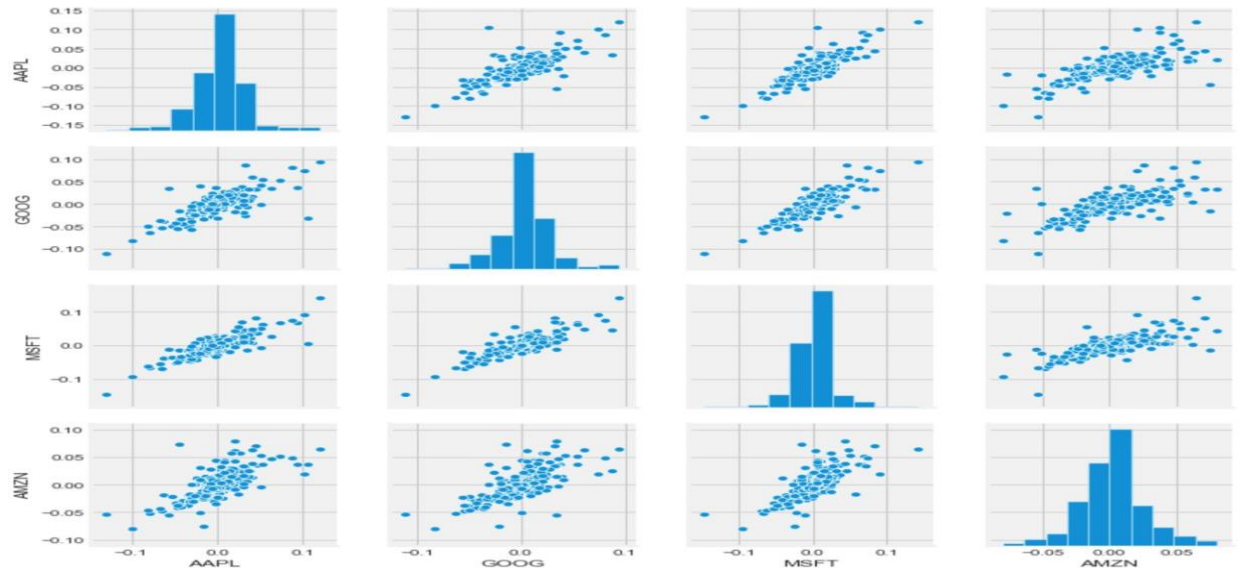


The above correlation matrix highlights strong negative correlation. These stocks are not dependent with each other. So, some investors think instead of investing in similar stocks and lose money it is better to invest in stocks that are not correlated at all. All state incorporation and Mattel Inc are example of high negative correlation.

Use Case:

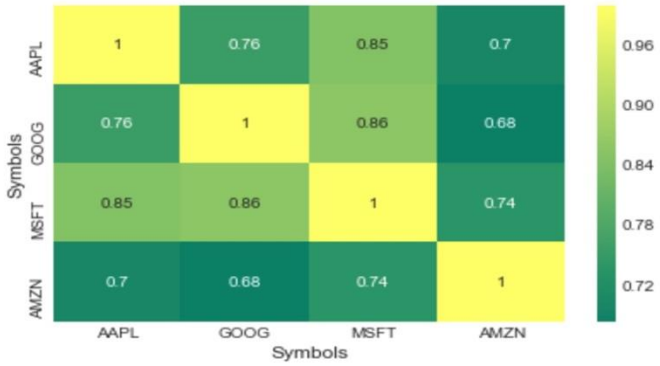
To dive into specifics, we selected the top 4 companies for analysis which are Google, Microsoft, Amazon and Apple. These are companies have highest market cap with more than 1 Trillion USD.

Seaborn is used to derive the following plot. It is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. A pair plot is used to plot a pairwise relationships in a dataset. The following contains interactive plot comparing correlations using the daily return which measures the dollar change in a stock's price as a percentage of the previous day's closing price.



The below figure shows a more visually appealing correlation matrix. Here we can google, and Microsoft have high correlation among tech companies since Microsoft and google have many

associated products. Google and amazon have slightly less correlation due amazons' own application in the mobile phones.



We have come up with an interesting question which is “How would your portfolio perform if you invested $1 in each company initially at the start of the period and did not touch it for one year?” We wanted to know if it might end up in profit or loss. First, we took the start of that particular year with name and closing price of the values and then mapped dictionary to store the value. After that we calculated the change in closing price with that of the previous day. We calculated the sum of the values and derived the mean. Our experiment concluded that we ended up with 55$ profit and 11% rate of return.

```
# Obtain all names to corresponding close...
name, close = data[data['Date'] == data['Date'].min()][['Name', 'Close']].T.values

# Map to dictionary...
base = {n : c for n, c in zip(name, close)}

# Use base to add Growth to data...
Base            = np.array(list(map(lambda x : base[x], data['Name'].values)))
data['Growth'] = data['Close'] / Base - 1

obj['Growth'].sum(), obj['Growth'].mean()
```

```
(55.180658630825114, 0.11058248222610244)
```

Risk Analysis

Every investor will calculate the risk while investing in the stock market. Many people will determine to invest in stock depending on the risk factor. We calculated the risk analysis and plotted it against expected return. This will help the investors to know which stock to invest list.

