Stock Prices Prediction Using Machine Learning and Deep Learning Techniques

The regular habits of traders are observing trends of the stock market which is uncertain because of an economic, liquidity crisis, interest rate, exchange rate fluctuations, politics, and fund flow. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy.

In this project, I will work with historical data about the stock prices of a publicly listed company. We will implement a mix of machine learning algorithms to predict the future stock price, starting with simple algorithms Linear Regression, and then move on to advanced techniques like Long Term Short Memory.

The objective of the project is to perform how the algorithms work and how to predict the stock price.

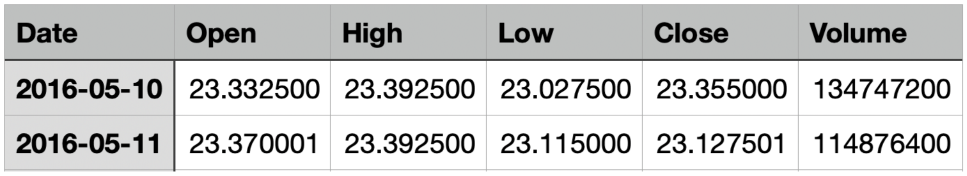
We'll be using a dataset from www.finance.yahoo.com, This is the website where you can find historical data for various stocks. For this project I have used the data for 'Apple Inc.'

Table of Contents

Data Visualization and Exploration

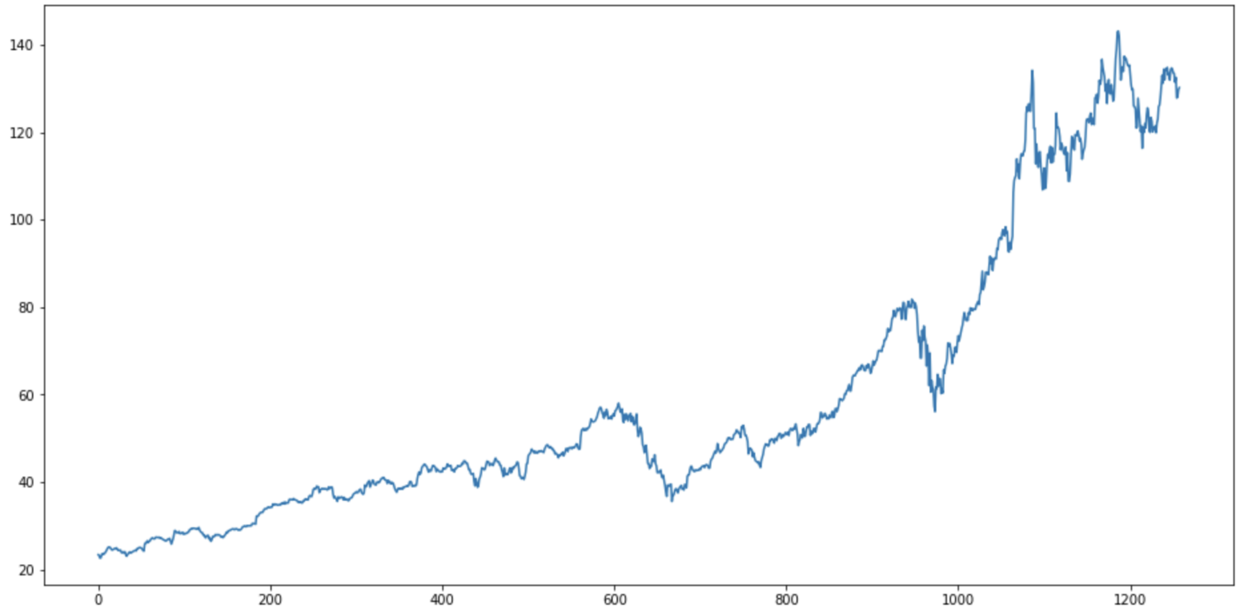
There are many significant variables about the dataset;

- Date - means when the date of data is
- Open - means the starting price on that date
- Close - means the final price on that date
- High - means the maximum price on that date
- Low - means the minimum price on that date
- Volume - means the summation of trade on that date



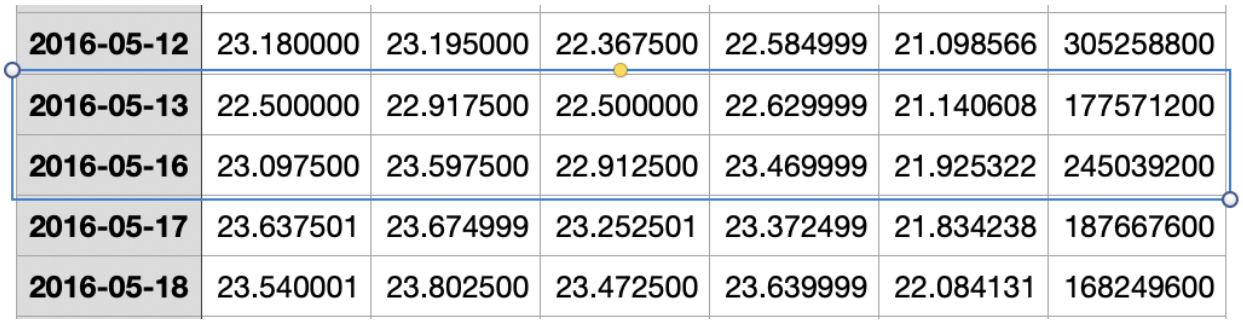
Date	Open	High	Low	Close	Volume
2016-05-10	23.332500	23.392500	23.027500	23.355000	134747200
2016-05-11	23.370001	23.392500	23.115000	23.127501	114876400

We can calculate the profit or loss from the final price each day, hence we will consider the closing price as the target variable. Then plot the closing price to understand how it's shaping up in our data



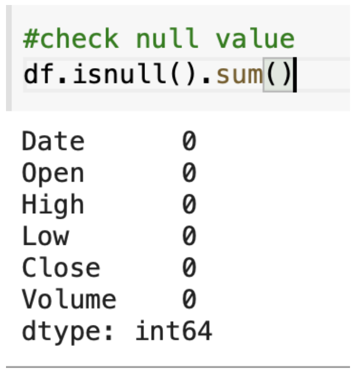
Another important thing is the stocks market always be closed on weekends and public holidays for example [in figure]

14th May and 15th May are during the weekends, so the data will not be collected.



