NASDAQ Stock Exchange - Predicting Stock prices using Machine Learning Techniques (Time Series Forecasting)

Akarsh Asokan / M00740829 - CST4050

Introduction

Forecasting has played a pivotal role in decision making with its applications ranging from Business platforms, to Scientific,Industrial, Economic and Commerical activities. Time Series Forecasting in particular deals with forecasting on data or observations that are bound by a time sequence. This could be Weather forecasting, Energy consumption, Product Sales predictions etc. (L. Yermal and P. Balasubramanian, 2017)

This particular project will focus on one aspect of it, which is Stock market predictions(Closing price) based on a 5 year historical data of Big Tech companies namely Amazon, Apple, Google, Microsoft and Facebook. These are some of the highest performers in the NASDAQ stock exchange which saw the emergence of "Data being the new oil" outperforming the traditional energy giants. (Kantrowitz, 2014)

Stock markets provide a closer outlook onto how an economy performs. It easily shows the peformance of an economy, region, country or business type and represents the trend of capital flow. This is not a question of numerical accuracy when it comes to predictions, but also learn to include investor sentiments that may be personal, political scenarios, technology or market changes which all are factors that affect the stock market prices. Concentrating on the closing price for prediction is because it best reflects the buying and selling of NASDAQ stock prices. (L. Yermal and P. Balasubramanian, 2017) (Friedman 2014)

Closing price prediction will be for the best performing tech company based on our hypothesis.

Objective : Closing price Prediction for a Tech company stock for 2020-21 period.

Data Source: Yahoo Finance 5Y Historical Data from 04/23/2015 to 04/23/2020

Data Fields and its description:

Dates: Ranging from 2015 to 2020 , 23rd April . Set as index for the time series data.

Open price: The first transaction for the day for the company stocks.

Close price: The last transaction for the day for the company stocks.

High price: The highest price of the company stocks for the day.

Low price: The lowest price of the company stocks for the day.

Volume : Volume of stocks traded for the company for the day. Higher the stock volume, higher is its liquidity.

Important Points to note regarding Stock Trading (https://www.investopedia.com/ (https://www.investopedia.com/)) :

Stock trading takes place even in after hours and hence there is no guarantee that the closing price of a day could be the opening price for the next day. They are interdependent on the stock investors sentiment. There are cases where companies tend to make major announcements post market closing , which affects the stock placements for the next day as well.

Bullish Trends :

When the investors believe their stocks will go higher.

Bearish Trends:

When the investors believe their stocks will drop.

When the opening price for the day is closest to the highest price and the closing price is close to the low price, the trend is Bearish.

When the closing price for the day is closest to the highest price and the opening price is close to the low price, the trend is Bullish.

Moving Averages:

It is a constantly updated average stock price over a particular period of time (historical data) which differs with every time period making moving average method more receptive to the evolving stock prices. For example each day average would be a contribution of previous values dropping the oldest value from the group while considering the average.

Literary Review for the Machine Learning Algorithms Involved :

The dataset is split into validation and training set where the 4 year historical data is considered the training set and last year to present time frame as the validation set. The traditional train test split method of randomizing the data is not advantageous due to the time series frame involved. We are trying to predict the future trends based on the actual historical data from the past.

Baseline Algorithm (Moving Averages):

The baseline to compare the other algorithms tested in the project is done by implementing Moving averages method to the selected dataset. The dataset is split into validation and training set as mentioned earlier.

Applying the moving average method which means the average for each day would involve dropping th previously considered oldest value in the set. As mentioned earlier, Moving averages are generally considered advantageous over simple averaging techinques due to the time series nature of our data. Further down the section, while implementing the algorithm on the dataset we may evaluate the pros and cons of the model.

Auto-Regressive Integrated Moving Averages (ARIMA) :

ARIMA is a linear approach towards statistical forecasting like in the case of stock market predictions. It is a combination of Moving Averages(MA) integrated with Auto regression (AR) , the linear pattern considered here would be the Closing price for the varaince in time for the 5 year period which is like a univariate time series for which ARIMA works perfectly.

AR: Auto regressive or the dependent nature of the observations to its previous observations. It differs from the Moving average model in terms of a unit root presence.

I: Replacing datapoints with the differences between the observation to its previous ones , making the time series stationary.

MA: Representing the regression residuals over the period of time as discussed previously.

Each of these components are given as parameters in the model which can be configured to do either AR,I, or MA as needed.

The parameters are namely:

p: No of time lags called the lag order.

d: No of times the observation differences took place, ie the degree of differencing.

q: the order of moving average.

The major disadvantage of the model is that it does not include the seasonality or other unpredictable categories that is needed for stock price predictions realistically.

Auto ARIMA is chosen for the model as it selects the parameters required for the algorithms automatically.

Algorithmic performance will be compared with the baseline and discussed on execution.

(L. Yermal and P. Balasubramanian,2017)

Long Short Term Memory (LSTM)

Recurrent Neural Networks in Deep learning where the classes of variable may exhibit temporal dynamic behaviour as in the case of a time series where we make sense of a temporal sequence. LSTM is a RNN that can deal with gradient vanishing problem as there can be lags of unknown duration between important events in a time series.

