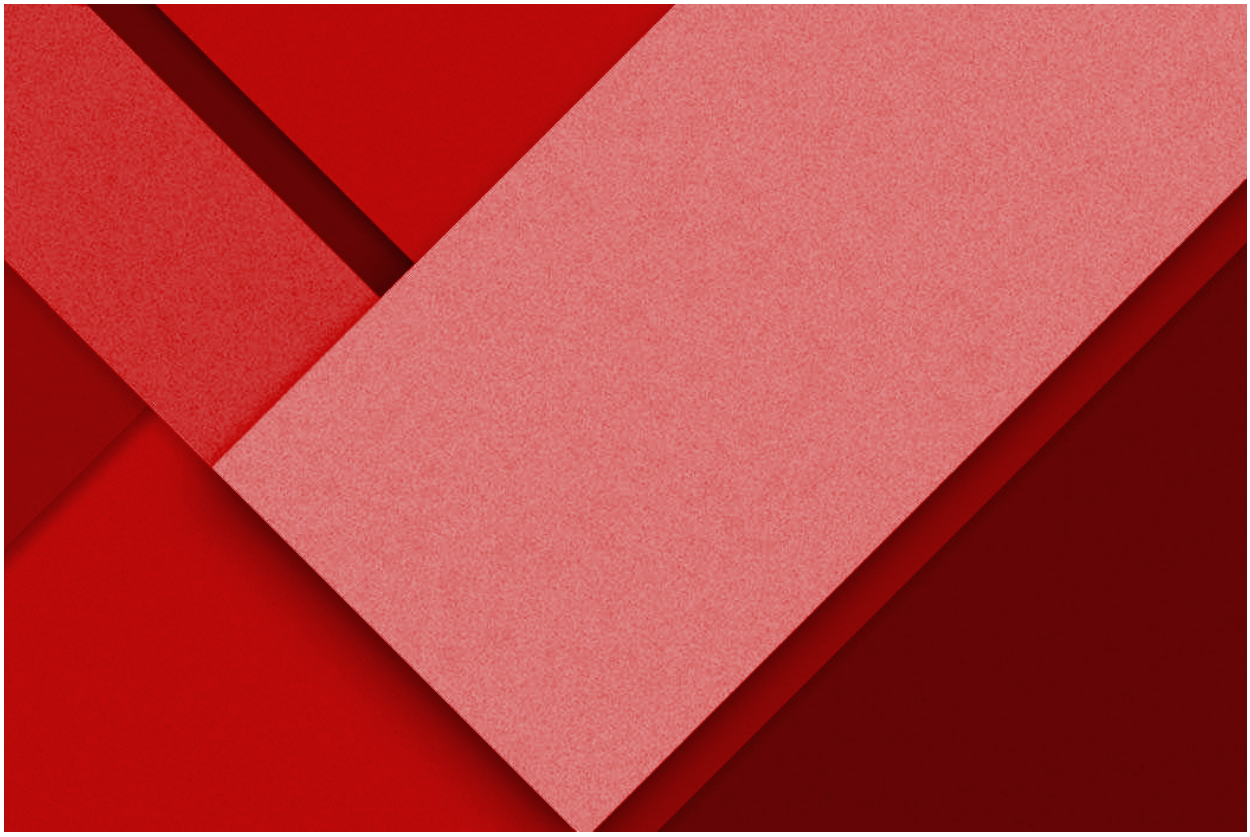


Artificial Intelligence for Economics Project Report

Forecasting Stock Market Trade using SMA, EMA, ARIMA and LSTMs

- Arup Baral (20MA20010)
- Shatansh Patnaik (20MA20067)
- Atharv Bajaj (20MA20014)
- Bussa Varshith (20MA20016)
- Princy (20MA20046)
- Masipeddi Saavn (20BT30014)

Contents:

Overview about the Project:

One can show machine learning models vast amounts of historical data of a company's stock (several decades' worth of data) and use the model to extract key trends and essential features that define the company's stock performance. The Stock Market is known for being volatile and dynamic in nature. Predicting the stock market is influenced by several factors such as political events, unexpected accidents, global economic conditions etc. In this project we will visualize the stock market as a time series data, and will apply different models to it in order to predict what the future data looks like. We'll also compare and justify why certain models give a better fit to the data than the rest of themodels. We need to keep in mind owing to the volatility of the stock prices, they cannot be visualized as randomly generated numbers, but they are analyzed as a sequence of discrete time data. Therefore as the data can be visualized in a sequential manner, we are eligible to apply the moving average models in order to predict it. We'll start off by applying the simple moving average model to predict the data, next we'll apply the exponential moving average model to the data and see the predictions and compare them with the predictions obtained from SMA.Time Series data may not be stationary. We'll apply ARIMA(Autoregressive Integrated Moving Average Model) that will convert it to stationary data (We'll test it separately using the Augmented DickeyFuller Test) and predict the prices. We shall also use LSTMs (Long ShortTerm Memory) for Stock Prices Forecasting.

An LSTM is a Recurrent Neural Network that works on data sequences, learning to retain only relevant information from a time window. New information which the network learns is added to a "memory" that gets updated with each timestep based on how significant the new sample seems to the model. At the end we will analyze, compare and infer why certain models give us the best fit for the data.

Data Analysis:

Data Summary and Source:

In this project, we will be using the closing prices of Apple's stock (AAPL) from the last 21 years from 1st November 1999 to 9th July 2021. For loading the data in the notebook, we will be using Alpha Vintage which is a free API for historical and real-time stock market data.

For training all the models in the project, we will be using 70-30 split or in other words, 70% of the data will be used for training purposes while 30% will be used for testing the results.

```
[ ] import pandas as pd
```

```
[ ] df=pd.read_csv("data.csv")
```

```
[ ] df
```

	Unnamed: 0	Date	Low	High	Close	Open
0	5453	2021-07-06	140.070	143.15	142.02	140.07
1	5452	2021-07-02	137.745	140.00	139.96	137.90
2	5451	2021-07-01	135.760	137.33	137.27	136.60
3	5450	2021-06-30	135.870	137.41	136.96	136.17
4	5449	2021-06-29	134.350	136.49	136.33	134.80
...	...	...	...	...	...	...
5449	4	1999-11-05	84.000	88.37	88.31	84.62
5450	3	1999-11-04	80.620	85.37	83.62	82.06
5451	2	1999-11-03	81.000	83.25	81.50	81.62
5452	1	1999-11-02	77.310	81.69	80.25	78.00
5453	0	1999-11-01	77.370	80.69	77.62	80.00

5454 rows × 6 columns

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5454 entries, 0 to 5453
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  5454 non-null   int64  
 1   Date        5454 non-null   object 
 2   Low         5454 non-null   float64
 3   High        5454 non-null   float64
 4   Close       5454 non-null   float64
 5   Open        5454 non-null   float64
dtypes: float64(4), int64(1), object(1)
memory usage: 255.8+ KB
```