2016-05-12	23.180000	23.195000	22.367500	22.584999	21.098566	305258800
2016-05-13	22.500000	22.917500	22.500000	22.629999	21.140608	177571200
2016-05-16	23.097500	23.597500	22.912500	23.469999	21.925322	245039200
2016-05-17	23.637501	23.674999	23.252501	23.372499	21.834238	187667600
2016-05-18	23.540001	23.802500	23.472500	23.639999	22.084131	168249600

Check if they have a null value or not. As below, none of null value.



```
#check null value
df.isnull().sum()
```

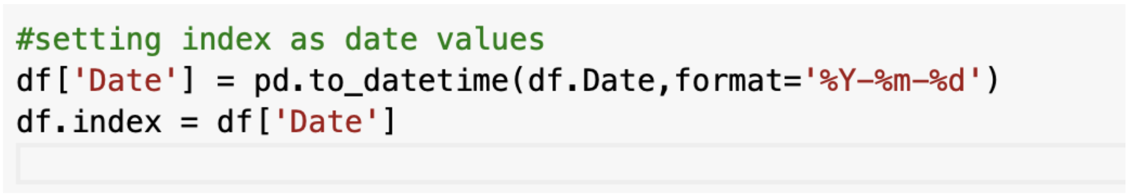
```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

LINEAR REGRESSION

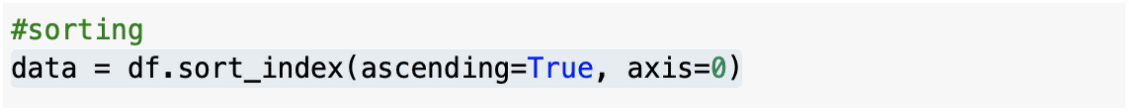
The most basic machine learning algorithm that can be implemented on this data is linear regression. The linear regression model returns an equation that determines the relationship between the independent variables and the dependent variable.

- **Preprocessing**

First set the Date column to be the date and set format to be Year-month-day and sort the dataset by date in ascending order.

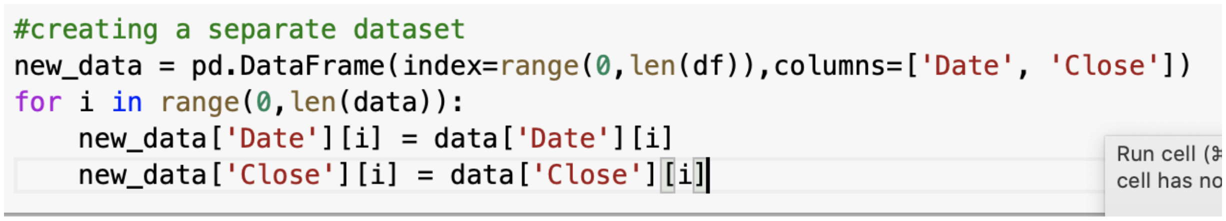


```
#setting index as date values
df['Date'] = pd.to_datetime(df.Date,format='%Y-%m-%d')
df.index = df['Date']
```



```
#sorting
data = df.sort_index(ascending=True, axis=0)
```

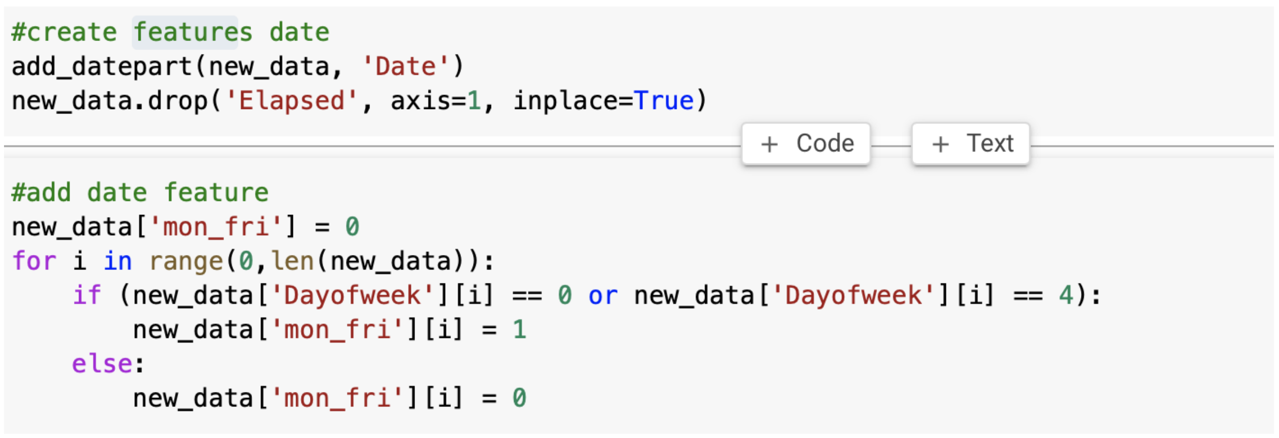
Separating the data before creating features for preventing the data combined with original data.