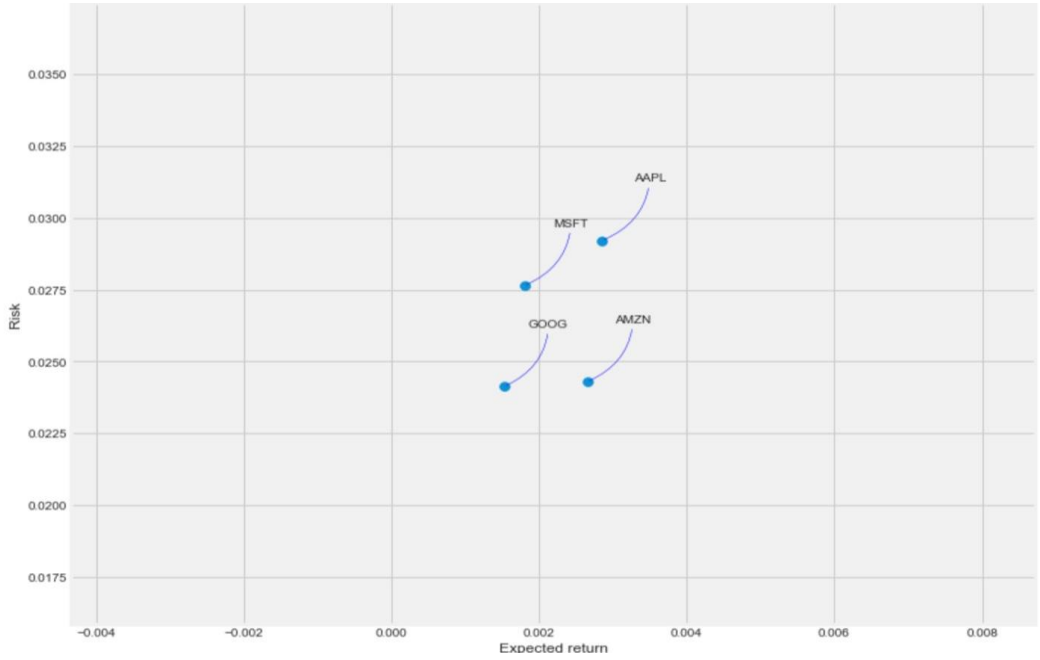
For example, if we look at the below graph, we can say that Amazon has low risk and high expected return compared to other ones. Apple has high expected return with high risk. Investors can take safe approach or the risk one. We can plot other companies also to get the desired results.

```
tech_rets=df.pct_change()
rets = tech_rets.dropna()

area = np.pi*20

plt.figure(figsize=(12, 10))
plt.scatter(rets.mean(), rets.std(), s=area)
plt.xlabel('Expected return')
plt.ylabel('Risk')

for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(label, xy=(x, y), xytext=(50, 50), textcoords='offset points', ha='right', va='bottom',
                 arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))
```

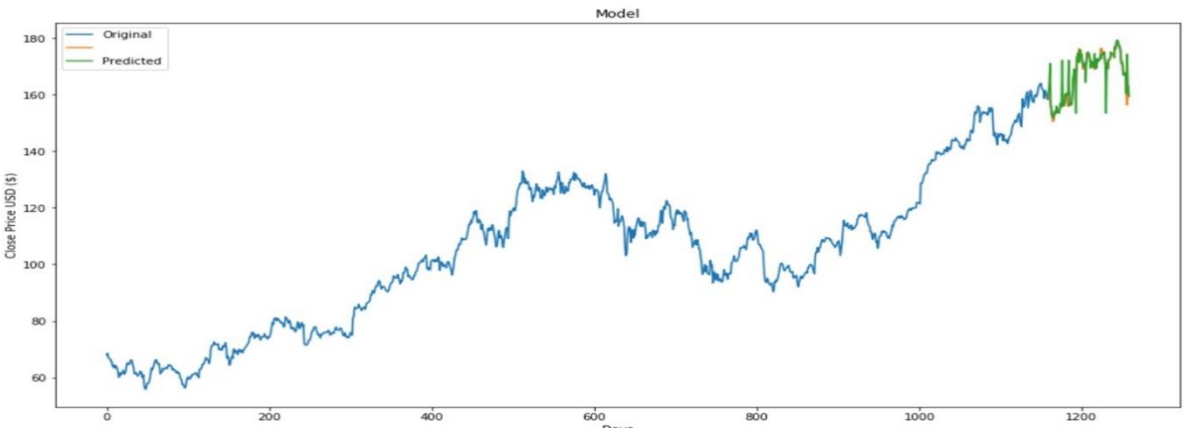


Apple Analysis

Apple is one of the most valuable companies in the world. Apple products are valued across the world. They are also having high sales. We are predicting the closing price of the stock for up to 100 days in the future. Implemented Linear Regression and Decision tree algorithms. We trained 80% of data and tested 20%. The accuracy of the prediction when compared to real time stock