```
[ ] df_train
```

	Date	Low	High	Close	Open
5453	1999-11-01	77.37	80.69	77.62	80.00
5452	1999-11-02	77.31	81.69	80.25	78.00
5451	1999-11-03	81.00	83.25	81.50	81.62
5450	1999-11-04	80.62	85.37	83.62	82.06
5449	1999-11-05	84.00	88.37	88.31	84.62
...	...	...	...	...	...
1640	2014-12-29	113.70	114.77	113.91	113.79
1639	2014-12-30	112.11	113.92	112.52	113.64
1638	2014-12-31	110.21	113.13	110.38	112.82
1637	2015-01-02	107.35	111.44	109.33	111.39
1636	2015-01-05	105.41	108.65	106.25	108.29

3818 rows × 5 columns

Time Series Analysis:

What is ARIMA?

We already know that time series data (prices of the stock market) vs the corresponding dates will be non-stationary data. So we are required to convert the data to stationary data. This is achieved by using the Auto-regressive Integrated Moving Average (ARIMA) Model. It is one of the most widely used models for predicting linear time series data. Let us suppose that there are two quantities (X and Y (which is X's previous values in this case)) such that X's current value is predicted from X's previous values which implies that the future value of X could be predicted using the present values of X.

What is AR?

AR (Autoregressive) Algorithm determines the linear regression of the presently fitted values vs. the past fitted values.

$$X_t = \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

What is MA?

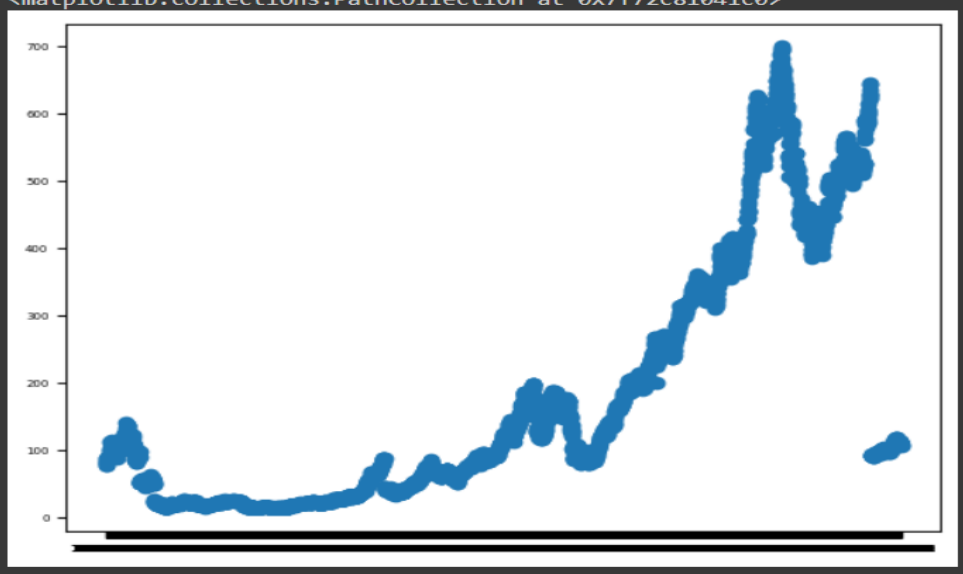
MA (Moving Average) is a stock indicator used in technical analysis which is used to smoothen out the prices data by creating a constantly updated average price. A rising moving average curve indicates that the security is in an uptrend while a declining or falling curve indicates a downtrend.

Data Visualization:

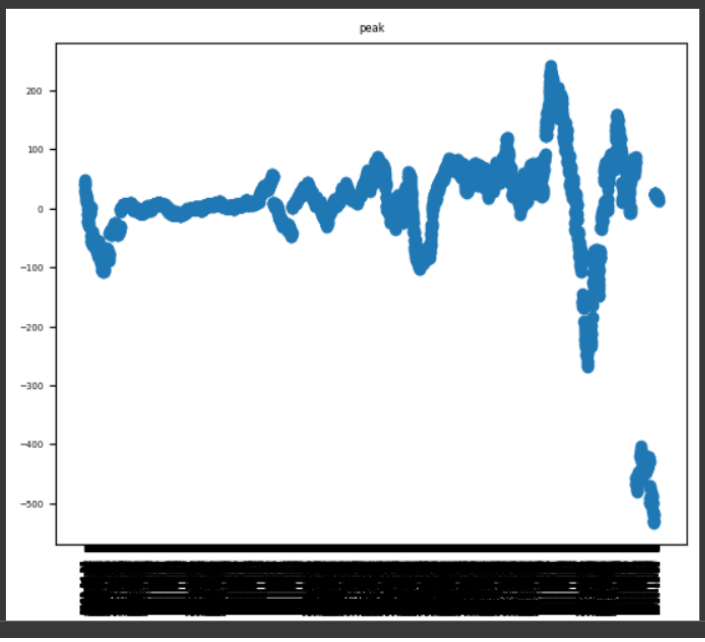
A plot between 'Low' prices and the corresponding dates:

```
plt.scatter(x,y_low)
```

```
<matplotlib.collections.PathCollection at 0x7f72c81041c0>
```



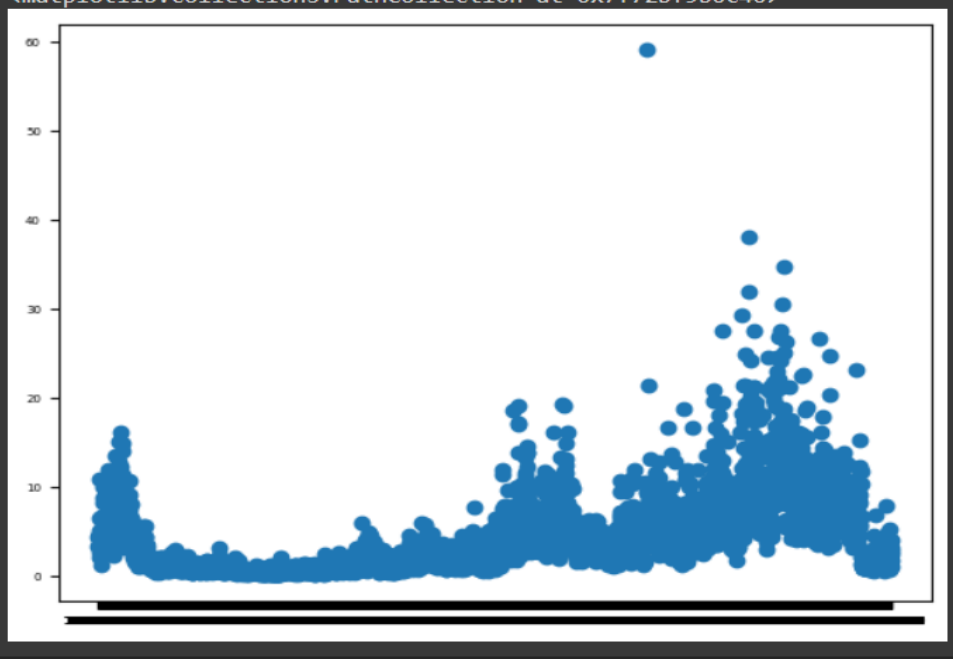

```
plt.scatter(x.iloc[time_lag:], y_high.iloc[time_lag:].to_numpy() - y_high.iloc[0:len(y_high) - time_lag].to_numpy())
plt.xticks(x.iloc[time_lag:],rotation = 90)
plt.title('peak')
plt.show()
```



We observe that the difference between the high price and low price trend is almost the same from the following scatter plot:

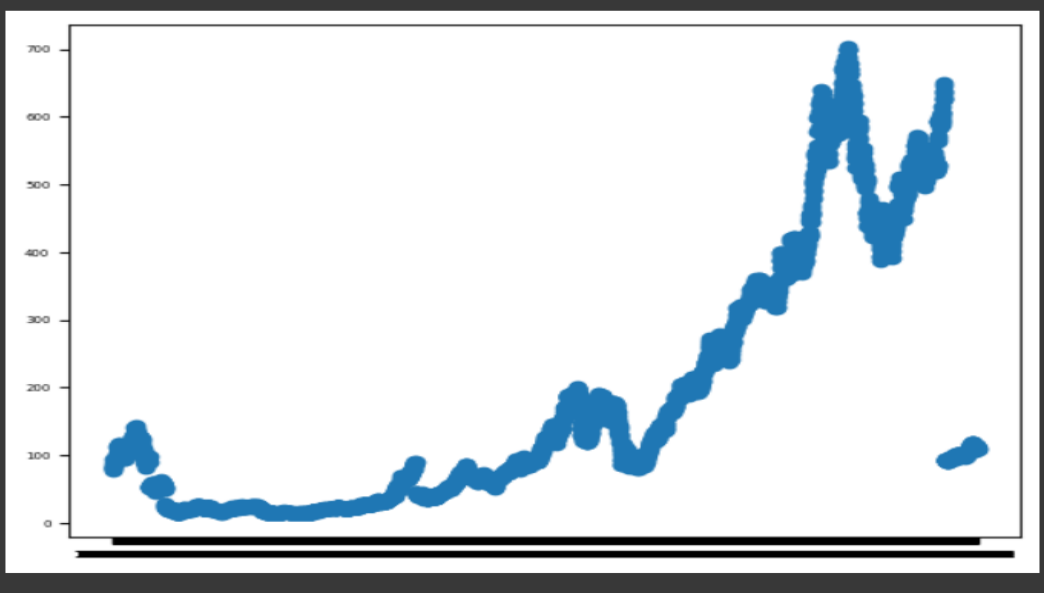
```
plt.scatter(x,y_high-y_low)
```