```
#creating a separate dataset
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])
for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]
```

I believe that days of the week would be relevant for prices. For instance, my hypothesis is that the first and last days of the week could potentially affect the closing price of the stock far more than the other days. So, I created features that identifies a given day

Creates features used add_datepart from fastai library.



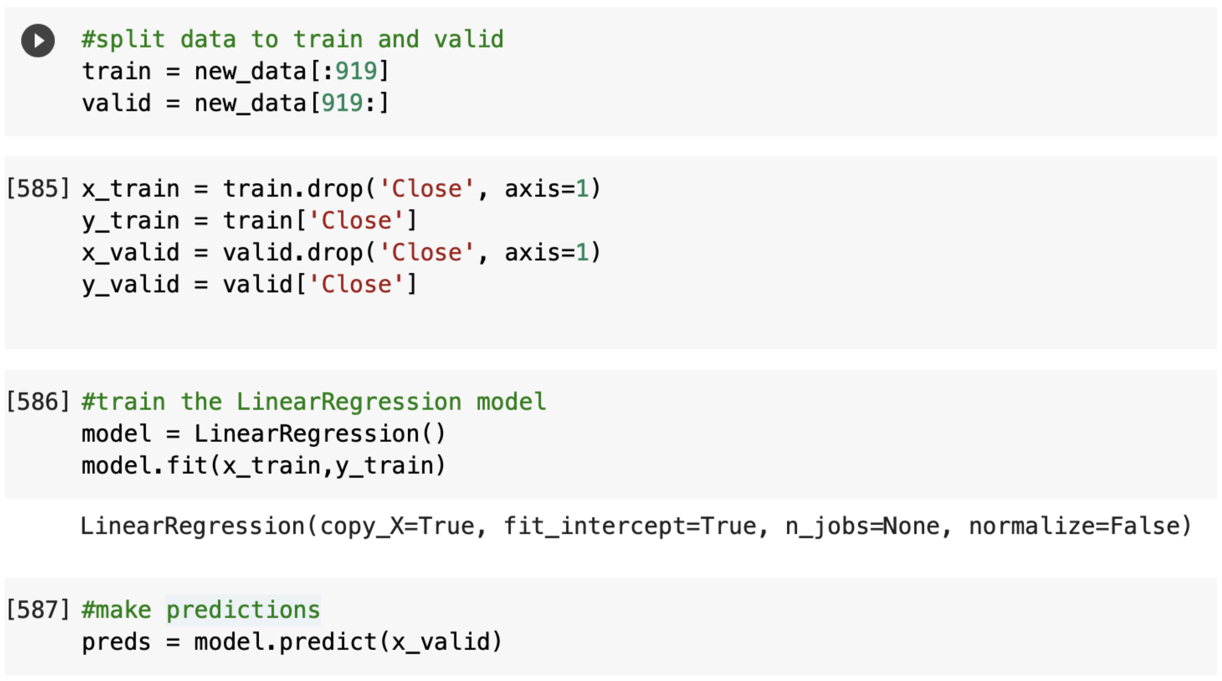
```
#create features date
add_datepart(new_data, 'Date')
new_data.drop('Elapsed', axis=1, inplace=True)
```

```
#add date feature
new_data['mon_fri'] = 0
for i in range(0,len(new_data)):
    if (new_data['Dayofweek'][i] == 0 or new_data['Dayofweek'][i] == 4):
        new_data['mon_fri'][i] = 1
    else:
        new_data['mon_fri'][i] = 0
```

If the day of week is equal to 0 or 4, the column value will be 1, otherwise 0.

- **Data Modelling**

Split data, first 919 data will be set as a train set to train the model, and after that will be set as a valid set to check the performance. And create x_train, y_train, x_valid and y_valid then training the Linear Regression model with x_train and y_train and make predictions



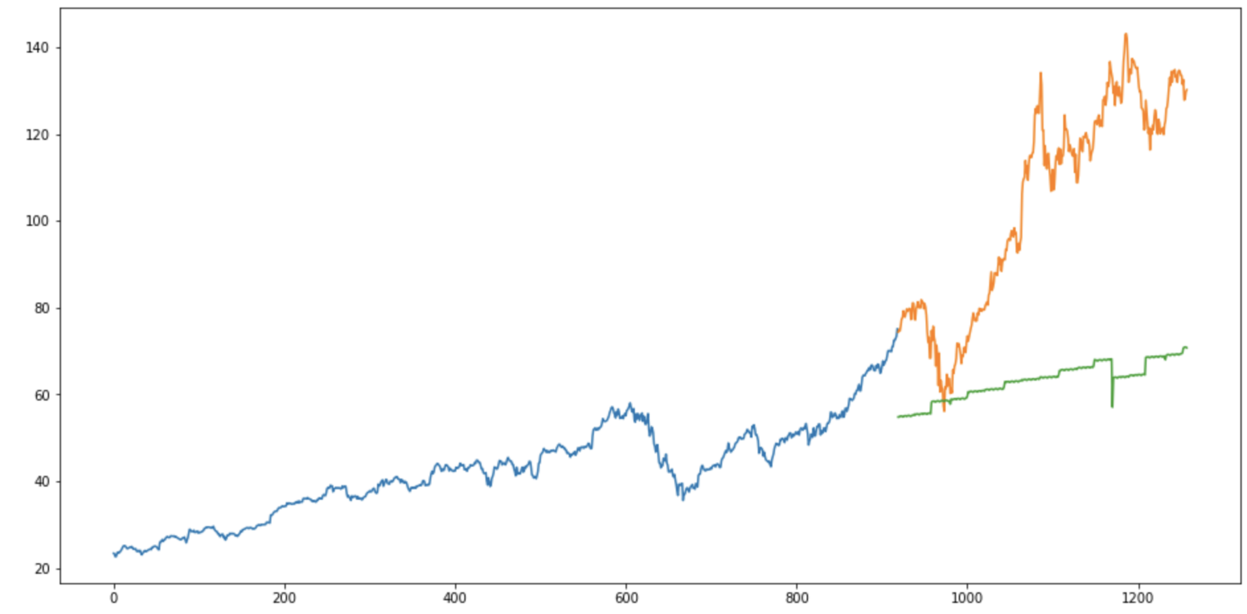
```
#split data to train and valid
train = new_data[:919]
valid = new_data[919:]
```