price is 73.84% for linear regression and 58.42 for decision tree. Clearly linear regression model gave us a good prediction.

The below image visually describes the original and predicted price



Google Analysis

The below image has Google analysis with green bar indicates that the stock yielded a profit for that day and red bar indicates that the stock yielded a loss that day. This is calculated by open and close price of that particular day.

Machine learning methods:

Linear Regression:

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables.

Decision Tree Regressor:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
# Implementing Linear and Decision Tree Regression Algorithms.
tree = DecisionTreeRegressor().fit(x_train, y_train)
lr = LinearRegression().fit(x_train, y_train)

x_future = df2.drop(['Prediction'], 1)[:-future_days]
x_future = x_future.tail(future_days)
x_future = np.array(x_future)
x_future
```

```
array([[144.53],
       [143.68],
       [143.79],
       [143.65],
       [146.58],
       [147.51],
```

```
accuracy = lr.score(x_test,y_test)
print(accuracy*100,'%')
```

```
73.84202705813678 %
```

The accuracy of Linear Regression is 73.84%.

```
accuracy = tree.score(x_test,y_test)
print(accuracy*100,'%')
```

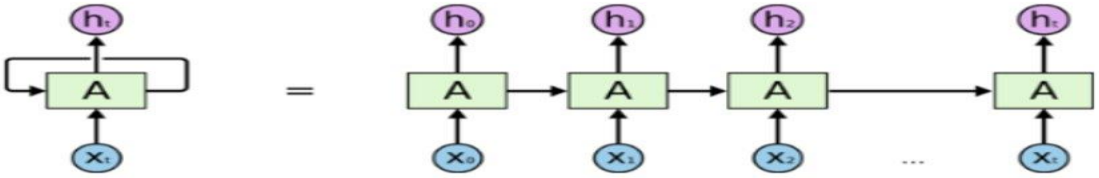
```
58.417663292674746 %
```

The accuracy of Decision Tree Regressor is 58.41%

Deep learning:

We used LSTM for predicting google stock price. Before we dive deeper into LSTM let's learn about Neural Network and Recurrent Neural Network. Neural Networks are set of algorithms which closely resemble the human brain and are designed to recognize patterns. They interpret sensory data through a machine perception, labelling or clustering raw input. They can recognize numerical patterns, contained in vectors, into which all real-world data (images, sound, text or time series), must be translated.

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. A small issue regarding recurrent neural network is has gradient vanishing and exploding problems. As the data goes further in the recurrent neural network the initial values start to disapper. This problem is solved in the LSTM



An unrolled recurrent neural network.

The Problem, Short-term Memory

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So, if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weight. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

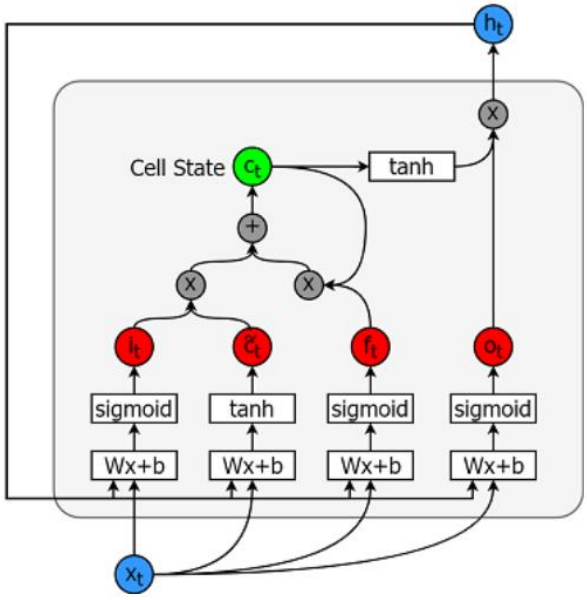
What is LSTM?

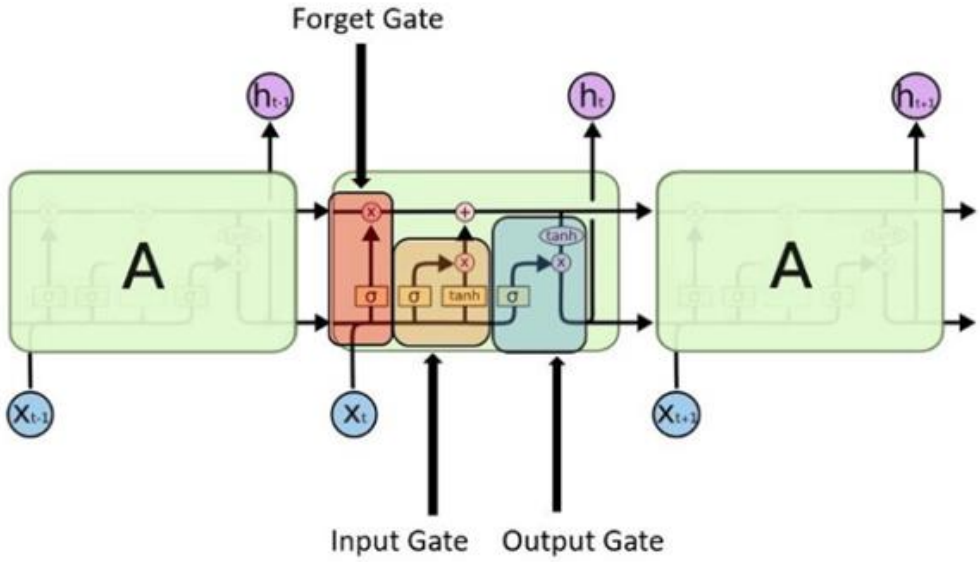
Long Short-Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. LSTM has a chain-like structure with repeating modules just like RNN but instead of a single Neural network layer in RNN, LSTM has four layers which are interacting in a very different way each performing its unique function in the network.

- Cell state (c_t) - This represents the internal memory of the cell which stores both short term memory and long-term memories
- Hidden state (h_t) - This is output state information calculated w.r.t. current input, previous hidden state and current cell input which you eventually use to predict the future stock market prices. Additionally, the hidden state can decide to only retrieve the short or long-term or both types of memory stored in the cell state to make the next prediction.
- Input gate (i_t) - Decides how much information from current input flows to the cell state

- Forget gate (f_t) - Decides how much information from the current input and the previous cell state flows into the current cell state
- Output gate (o_t) - Decides how much information from the current cell state flows into the hidden state, so that if needed LSTM can only pick the long-term memories or short-term memories and long-term memories

A cell is pictured below.





Forget gate

First, we have the forget gate. This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

Input Gate

To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

Output Gate

Last, we have the output gate. The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