A benefit of this type of network is that it retains information of long sequential data and learns from it.The forget gate helps in removing those information thats not needed anymore for developing data (apart from the input , output gate). For this you need to set the the “stateful” argument to “True” when defining an LSTM layer. Reset_states()function helps in managing the LSTM batch size for the sample inputs which is cleared by default.

The LSTM layer has a matrix structure: [samples, time steps, features]

Samples: data observations in rows Time steps for a given obesvation Features: Specific measures observed at the time of observation.

The major disadvantages of using LSTM would be its time lag in predictions, and also requirement of bigger datasets. It might not be recquired on univariate time series forecasting where perhaps ARMA models could perform better.

(Dou Wei, 2019),(Gers F.A., Eck D. & Schmidhuber J. (2001))

Prophet from Facebook

Prophet is a statistical approach to forecasting developed by Facebook that takes seasonality, trends and holidays into accounts while doing time forecasting. They have changes on a monthly , weekly to yearly automated forecasting using the portion of time series thats required for the point. With prophet you can add the domain expertise of the concerned person without much knowledge in statistics making it user friendly. Important turn of events maybe mentioned that can affect the data in anyway the domain expert deems relevant. The ability to apply judgements to the forecasts while still retaining the statistical forecasting methods is what makes prophet a great tool.

The disadvantages of Prophet is that it might be too seasonal or trend oriented more than implementing the time series forecasting.

Algorithm performance and comparisons can be related in Model Evaluation and Inference stage.

(Taylor, Sean ; Letham, Benjamin (2017))

Hypothesis:

1) Highest performer from the tech company list is Amazon.

These can be identified from Historical data visualizations.

2) Overall stock performance is lower on the first and last working day (Monday and Friday).

These can be identified from Historical data visualizations and proved further by algorithms like prophet.

3) Stock performance boosted since Donald trump took U.S. presidential office.

These can be identified from Historical data visualizations

4) Stock performance post Corona pandemic is still good for Amazon.

These can be identified from Historical data visualizations and proved further by algorithms like prophet.

5) Stock performance of second and third quarter of a year should outweigh the first and last quarter.

These can be identified from Historical data visualizations and proved further by algorithms like prophet.

Data loading and preprocessing

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

Datasets are imported into the kernel with index as dates and parse_dates with the column 'Date'.

In [2]:

```
AMZN= pd.read_csv(r"C:\Users\akars\Downloads\AMZN.csv",index_col='Date', parse_dates=[
'Date'])
GOOGL=pd.read_csv(r'C:\Users\akars\Downloads\GOOGL.csv',index_col='Date', parse_dates=[
'Date'])
MSFT=pd.read_csv(r'C:\Users\akars\Downloads\MSFT.csv',index_col='Date', parse_dates=['D
ate'])
AAPL=pd.read_csv(r'C:\Users\akars\Downloads\AAPL.csv',index_col='Date', parse_dates=['D
ate'])
FB=pd.read_csv(r'C:\Users\akars\Downloads\FB.csv',index_col='Date', parse_dates=['Date'
])
```

In [3]:

```
AMZN.head()
```

Out[3]:

Date	Open	High	Low	Close	Adj Close	Volume
2015-04-23	390.209991	391.880005	386.149994	389.989990	389.989990	7980000
2015-04-24	439.000000	452.649994	439.000000	445.100006	445.100006	17176900
2015-04-27	443.859985	446.989990	437.410004	438.559998	438.559998	5430900
2015-04-28	438.510010	439.000000	428.040009	429.309998	429.309998	4140500
2015-04-29	426.750000	434.239990	426.029999	429.369995	429.369995	3621700

In [4]:

```
AMZN.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 6 columns):
Open         1259 non-null float64
High         1259 non-null float64
Low          1259 non-null float64
Close        1259 non-null float64
Adj Close    1259 non-null float64
Volume       1259 non-null int64
dtypes: float64(5), int64(1)
memory usage: 68.9 KB
```

In [5]:

```
GOOGL.head()
```

Out[5]:

Date	Open	High	Low	Close	Adj Close	Volume
2015-04-23	550.409973	561.169983	550.080017	557.460022	557.460022	3909200
2015-04-24	580.049988	584.700012	568.349976	573.659973	573.659973	4608400
2015-04-27	572.770020	575.520020	562.299988	566.119995	566.119995	2403100
2015-04-28	564.320007	567.830017	560.960022	564.369995	564.369995	1858900
2015-04-29	560.510010	565.840027	559.000000	561.390015	561.390015	1681100

In [6]:

```
GOOGL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 6 columns):
Open         1259 non-null float64
High         1259 non-null float64
Low          1259 non-null float64
Close        1259 non-null float64
Adj Close    1259 non-null float64
Volume       1259 non-null int64
dtypes: float64(5), int64(1)
memory usage: 68.9 KB
```

In [7]:

```
MSFT.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 6 columns):
Open         1259 non-null float64
High         1259 non-null float64
Low          1259 non-null float64
Close        1259 non-null float64
Adj Close    1259 non-null float64
Volume       1259 non-null int64
dtypes: float64(5), int64(1)
memory usage: 68.9 KB
```

In [8]:

```
AAPL.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 6 columns):
Open         1259 non-null float64
High         1259 non-null float64
Low          1259 non-null float64
Close        1259 non-null float64
Adj Close    1259 non-null float64
Volume       1259 non-null int64
dtypes: float64(5), int64(1)
memory usage: 68.9 KB
```