```
[585] x_train = train.drop('Close', axis=1)
      y_train = train['Close']
      x_valid = valid.drop('Close', axis=1)
      y_valid = valid['Close']
```

```
[586] #train the LinearRegression model
      model = LinearRegression()
      model.fit(x_train,y_train)

      LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
[587] #make predictions
      preds = model.predict(x_valid)
```

- **Results**



The RMSE value is high, which clearly shows that linear regression has performed poorly.

```
[588] #find the rms
      rms=np.sqrt(np.mean(np.power((np.array(y_valid)-np.array(preds)),2)))
      rms
```

```
45.960836927541884
```

- **Conclusion**

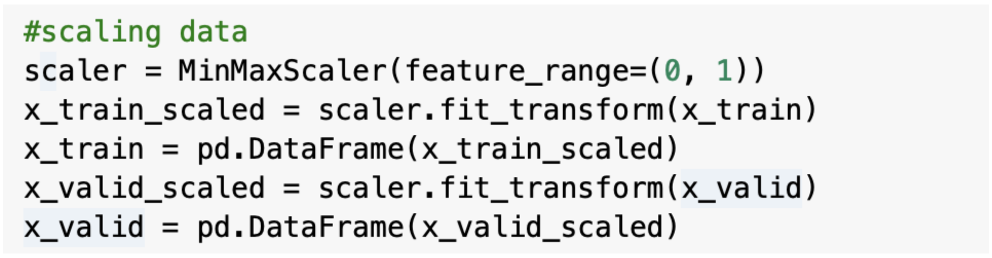
Linear reversion is a basic technique and has some negative aspects such as overfitting data. The model will compare the value from the similar date with the previous month, and the similar date or month with the year before.

KNN (K-Nearest-Neighbors)

Another interesting ML algorithm that one can use here is kNN (k nearest neighbours). Based on the independent variables, kNN finds the similarity between new data points and old data points.

- **Preprocessing**

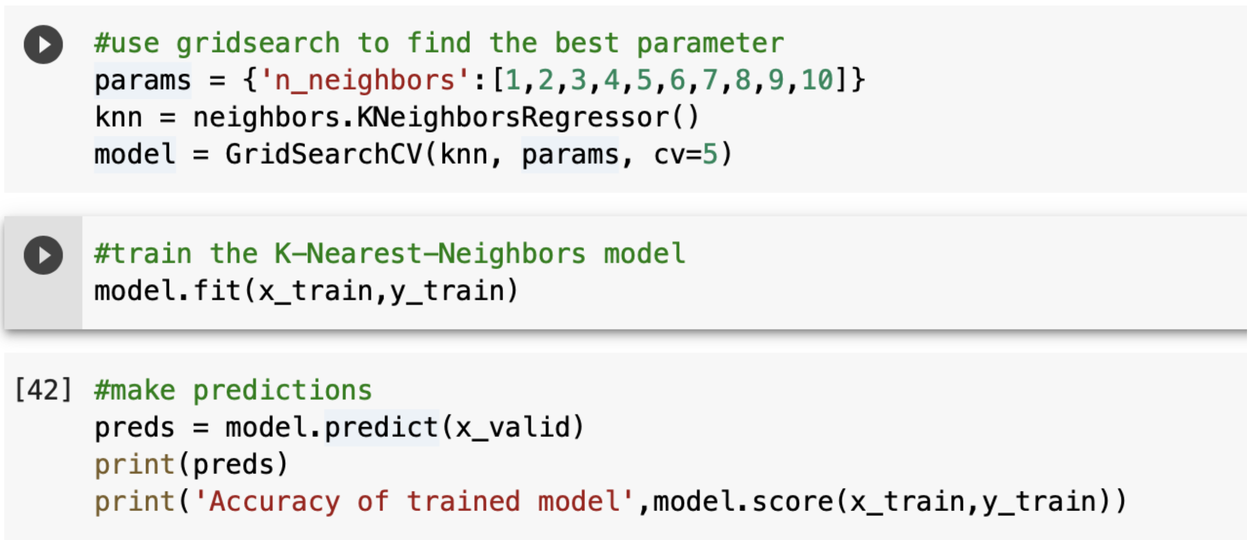
For this techni it’s using the same train and valid set from the linear regression section but it’s need to scale the data to make all variables similarly scaled and centered