```
<matplotlib.collections.PathCollection at 0x7f72bf9b0c40>
```



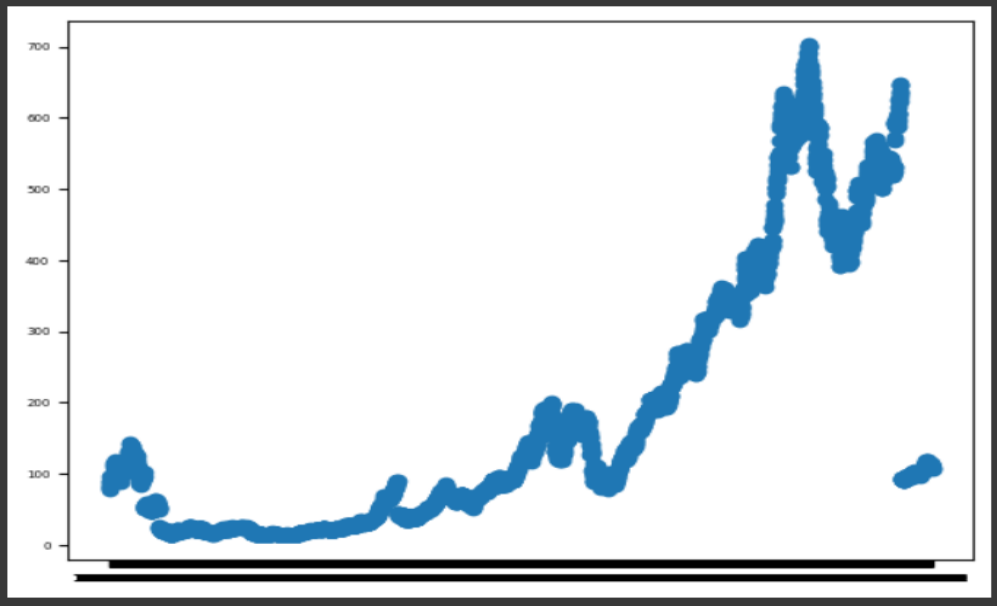
The following scatter plot shows us the trend of opening prices:

```
[ ]  plt.scatter(x,y_open)
     plt.show()
```



The following scatter plot shows us the trend of closing prices:

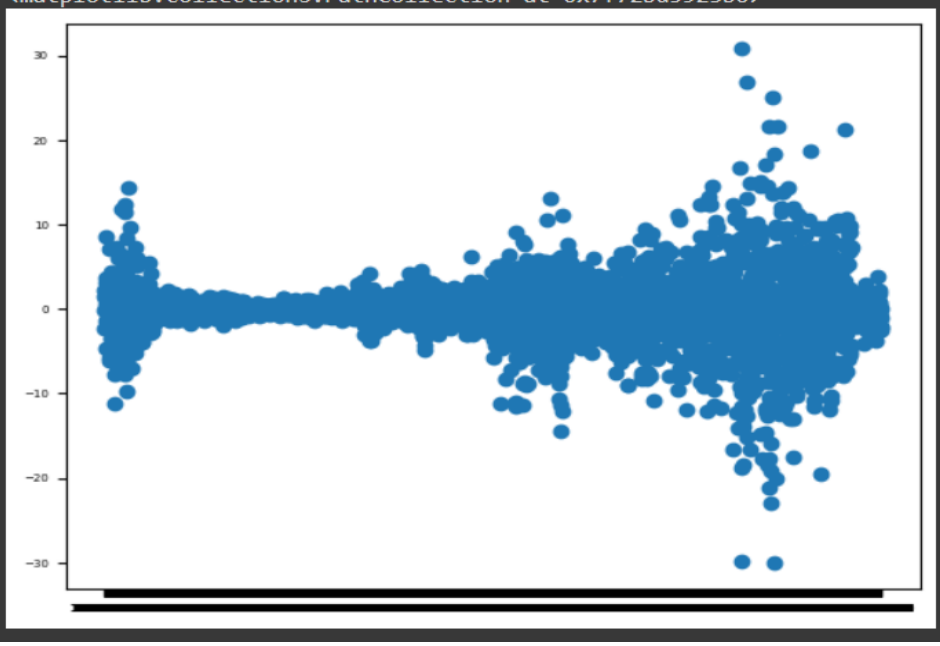
```
plt.scatter(x,y_close)
plt.show()
```



Difference between the opening and closing stock prices:

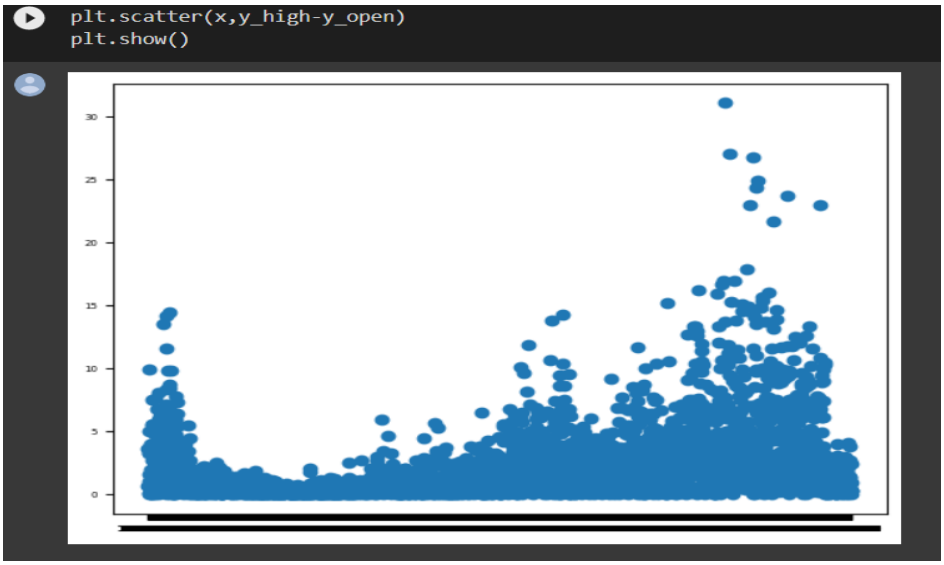
```
plt.scatter(x,y_close-y_open)
```

```
<matplotlib.collections.PathCollection at 0x7f72bd5923b0>
```



Difference between high and opening stock prices vs time:

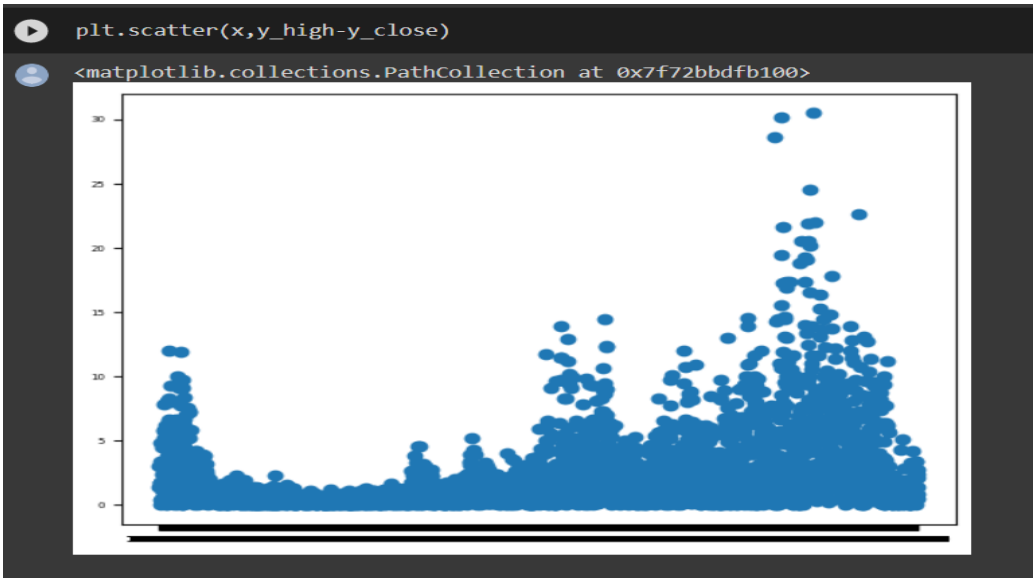
```
plt.scatter(x,y_high-y_open)
plt.show()
```



Difference between high and closing stock prices vs time:

```
plt.scatter(x,y_high-y_close)
```

```
<matplotlib.collections.PathCollection at 0x7f72bbdfb100>
```



So from the plot of the price and time we see that the opening price,closing price,highest price and the lowest price follow the same trend and the maximum difference between them is about 30 dollars so we can analyze one and safely say that others will follow the similar trend

Forecasting Models: ARIMA (Auto-Regressive Integrated Moving Average)

Point Forecasts using ARIMA:

Point Forecasts can be calculated using ARIMA using the following steps:

1. Expand the ARIMA equation so that y_t is on the left hand side and all other terms are on the right.
2. Rewrite the equation by replacing t with $T+h$.
3. On the right hand side of the equation, replace future observations with their forecasts, future errors with zero, and past errors with the corresponding residuals.

First of all we need to implement the ADF Test:

One of the most widely used statistical tests is the Dickey-Fuller test. It can be used to determine whether or not a series has a unit root, and thus whether or not the series is stationary. This test's null and alternative hypotheses are:

Null Hypothesis: The series has a unit root (value of p =1)

Alternative Hypothesis: The series has no unit root.

If the null hypothesis is not rejected, the series is said to be non-stationary. The series can be linear or different stationary as a result of this. The series becomes stationary if both the mean and standard deviation are flat lines (constant mean and constant variance).

Calculation of the p value and the implementation of ADF Test:

```
from statsmodels.tsa.stattools import adfuller
from numpy import log
result = adfuller(df_train.Open.dropna())
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

```
ADF Statistic: -1.727597
p-value: 0.416953
```

```
[ ] from statsmodels.tsa.stattools import adfuller
    from numpy import log
    result = adfuller(df_train.Close.dropna())
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
```

```
ADF Statistic: -1.740957
p-value: 0.410143
```

```
[ ] from statsmodels.tsa.stattools import adfuller
    from numpy import log
    result = adfuller(df_train.High.dropna())
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
```

```
ADF Statistic: -1.692975
p-value: 0.434735
```

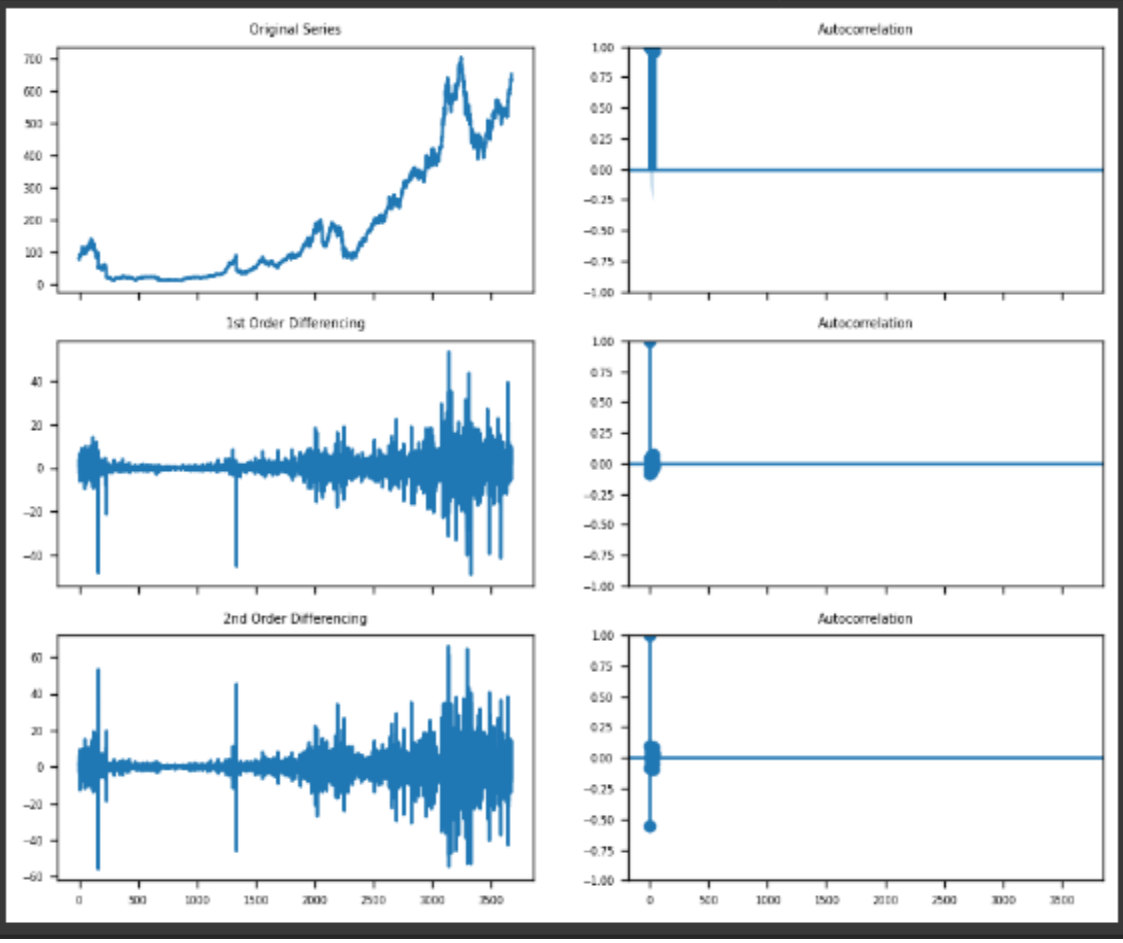
```
[ ] from statsmodels.tsa.stattools import adfuller
    from numpy import log
    result = adfuller(df_train.Low.dropna())
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
```

```
ADF Statistic: -1.751467
p-value: 0.404809
```

As the p value is more than 0.05 we go ahead to check the order of differencing

Checking the order of differencing

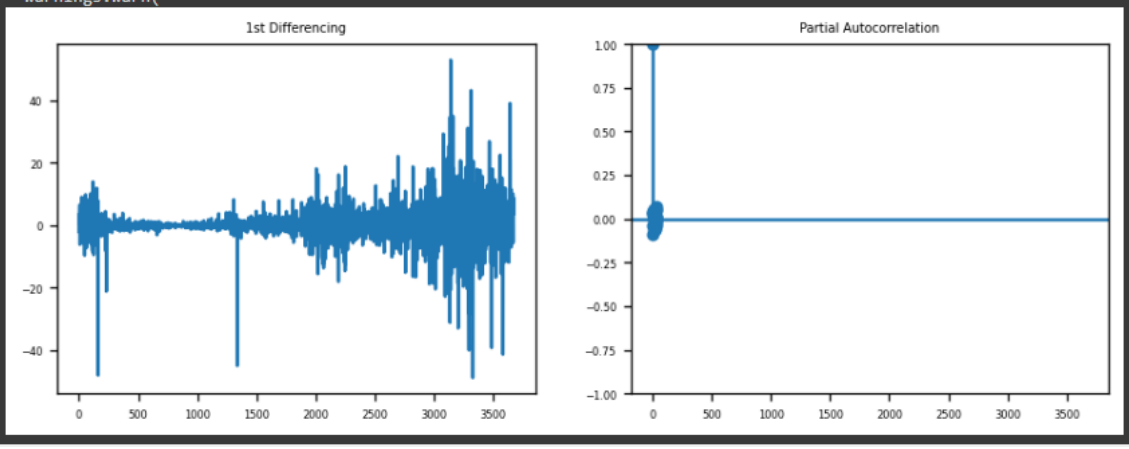
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt
plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})


# Original Series
fig, axes = plt.subplots(3, 2, sharex=True)
axes[0, 0].plot(Y_open_new); axes[0, 0].set_title('Original Series')
plot_acf(Y_open_new, ax=axes[0, 1])