```
# reshape
# Choice "open" feature:
dataset = data.iloc[:,1].values
dataset = dataset.reshape(-1,1) # (975,) sometimes can be problem
dataset = dataset.astype("float32")
dataset.shape
```

```
]: (975, 1)
```

```
# scaling
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

```
# train test split
train_size = int(len(dataset) * 0.75) # Split dataset 75% for train set, 25% for test set
test_size = len(dataset) - train_size
train = dataset[0:train_size,:]
test = dataset[train_size:len(dataset),:]
print("train size: {}, test size: {} ".format(len(train), len(test)))
```

```
train size: 731, test size: 244
```

```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore_vanilla = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore_vanilla))
```

```
Train Score: 10.41 RMSE
Test Score: 41.49 RMSE
```

For LSTM model 75% of data set is used for train data and 25% is used for test set. RMSE for train data came out to be 10.41 and for test score its 41.49

Challenges:

These are the challenges we encounter in this project. They are mentioned as follows.

1. Stock market prices are highly unpredictable and volatile. This means that there are no consistent patterns in the data that allow you to model stock prices over time near-perfectly.
2. Trading is done on numerous companies and each company have several factors to consider, hence it is quite difficult to narrow down and predict on specific ones.
3. For each company we have to consider different factors, it is challenging to find out the variables which helps in predicting the closing value of the stock.
4. It is obvious that data won't be near readily available, lot of feature transformation, cleaning the data to be done.

References:

https://medium.com/@josephabraham9996/time-series-analysis-on-stock-market-forecasting-arima-prophet-2b60cacf604

https://towardsdatascience.com/performing-a-time-series-analysis-on-the-aapl-stock-index-3655da9612ff

https://www.researchgate.net/publication/311933224_Stock_market_prediction_using_time_series

https://towardsdatascience.com/predicting-stock-price-with-lstm-13af86a74944

https://towardsdatascience.com/lstm-time-series-forecasting-predicting-stock-prices-using-an-lstm-model-6223e9644a2f

https://www.kdnuggets.com/2018/11/keras-long-short-term-memory-lstm-model-predict-stock-prices.html

https://www.sciencedirect.com/science/article/pii/S1877050920304865

https://www.statisticssolutions.com/what-is-linear-regression/

https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/