```
#scaling data
scaler = MinMaxScaler(feature_range=(0, 1))
x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)
x_valid_scaled = scaler.fit_transform(x_valid)
x_valid = pd.DataFrame(x_valid_scaled)
```

- **Data Modelling**

Using gridsearch to find the best parameter in the dataset and set grid search cross-validation equal to 5 then training the K-Nearest-Neighbor model with x_train and y_train and make predictions

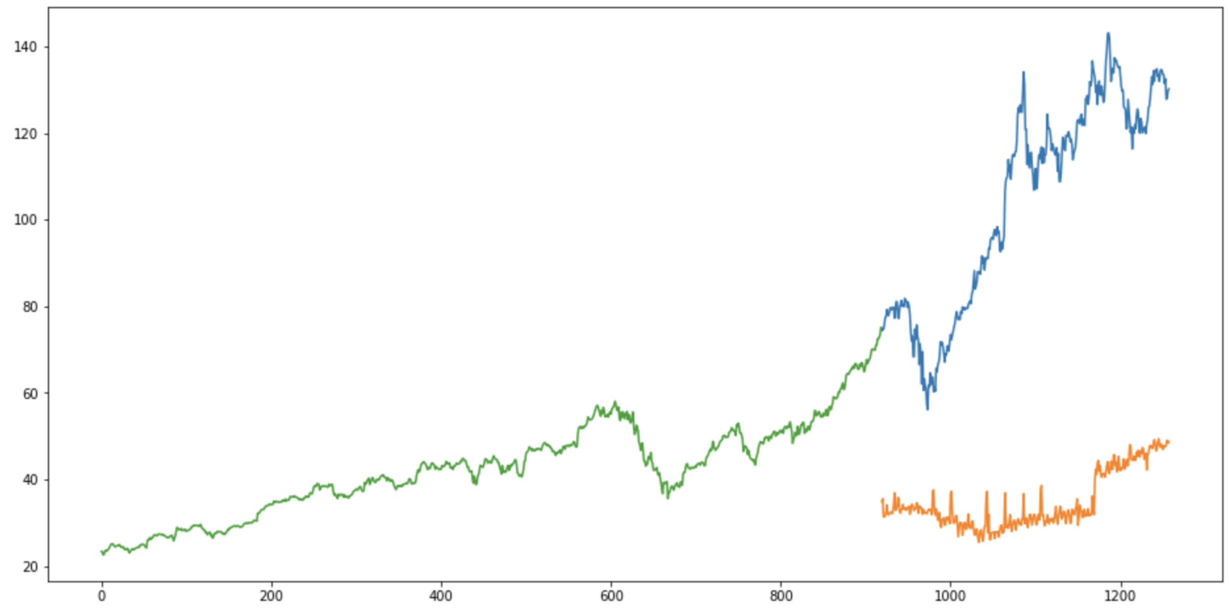


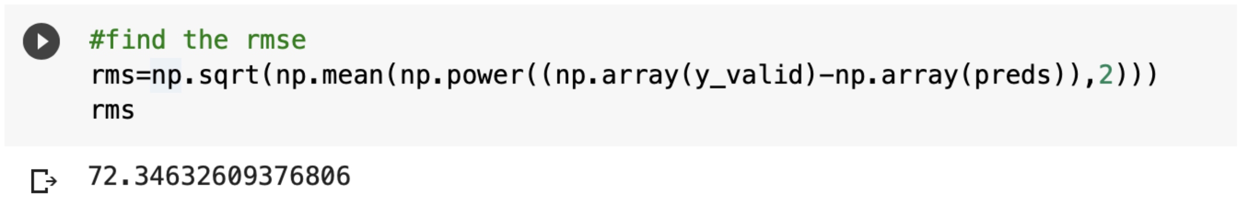
```
#use gridsearch to find the best parameter
params = {'n_neighbors':[1,2,3,4,5,6,7,8,9,10]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)
```

```
#train the K-Nearest-Neighbors model
model.fit(x_train,y_train)
```

```
[42] #make predictions
     preds = model.predict(x_valid)
     print(preds)
     print('Accuracy of trained model',model.score(x_train,y_train))
```

- **Results**





```
#find the rmse
rms=np.sqrt(np.mean(np.power((np.array(y_valid)-np.array(preds)),2)))
rms
```

```
72.34632609376806
```

- **Conclusion**

The RMSE value is higher then the linear regression model, and form plot shows that results of both two models fluctuate form close price. We can say that regression algorithms have not performed well on this type of dataset.

PROPHET

There are a number of time series techniques that can be implemented on the stock prediction dataset, but most of these techniques require a lot of data preprocessing before fitting the model. Prophet, designed and pioneered by Facebook, is a time series forecasting library that requires no data preprocessing and is extremely simple to implement.

- **Preprocessing**

First create a dataframe into two columns; Date and Close. Then, set the Date column to be the date and set format to be Year-month-day

Create an input for the Prophet is a dataframe with two columns target and date (y and ds).

```
#creating dataframe
new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])

for i in range(0,len(data)):
    new_data['Date'][i] = data['Date'][i]
    new_data['Close'][i] = data['Close'][i]

new_data['Date'] = pd.to_datetime(new_data.Date,format='%Y-%m-%d')
new_data.index = new_data['Date']
```