# 1st Differencing
axes[1, 0].plot(Y_open_new.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(Y_open_new.diff().dropna(), ax=axes[1, 1])

# 2nd Differencing
axes[2, 0].plot(Y_open_new.diff().diff()); axes[2, 0].set_title('2nd Order Differencing')
plot_acf(Y_open_new.diff().diff().dropna(), ax=axes[2, 1])

plt.show()
```



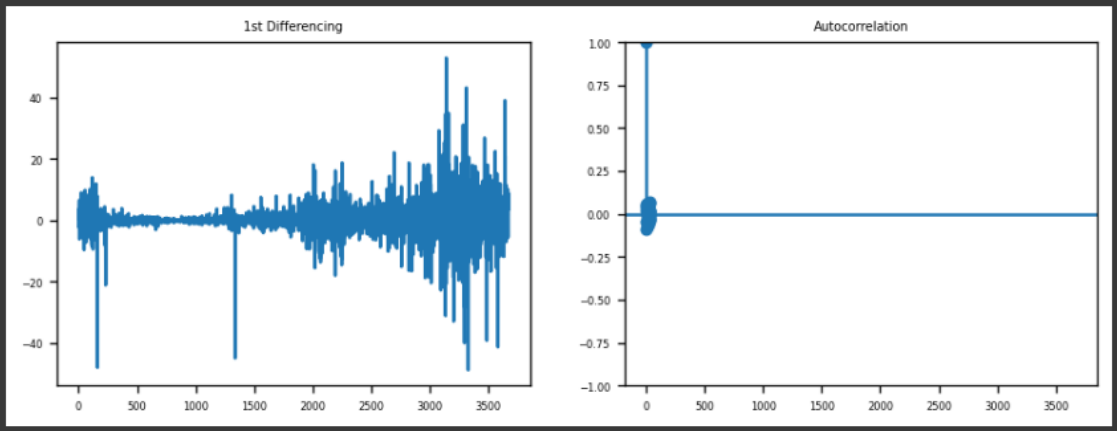
Finding the p and q values for applying the ARIMA Model:

```
#finding the p terms of ARIMA
plt.rcParams.update({'figure.figsize':(9,3), 'figure.dpi':120})

fig, axes = plt.subplots(1, 2, sharex=True)
axes[0].plot(Y_open_new.diff()); axes[0].set_title('1st Differencing')
axes[1].set(ylim=(0,5))
plot_pacf(Y_open_new.diff().dropna(), ax=axes[1])

plt.show()
#significant lag at p=1
```

```
/home/arup/.local/lib/python3.10/site-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method 'yw'
  warnings.warn(
```




```
#finding the q value
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plt.rcParams.update({'figure.figsize':(9,3), 'figure.dpi':120})

fig, axes = plt.subplots(1, 2, sharex=True)
axes[0].plot(Y_open_new.diff()); axes[0].set_title('1st Differencing')
axes[1].set(ylim=(0,1.2))
plot_acf(Y_open_new.diff().dropna(), ax=axes[1])

plt.show()
# q=1
```



Representation of Apple's Highest, Lowest, Opening and Closing Prices as a time series data:

```
[ ] plt.figure(figsize=(14,4))
    plt.plot(Apple_sales["Open"])
    plt.title('Apple Stock Opening Price', fontsize=20)
    plt.ylabel('Price', fontsize=16)
```

```
Text(0, 0.5, 'Price')
```


```
[ ] plt.figure(figsize=(14,4))
    plt.plot(Apple_sales["Close"])
    plt.title('Apple Stock Closing Price', fontsize=20)
    plt.ylabel('Price', fontsize=16)
```

```
Text(0, 0.5, 'Price')
```

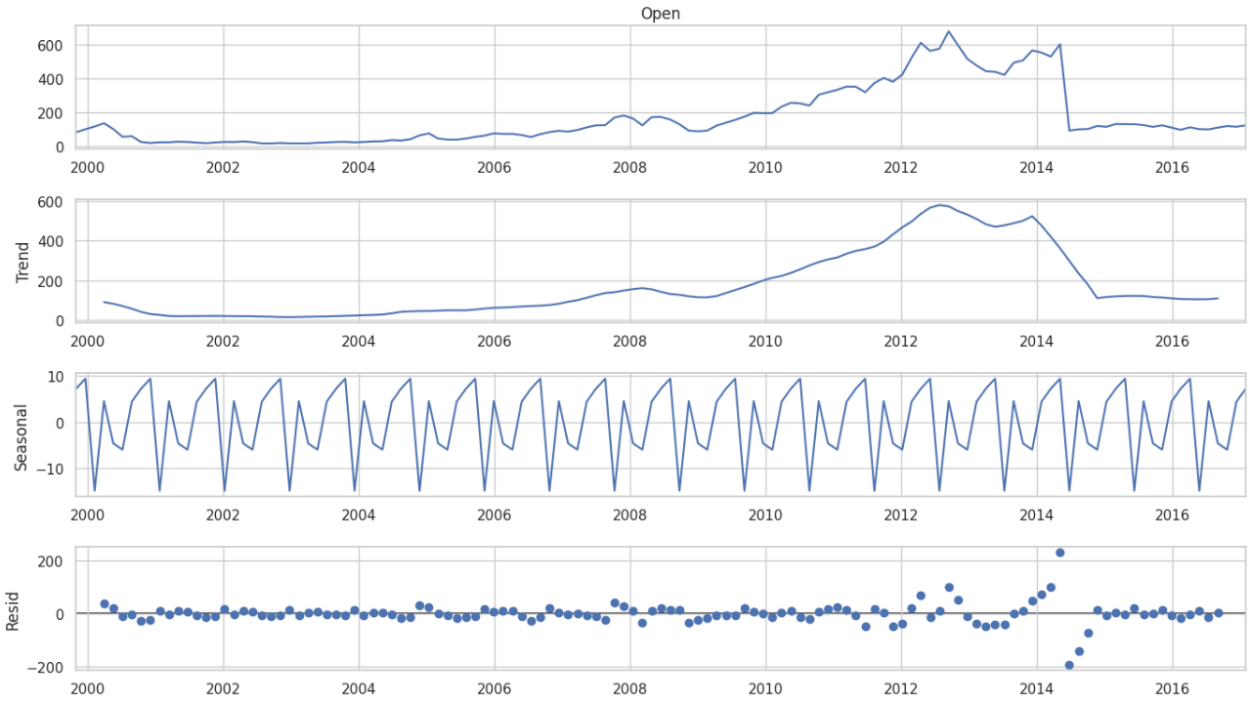
```
[ ] plt.figure(figsize=(14,4))
    plt.plot(Apple_sales['High'])
    plt.title('Apple Stock Highest Price', fontsize=20)
    plt.ylabel('Price', fontsize=16)
```

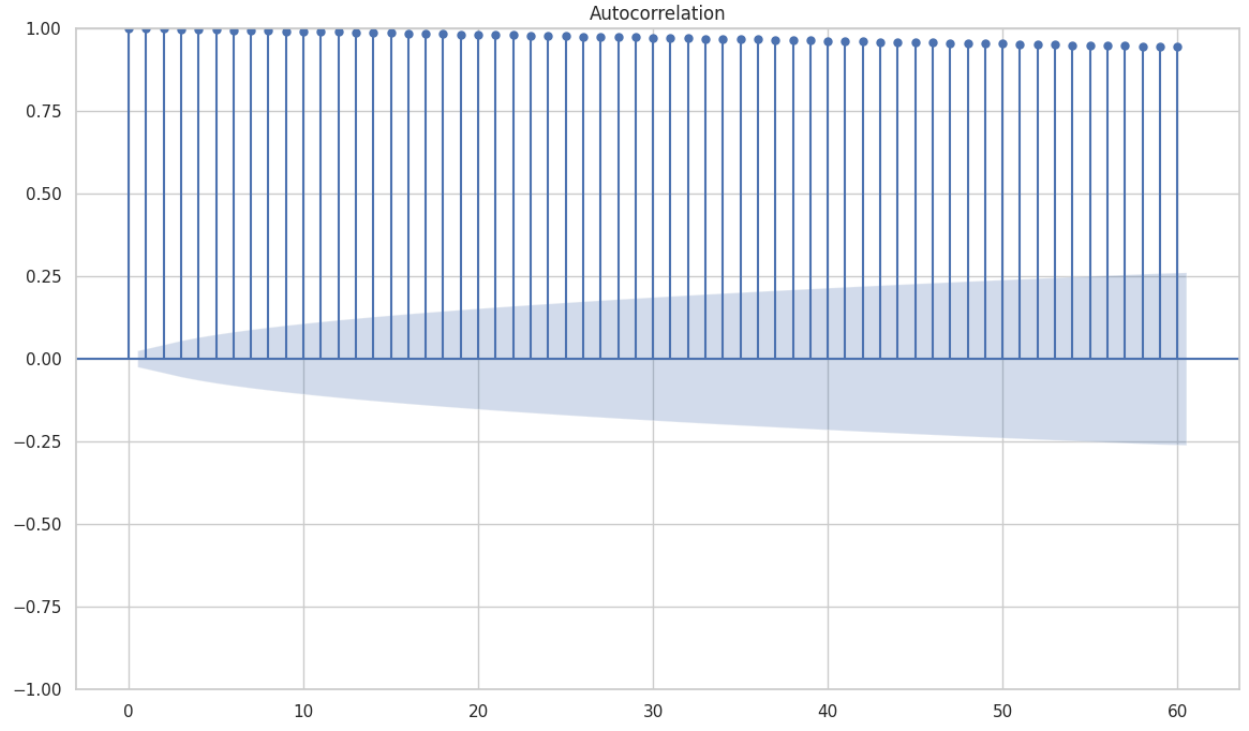
```
Text(0, 0.5, 'Price')
```