```
#preparing data
new_data.rename(columns={'Close': 'y', 'Date': 'ds'}, inplace=True)
```

- **Data Modelling**

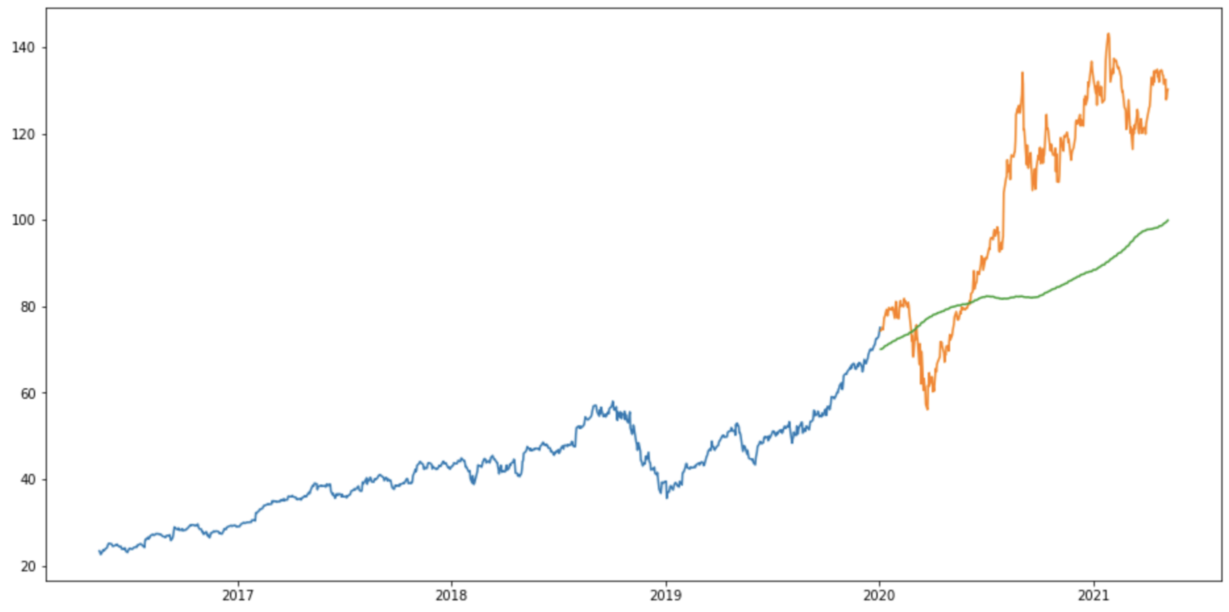
Split data, first 919 data will be set as a train set to train the model, and after that will be set as a valid set to check the performance then training the Prophet model with train and make predictions

```
#split data to train and valid
train = new_data[:919]
valid = new_data[919:]
```

```
#train the Prophet model
model = Prophet()
model.fit(train)
```

```
#predictions
close_prices = model.make_future_dataframe(periods=len(valid))
forecast = model.predict(close_prices)
```

- **Results**



```
#find the rmse
forecast_valid = forecast['yhat'][919:]
rms=np.sqrt(np.mean(np.power((np.array(valid['y'])-np.array(forecast_valid)),2)))
rms
```

```
27.309376651326936
```

- **Conclusion**

The Prophet tries to capture the trends from the previous data, usually performing well on time datasets. For the stock market, the prices will not be certain because of various factors. Hence, the prediction also will not be accurate.

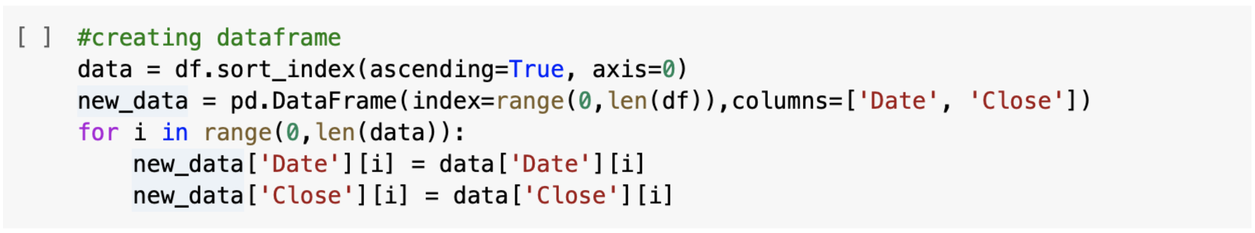
Long Short Term Memory (LSTM)

LSTMs are widely used for sequence prediction problems and have proven to be extremely effective. The reason they work so well is because LSTM is able to store past information that is important, and forget the information that is not. LSTM has three gates:

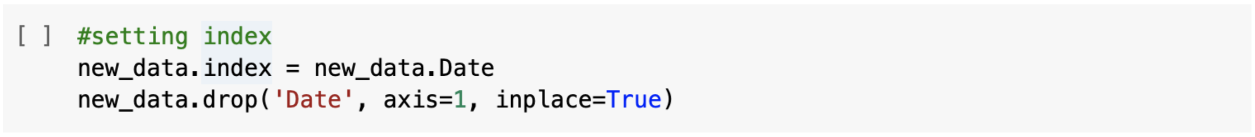
- **The input gate:** The input gate adds information to the cell state
- **The forget gate:** It removes the information that is no longer required by the model
- **The output gate:** Output Gate at LSTM selects the information to be shown as output

- **Preprocessing**

Sort the dataset by date in ascending order and create a dataframe into two columns; Date and Close. Then, set the date to be an index and delete an existing Date column.



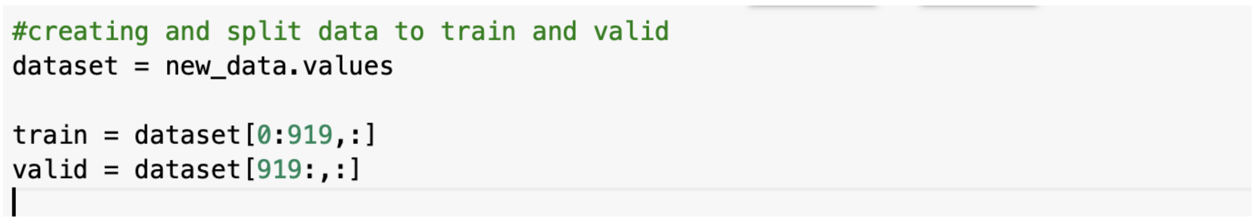
```
[ ]  #creating dataframe
     data = df.sort_index(ascending=True, axis=0)
     new_data = pd.DataFrame(index=range(0,len(df)),columns=['Date', 'Close'])
     for i in range(0,len(data)):
         new_data['Date'][i] = data['Date'][i]
         new_data['Close'][i] = data['Close'][i]
```



```
[ ]  #setting index
     new_data.index = new_data.Date
     new_data.drop('Date', axis=1, inplace=True)
```

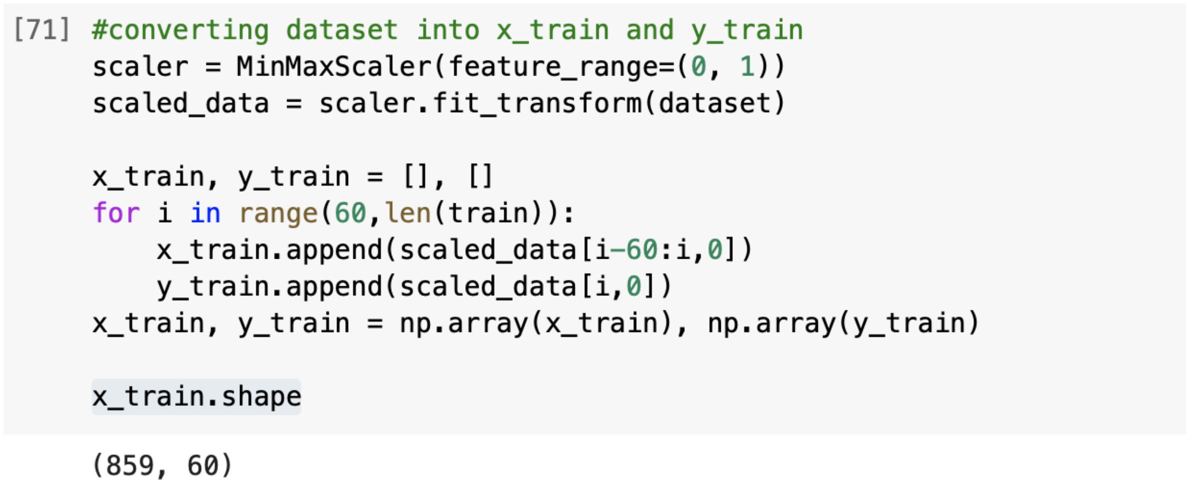
- **Data Modelling**

Split data, first 919 data will be set as a train set to train the model, and after that will be set as a valid set to check the performance.



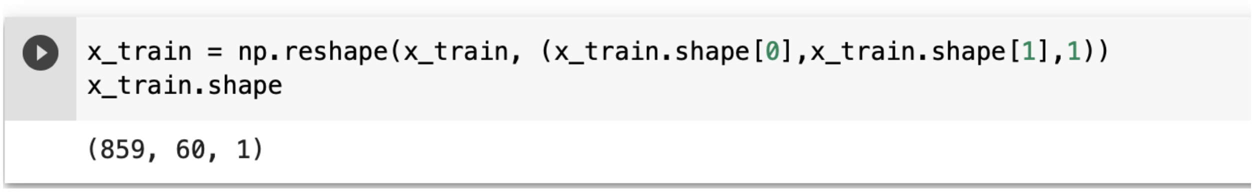
```
#creating and split data to train and valid
dataset = new_data.values

train = dataset[0:919,:]
valid = dataset[919:,:]
```

Convert the dataset into x_train and y_train, and convert both x_train and y_train into a numpy array.



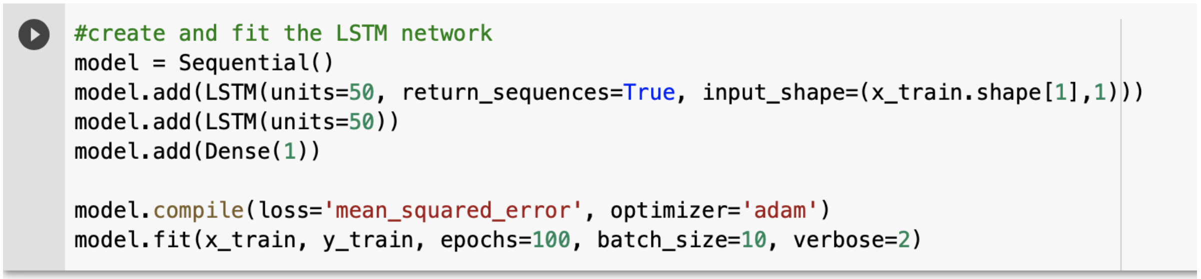
```
[71] #converting dataset into x_train and y_train
     scaler = MinMaxScaler(feature_range=(0, 1))
     scaled_data = scaler.fit_transform(dataset)

     x_train, y_train = [], []
     for i in range(60,len(train)):
         x_train.append(scaled_data[i-60:i,0])
         y_train.append(scaled_data[i,0])
     x_train, y_train = np.array(x_train), np.array(y_train)