```
[ ] plt.figure(figsize=(14,4))
    plt.plot(Apple_sales['Low'])
    plt.title('Apple Stock Lowest Price', fontsize=20)
    plt.ylabel('Price', fontsize=16)
```

```
Text(0, 0.5, 'Price')
```

STL DECOMPOSITION





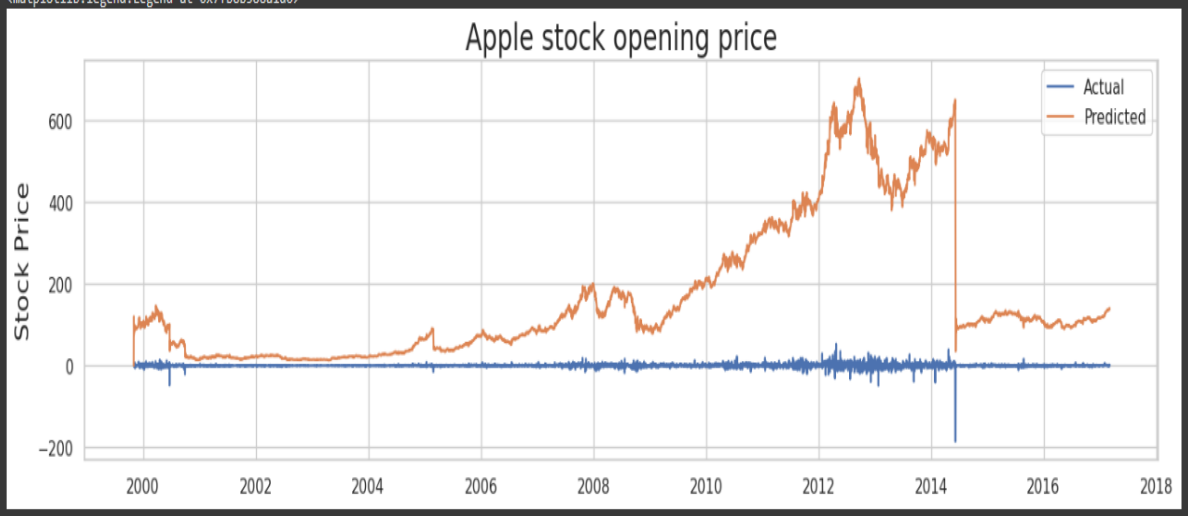
The STL (Seasonal and Trend decomposition using Loess) we get that the data is seasonal with 12 months as the period or 1 year

Prediction of Apple's Stock Opening Prices using ARIMA Model:

```
[ ] arima = ARIMA(Apple_sales_train['Open'], order=(12,2,2))
    predictions = arima.fit().predict()
```

```
plt.figure(figsize=(16,4))
plt.plot(Apple_sales_train['Open'].diff(), label="Actual")
plt.plot(predictions, label="Predicted")
plt.title('Apple stock opening price', fontsize=20)
plt.ylabel('Stock Price', fontsize=16)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb8b588a1d0>
```



SARIMAX Results

Dep. Variable:	Open	No. Observations:	6335
Model:	ARIMA(12, 2, 2)	Log Likelihood	-19301.776
Date:	Mon, 28 Nov 2022	AIC	38633.551
Time:	17:10:19	BIC	38734.854
Sample:	11-01-1999 - 03-05-2017	HQIC	38668.635
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.6556	0.068	-9.705	0.000	-0.788	-0.523
ar.L2	0.4415	0.022	20.438	0.000	0.399	0.484
ar.L3	-0.0139	0.009	-1.511	0.131	-0.032	0.004
ar.L4	-0.1065	0.015	-7.193	0.000	-0.136	-0.078
ar.L5	0.0324	0.007	4.487	0.000	0.018	0.047
ar.L6	0.0062	0.011	0.541	0.589	-0.016	0.028
ar.L7	0.0346	0.013	2.756	0.006	0.010	0.059
ar.L8	0.0051	0.014	0.374	0.708	-0.022	0.032
ar.L9	-0.0471	0.013	-3.601	0.000	-0.073	-0.021
ar.L10	-0.0113	0.014	-0.796	0.426	-0.039	0.016
ar.L11	-0.0081	0.016	-0.508	0.611	-0.039	0.023
ar.L12	-0.0191	0.014	-1.408	0.159	-0.046	0.007
ma.L1	-0.0240	0.068	-0.352	0.725	-0.157	0.109
ma.L2	-0.9759	0.068	-14.432	0.000	-1.108	-0.843
sigma2	25.9586	0.081	319.829	0.000	25.800	26.118

Ljung-Box (L1) (Q):	0.00	Jarque-Bera (JB):	38065928.75
Prob(Q):	1.00	Prob(JB):	0.00
Heteroskedasticity (H):	16.76	Skew:	-11.17
Prob(H) (two-sided):	0.00	Kurtosis:	382.15

This is the representation of the actual data taken in 1st order difference form and the orange one is the predicted by the Arima model and fits the training set very well

```
[ ] plt.figure(figsize=(16,4))
    plt.plot(Apple_sales_test['Open'], label="Actual")
    plt.plot(test_predictions, label="Predicted")
    plt.title('Apple stock opening price', fontsize=20)
    plt.ylabel('Stock Price', fontsize=16)
    plt.legend()

<matplotlib.legend.Legend at 0x7fb8b5716110>
```

This is the representation of predicted data(in orange) and actual data(in blue) of the test data

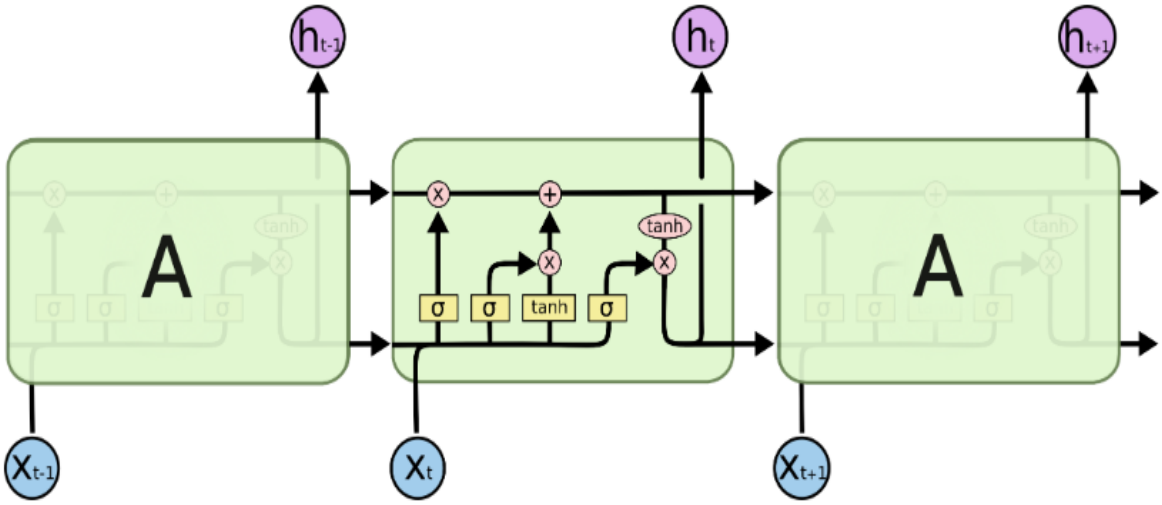
We also use the Auto-ARIMA Model to train the data:

```
[ ] plt.figure(figsize=(16,4))
    plt.plot(Apple_sales_test['Open'], label="Actual")
    plt.plot(forecast, label="Predicted")
    plt.title('Apple stock opening price', fontsize=20)
    plt.ylabel('Stock Price', fontsize=16)
    plt.legend()

<matplotlib.legend.Legend at 0x7fb8af33b790>
```

Forecasting Models: LSTMs (Long Short-Term Memory)

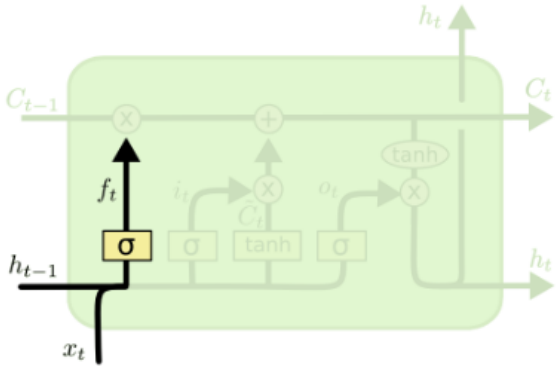
Long Short Term Memory networks – usually just called LSTMs – are a special kind of RNN, capable of learning long-term dependencies. According to Korstanje: "The LSTM cell adds long-term memory in an even more performant way because it allows even more parameters to be learned. This makes it the most powerful [Recurrent Neural Network] to do forecasting, especially when you have a longer-term trend in your data. LSTMs are one of the state-of-the-art models for forecasting at the moment"

So LSTMs are basically RNNs which are exclusively designed to avoid the long-term dependency problems of RNNs. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

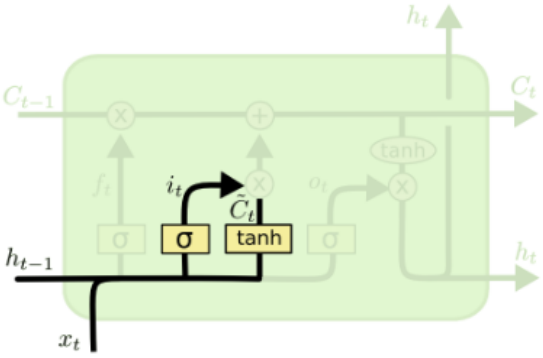


The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the **forget gate layer**.

It outputs a number between 0 and 1 for each number in the cell state, where 0 represents forgetting the data while 1 represents keeping the data. Now, a *tanh* layer creates a vector of new candidate values, C_t', that could be added to the state. In the next step, these are combined to create an update to the state. Now the old cell state is updated to the new one. Finally, the output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through *tanh* (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

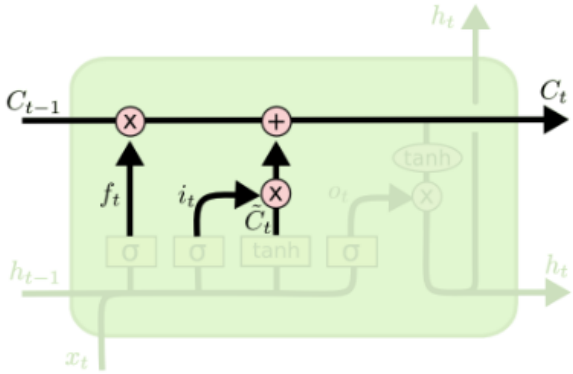


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

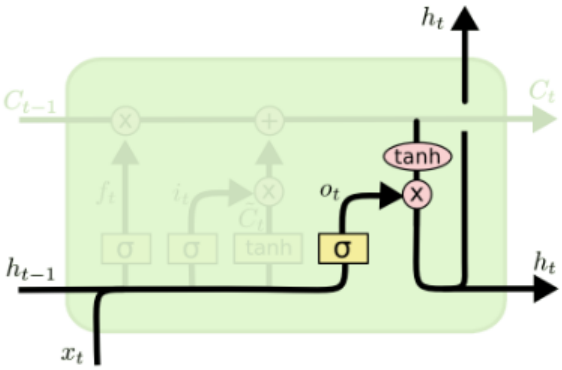


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



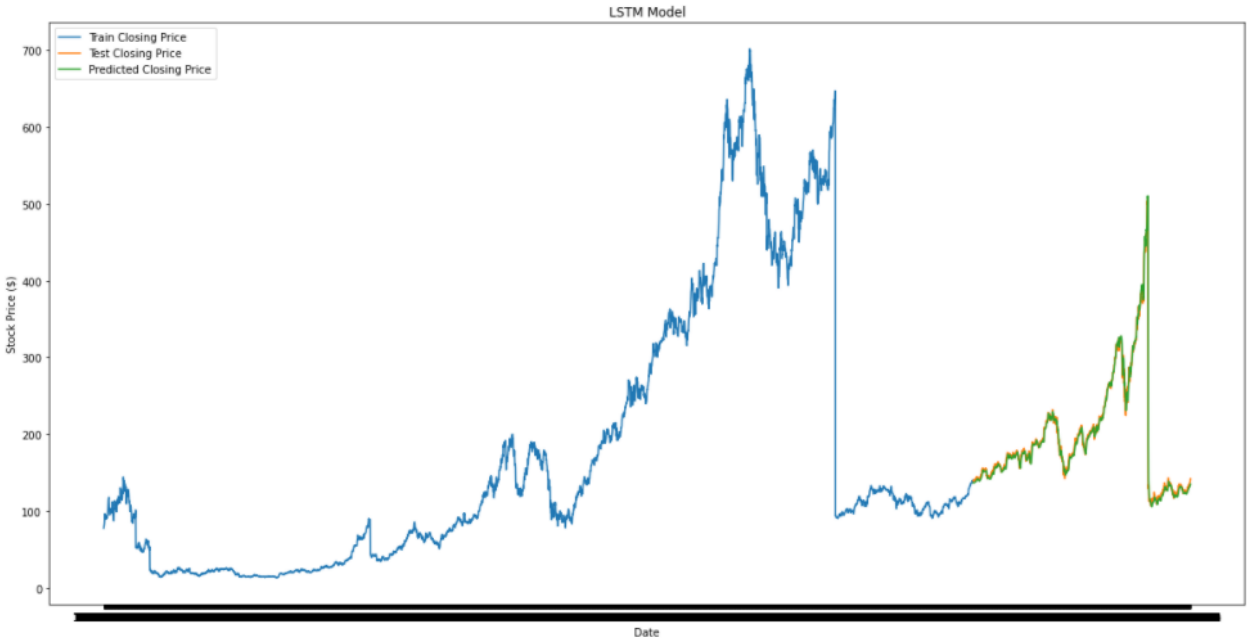
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Prediction of Closing stock prices using LSTM Model:



Forecasting Models: Simple Moving Average and Exponential Moving Average

Simple Moving Average:

A simple moving average (SMA) is an arithmetic moving average calculated by adding recent prices and then dividing that figure by the number of time periods in the calculation average. Short-term averages respond quickly to changes in the price of the underlying security, while long-term averages are slower to react.

$$\text{SMA} = \frac{A_1 + A_2 + ... + A_n}{n}$$

where:

A_n = the price of an asset at period n

n = the number of total periods

Exponential Moving Average:

An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points. The exponential moving average is also referred to as the exponentially weighted moving average.

$$EMA_{\text{Today}} = \left(\text{Value}_{\text{Today}} * \left(\frac{\text{Smoothing}}{1 + \text{Days}}\right)\right) + EMA_{\text{Yesterday}} * \left(1 - \left(\frac{\text{Smoothing}}{1 + \text{Days}}\right)\right)$$

where:

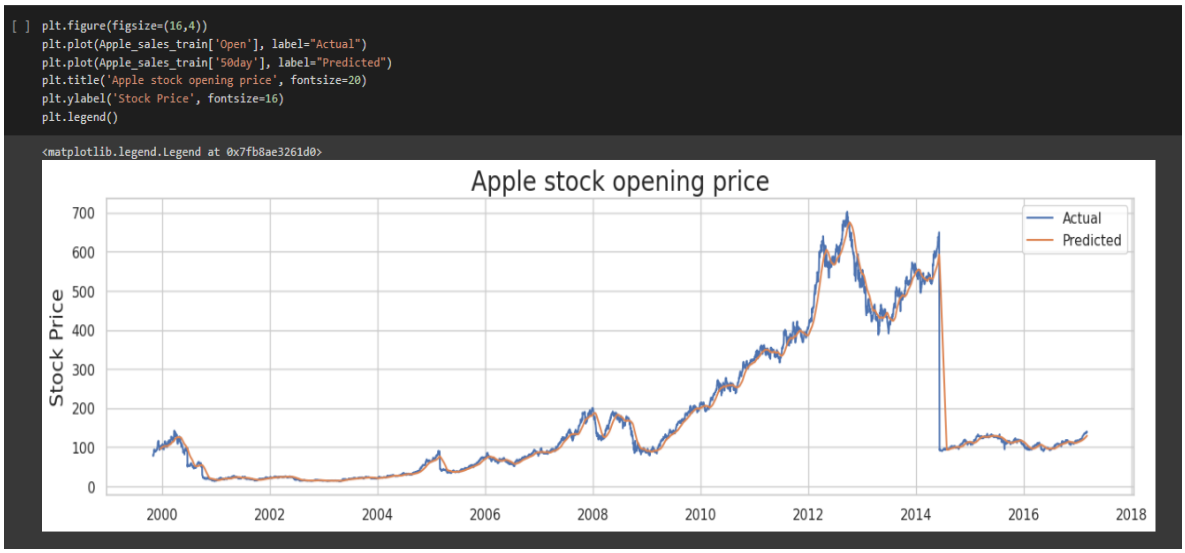
EMA = Exponential moving average

Simple Moving Average vs. Exponential Moving Average

The major difference between an exponential moving average (EMA) and a simple moving average is the sensitivity each one shows to changes in the data used in its calculation. More specifically, the EMA gives a higher weighting to recent prices, while the SMA assigns an equal weighting to all values.

The two averages are similar because they are interpreted in the same manner and are both commonly used by technical traders to smooth out price fluctuations. Since EMAs place a higher weighting on recent data than on older data, they are more reactive to the latest price changes than SMAs are, which makes the results from EMAs more timely and explains why the EMA is the preferred average among many traders.

Prediction using Simple Moving Average:



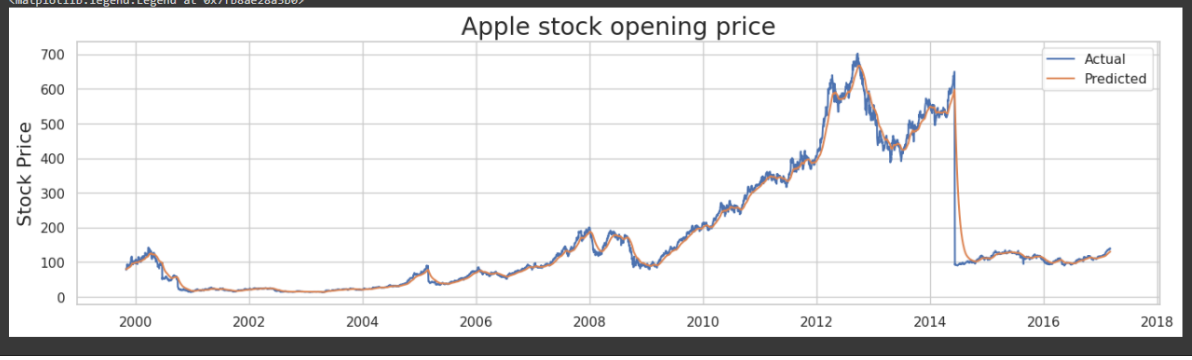
This is the representation of actual data and predicted data from the train data using the sma model.