     x_train.shape

     (859, 60)
```

Reshape numpy array from one-dimensional array into three-dimensional array, with 859 samples 60 times steps and 1 feature at each time step.

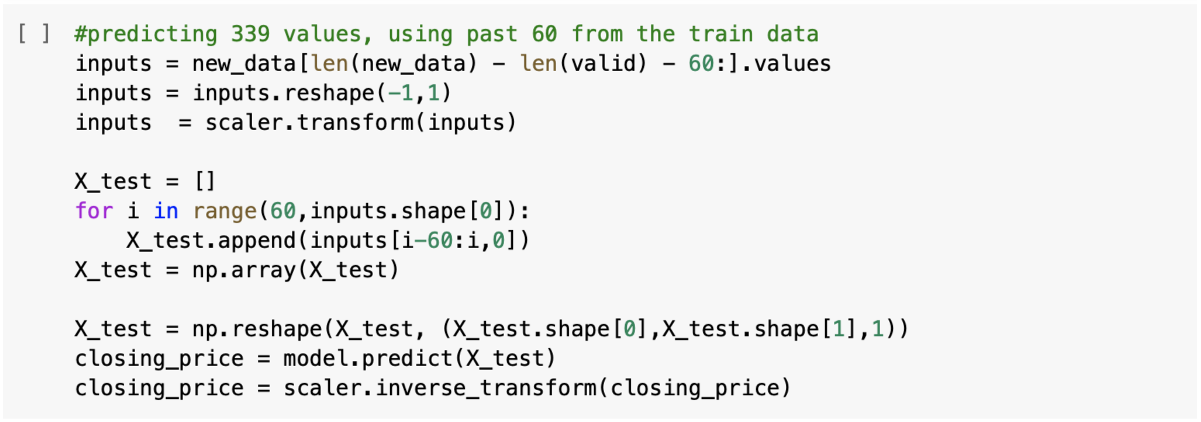


```
x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))
x_train.shape

(859, 60, 1)
```

Add hidden layer 1 and 2, equal to 50 units in each, and the last layer which is equal to 1 unit. Then, compile and train with 100 epochs.

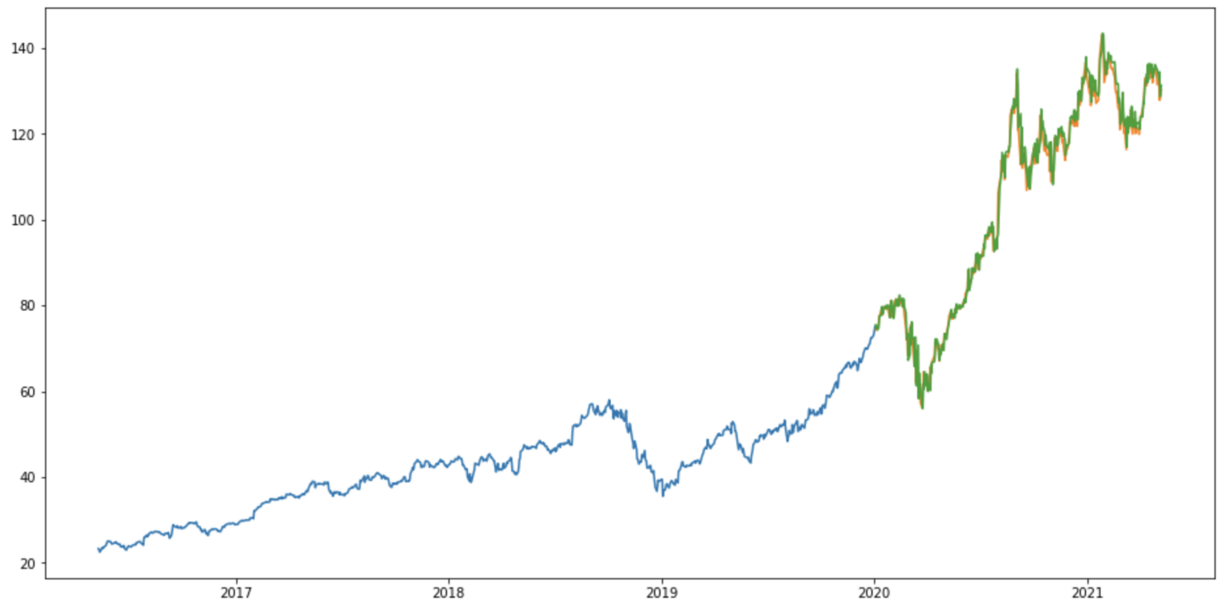


```
#create and fit the LSTM network
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(units=50))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(x_train, y_train, epochs=100, batch_size=10, verbose=2)
```



```
[ ] #predicting 339 values, using past 60 from the train data
    inputs = new_data[len(new_data) - len(valid) - 60:].values
    inputs = inputs.reshape(-1,1)
    inputs  = scaler.transform(inputs)

    X_test = []
    for i in range(60,inputs.shape[0]):
        X_test.append(inputs[i-60:i,0])
    X_test = np.array(X_test)

    X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
    closing_price = model.predict(X_test)
    closing_price = scaler.inverse_transform(closing_price)
```

- **Results**



```
[75] #find the rmse
     rms=np.sqrt(np.mean(np.power((valid-closing_price),2)))
     rms

     2.8528660505367625
```

- **Conclusion**

LSTM is the most accurate model and LSTM model can be tuned for various parameters such as changing the number of LSTM layers, adding dropout value or increasing the number of epochs to increasing the accurate But the predictions from LSTM are not enough to identify whether the stock price will increase or decrease

Conclusion

From researching this project, linear regression does not perform well enough in prediction. Found that Long Short Term Memory or LSTM is the best model when comparing with other three models because they are able to store past information. This is important because the previous close price of stock price is crucial in predicting the future close price.

On the other hands, the stock price is affected by many factors like the economy, the politics, and fund flows. So, it is often difficult to predict the trends surely and certainly. The traders should have studied these factors rather than depend only on the models.

Reference

https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/
https://www.statisticssolutions.com/what-is-linear-regression/

https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26
https://realpython.com/knn-python/

https://facebook.github.io/prophet/docs/quick_start.html
https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/

https://towardsdatascience.com/lstm-time-series-forecasting-predicting-stock-prices-using-an-lstm-model-6223e9644a2f
https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

Tanapol Buangam 6110545511