```
[ ] plt.figure(figsize=(16,4))
    plt.plot(Apple_sales_test['Open'], label="Actual")
    plt.plot(Apple_sales_test['50day'], label="Predicted")
    plt.title('Apple stock opening price', fontsize=20)
    plt.ylabel('Stock Price', fontsize=16)
    plt.legend()

<matplotlib.legend.Legend at 0x7fb8ae24c430>
```

This is the representation of actual data and predicted data from the test data using the SMA model.

Predictions using Exponential Moving Average Model (EMA):

```
[ ] plt.figure(figsize=(16,4))
    plt.plot(Apple_sales_train['Open'], label="Actual")
    plt.plot(Apple_sales_train['50day_EMA'], label="Predicted")
    plt.title('Apple stock opening price', fontsize=20)
    plt.ylabel('Stock Price', fontsize=16)
    plt.legend()

<matplotlib.legend.Legend at 0x7fb8ae28a3b0>
```



This is the representation of actual data and predicted data from the train data using the EMA model.

Apple_sales_test

Date	Low	High	Close	Open	200day	50day	50day_EMA
2017-03-06	138.59590	139.7700	139.340	139.3650	116.913075	139.365000	123.113488
2017-03-07	138.79000	139.9800	139.520	139.0600	117.064525	139.060000	123.113488
2017-03-08	138.82000	139.8000	139.000	138.9500	117.215275	138.950000	123.113488
2017-03-09	137.05000	138.7900	138.680	138.7400	117.364825	138.740000	123.113488
2017-03-10	138.64000	139.3571	139.140	139.2500	117.516775	139.250000	123.113488
...	...	...	...	...	...	...	...
2021-07-02	137.74500	140.0000	139.960	137.9000	129.255600	445.692451	123.113488
2021-07-03	138.32625	140.7875	140.475	138.4425	129.326112	446.033122	123.113488
2021-07-04	138.90750	141.5750	140.990	138.9850	129.383987	446.374054	123.113488
2021-07-05	139.48875	142.3625	141.505	139.5275	129.437125	446.715246	123.113488
2021-07-06	140.07000	143.1500	142.020	140.0700	129.492675	447.056699	123.113488

This is the representation of actual data and predicted data from the test data using the ema model.

Overall Analysis and Conclusion:

RMSE:

The root-mean-square error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. The RMSE represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences.

$$RMSE = \sqrt{\frac{1}{N} * \sum_{t=1}^{N} (At - Ft)^2}$$

MAPE:

MAPE is the mean absolute percentage error, which is a relative measure that essentially scales MAD to be in percentage units instead of the variable's units. Mean absolute percentage error is a relative error measure that uses absolute values to keep the positive and negative errors from canceling one another out and uses relative errors to enable you to compare forecast accuracy between time-series models.

$$MAPE = \frac{1}{N} * \sum_{t=1}^{N} \left| \frac{At - Ft}{At} \right|$$

Comparison of SMA, EMA, LSTM:

Name of the model	SMA	EMA	ARIMA	LSTM
MAPE	52.618	31.202	25.768	2.381

RMSE	125.747	101.946	87.683	12.647

From the analysis, we observe that model accuracy wise (best to worst): LSTMs > ARIMA > EMA > SMA

We conclude from the above analysis that LSTM works better if we are dealing with huge amounts of data and enough training data is available, while ARIMA is better for smaller datasets. Because the training will be better when we use a neural network and LSTMs are basically special types of RNNs. In case of smaller data sets, LSTMs aren't required because they consume too much memory. Also, ARIMA models are linear and LSTM models are nonlinear.

ARIMA requires a series of parameters (p,q,d) which must be calculated based on data, while LSTM does not require setting such parameters. However, there are some hyperparameters we need to tune for LSTM.