In [9]:

```
FB.info()
```

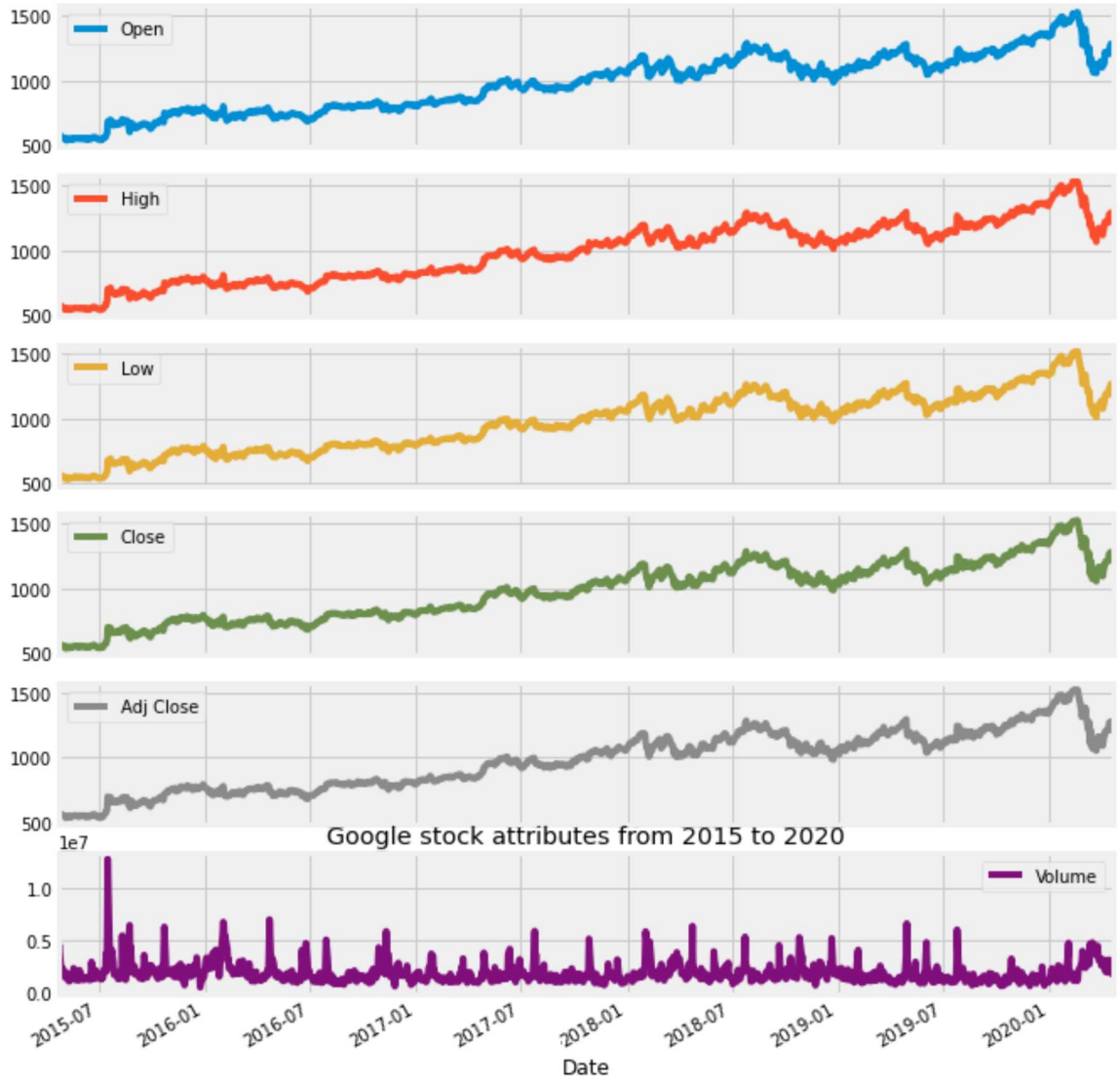
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 6 columns):
Open         1259 non-null float64
High         1259 non-null float64
Low          1259 non-null float64
Close        1259 non-null float64
Adj Close    1259 non-null float64
Volume       1259 non-null int64
dtypes: float64(5), int64(1)
memory usage: 68.9 KB
```

Exploratory Data Analysis and Feature Additions

Data visualizations of the individual tech giants prove that most of the price categories like open , close , high and low follow similar patterns in a days trajectory and hence coorelated. The theory in using only the closing price to identify the overall trend (Friedman 2014), is further cemented.

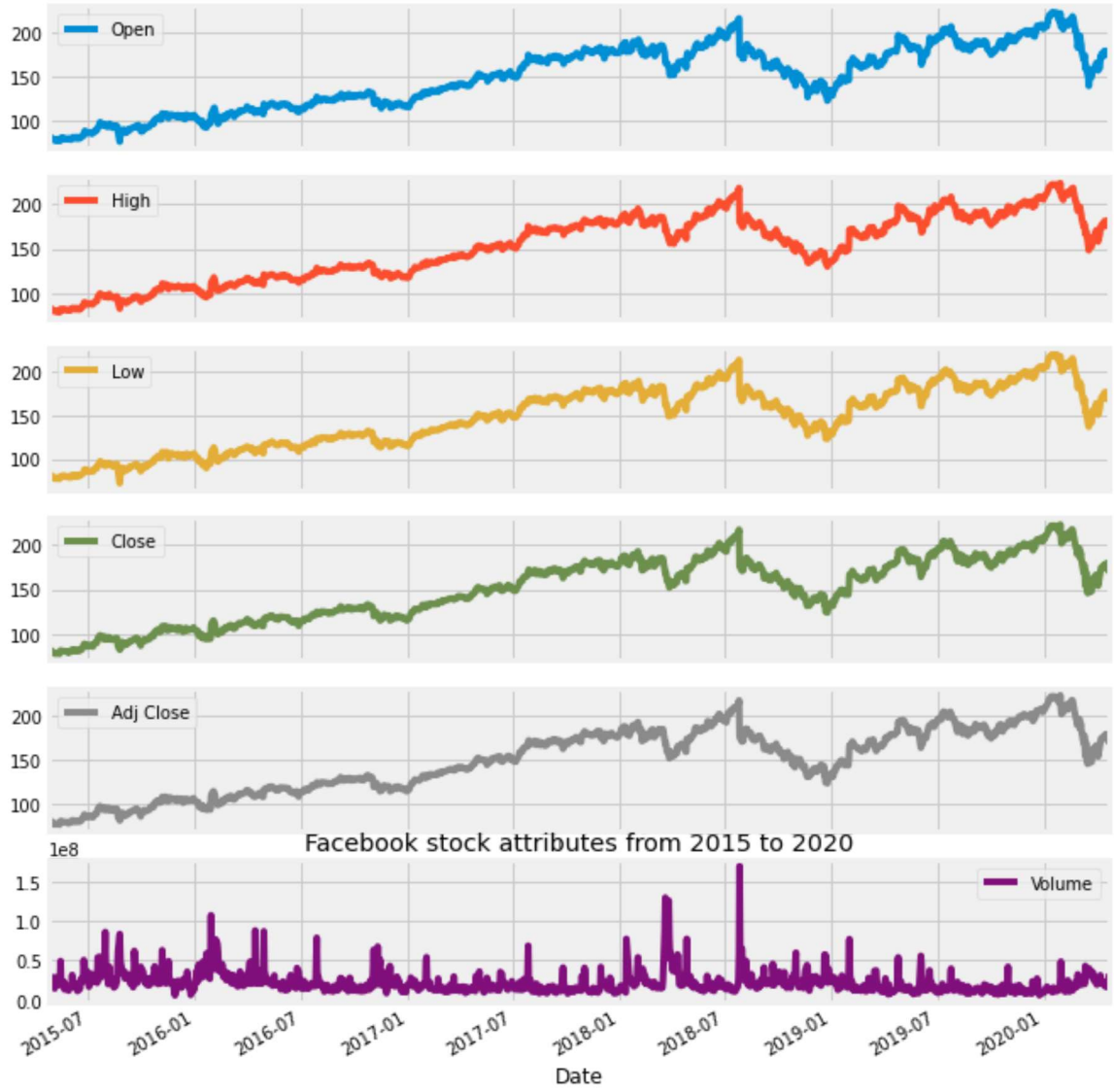
In [10]:

```
GOOGL['2015':'2020'].plot(subplots=True, figsize=(10,12))
plt.title('Google stock attributes from 2015 to 2020')
plt.show()
```


Open
High
Low
Close
Adj Close
1500
1000
500
Google stock attributes from 2015 to 2020
1e7
Volume
1.0
0.5
0.0
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01
2019-07
2020-01
Date

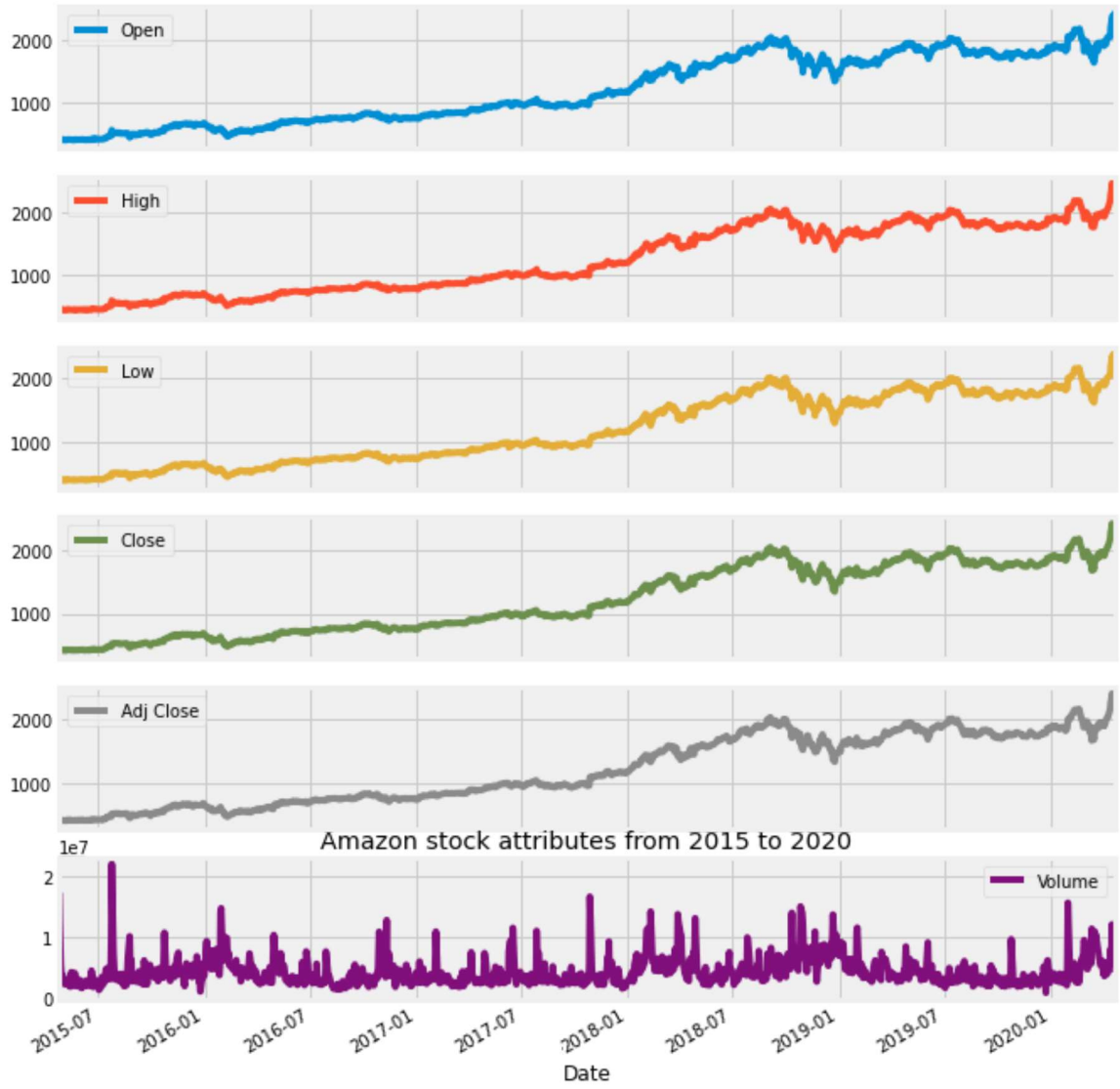
In [11]:

```
FB['2015':'2020'].plot(subplots=True, figsize=(10,12))
plt.title('Facebook stock attributes from 2015 to 2020')
plt.show()
```



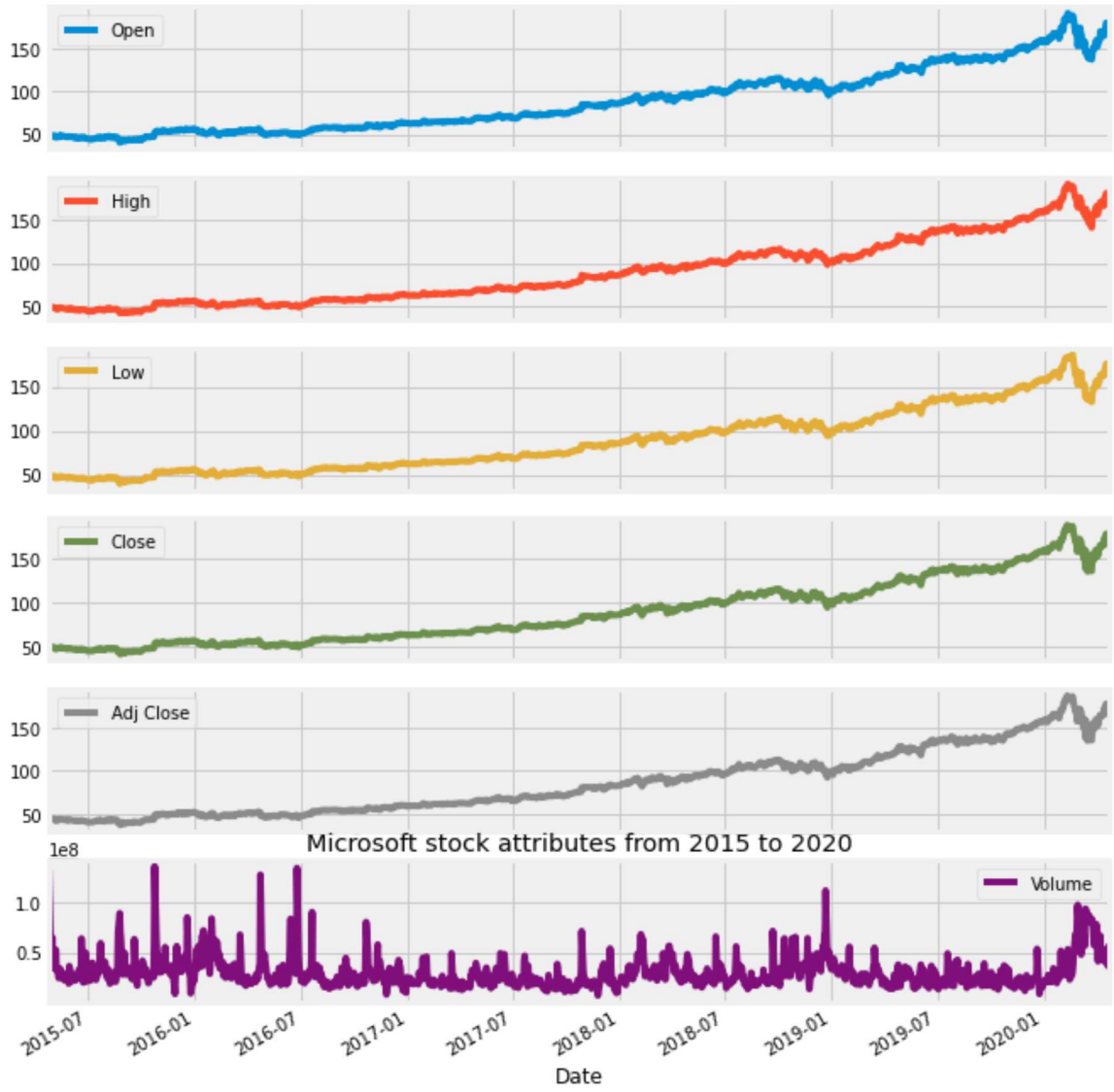
In [12]:

```
AMZN['2015':'2020'].plot(subplots=True, figsize=(10,12))
plt.title('Amazon stock attributes from 2015 to 2020')
plt.show()
```



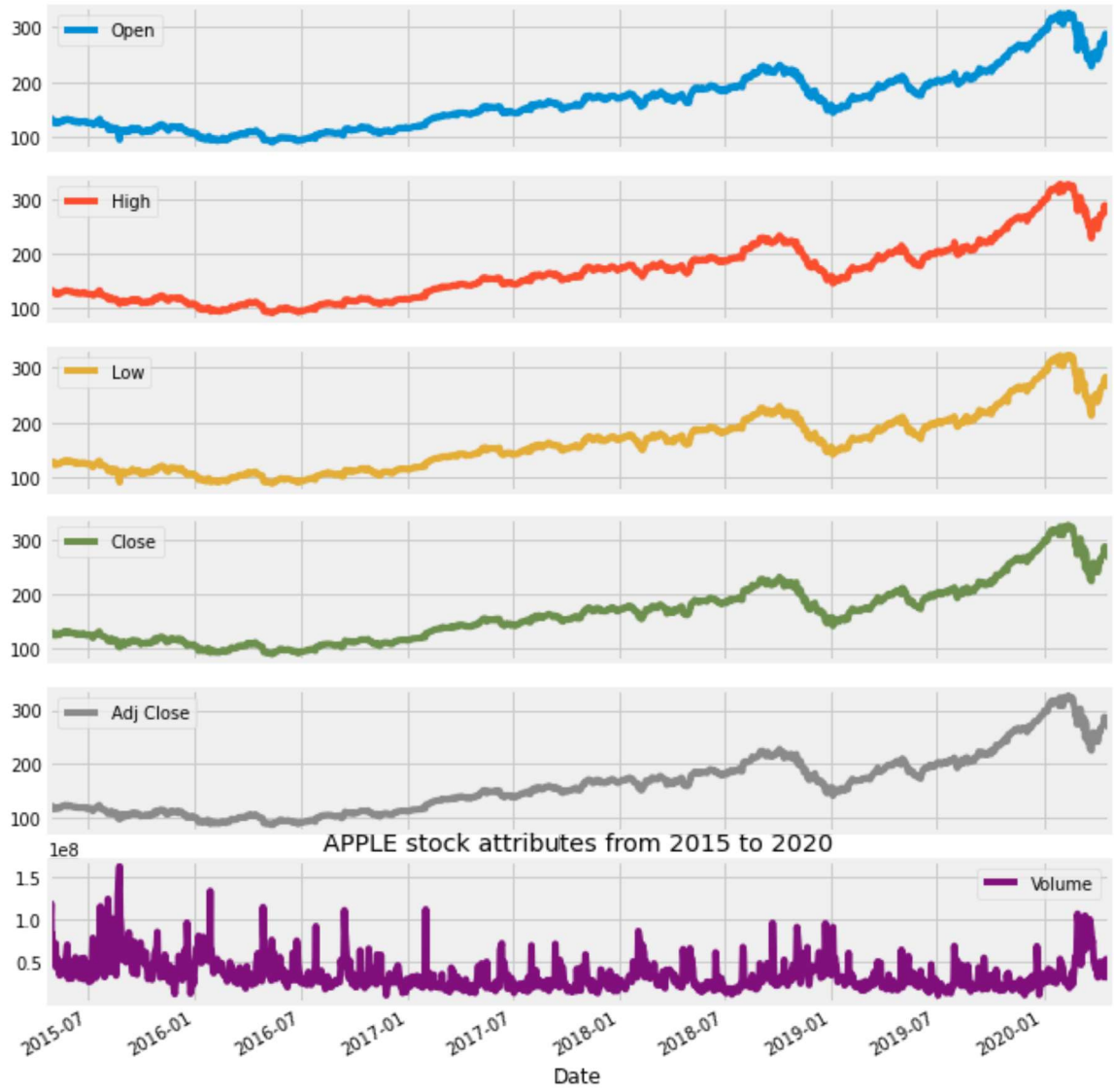
In [13]:

```
MSFT['2015':'2020'].plot(subplots=True, figsize=(10,12))
plt.title('Microsoft stock attributes from 2015 to 2020')
plt.show()
```



In [14]:

```
AAPL['2015':'2020'].plot(subplots=True, figsize=(10,12))
plt.title('APPLE stock attributes from 2015 to 2020')
plt.show()
```



All of the entities show a rise in their stock prices over the years , although they all commonly take a dip in the beginning 2019 before taking the upward trend again.

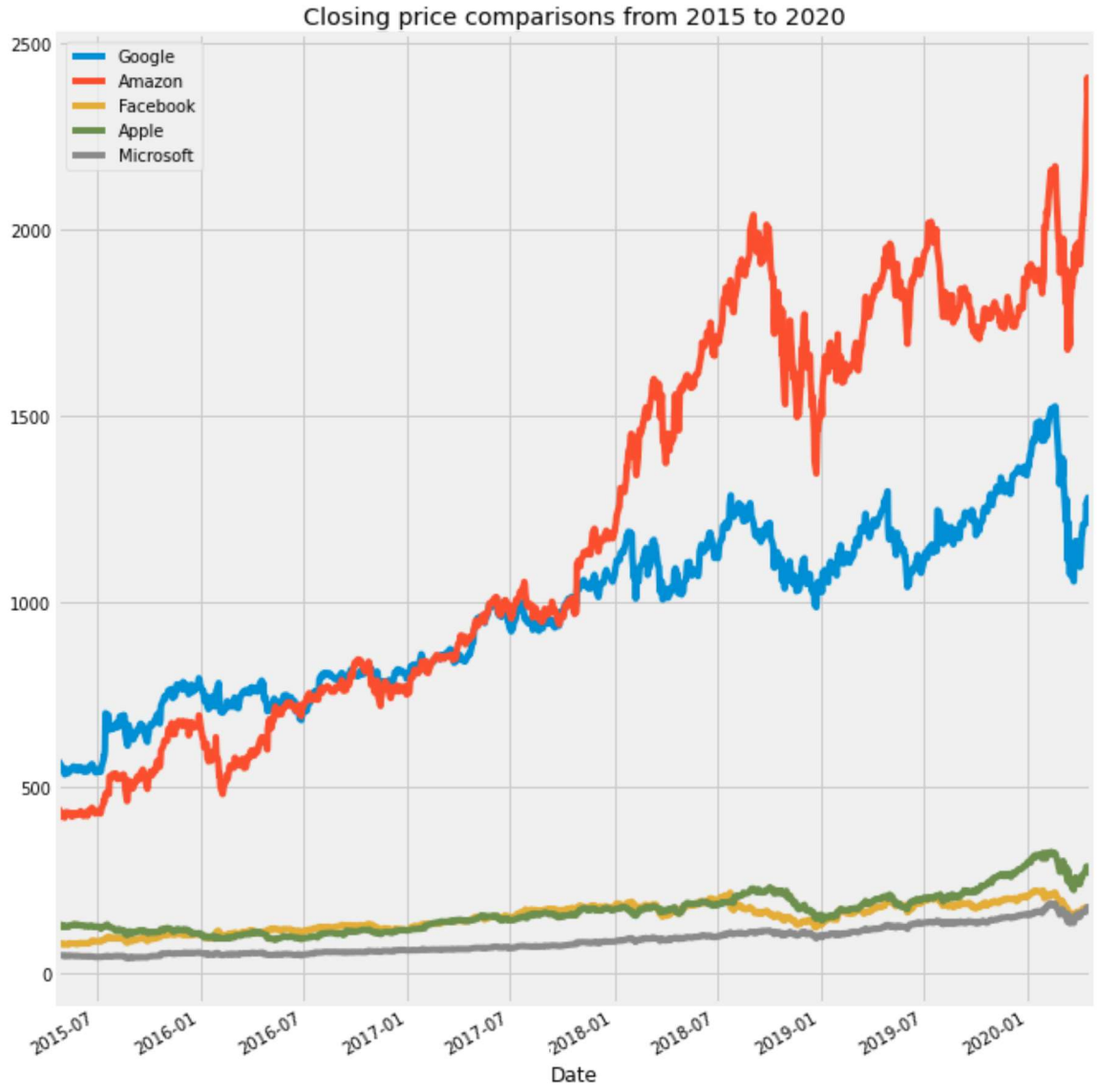
This is due to multiple factors with the overall S & P 500 index coming down due to Federal Reserve taking an interest hike, trading commodites hike on imported goods like aluminium and steel etc. Although the deciding factor particularly targeting tech giants would be all of them coming under scrutiny by the US congress for data privacy concerns , In particulary Facebook related to Cambridge Analytica.(Heawood, 2018) (Esteve,2017)

Resources: https://www.pbs.org/newshour/economy/making-sense/6-factors-that-fueled-the-stock-market-dive-in-2018 (https://www.pbs.org/newshour/economy/making-sense/6-factors-that-fueled-the-stock-market-dive-in-2018)

Lets compare the Close Price Comparisons between all the companies

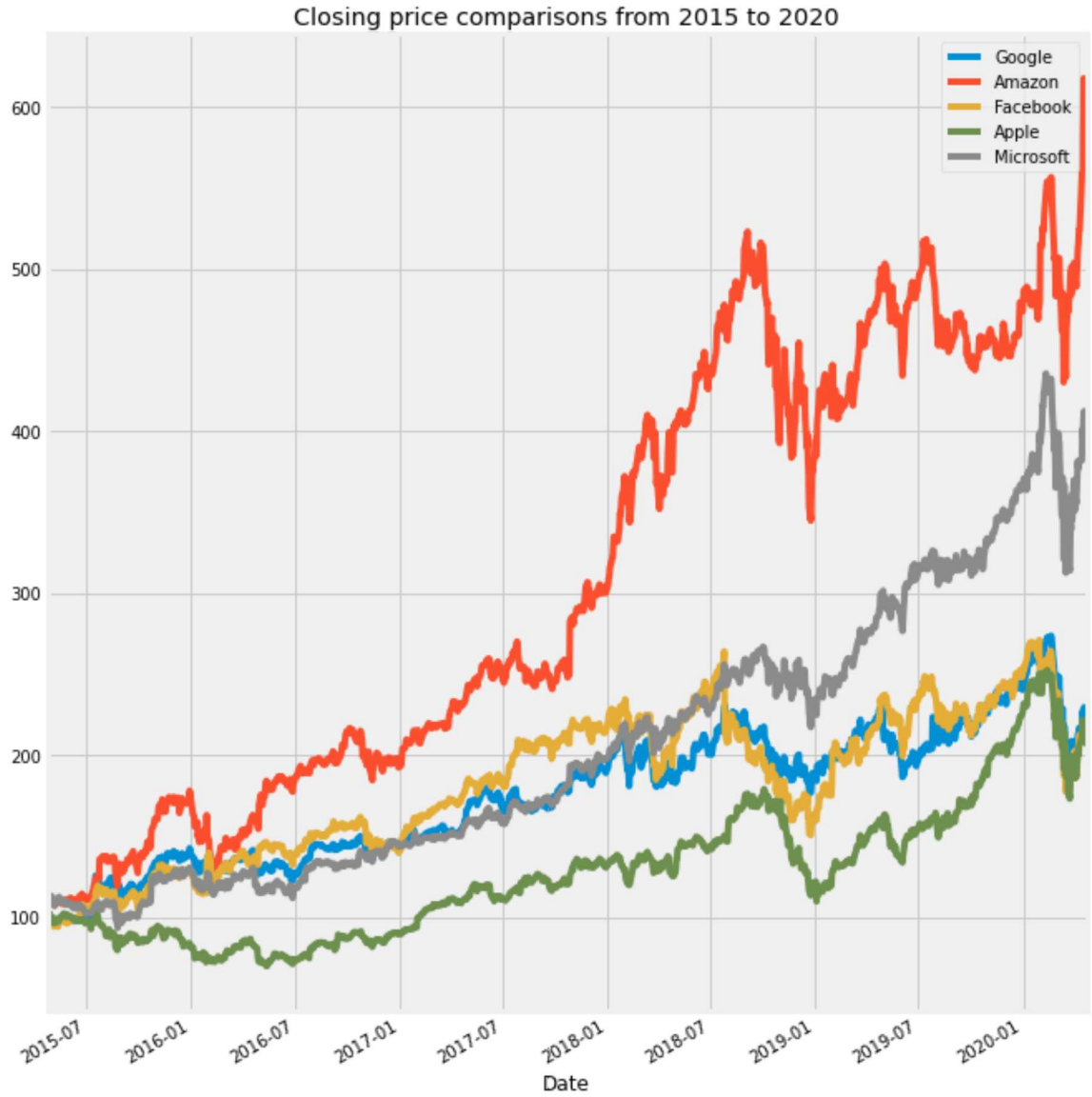
In [15]:

```
# Plotting before normalization
GOOGL.Close.plot(figsize=(10,12))
AMZN.Close.plot(figsize=(10,12))
FB.Close.plot(figsize=(10,12))
AAPL.Close.plot(figsize=(10,12))
MSFT.Close.plot(figsize=(10,12))
plt.legend(['Google','Amazon','Facebook','Apple','Microsoft'])
plt.title('Closing price comparisons from 2015 to 2020')
plt.show()
```


Closing price comparisons from 2015 to 2020
Google
Amazon
Facebook
Apple
Microsoft
2500
2000
1500
1000
500
0
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01
2019-07
2020-01
Date

In [16]:

```
# After Normalization
GOOGL_nm = GOOGL.Close.div(GOOGL.Close.iloc[0]).mul(100)
AMZN_nm = AMZN.Close.div(AMZN.Close.iloc[0]).mul(100)
FB_nm = FB.Close.div(FB.Close.iloc[0]).mul(100)
AAPL_nm = AAPL.Close.div(AAPL.Close.iloc[0]).mul(100)
MSFT_nm = MSFT.Close.div(MSFT.Close.iloc[0]).mul(100)

GOOGL_nm.plot(figsize=(10,12))
AMZN_nm.plot(figsize=(10,12))
FB_nm.plot(figsize=(10,12))
AAPL_nm.plot(figsize=(10,12))
MSFT_nm.plot(figsize=(10,12))
plt.legend(['Google','Amazon','Facebook','Apple','Microsoft'])
plt.title('Closing price comparisons from 2015 to 2020')
plt.show()
```



Hypothesis 1 is proved

Highest performer in terms of stocks is Amazon in the group, followed by Microsoft and Facebook.

Hypothesis 3 is proved

President Trump took office in January 20, 2017. Since then until date you can see the surge in Amazons stock which also includes other companies in the list, although not as high as Amazon achieved.

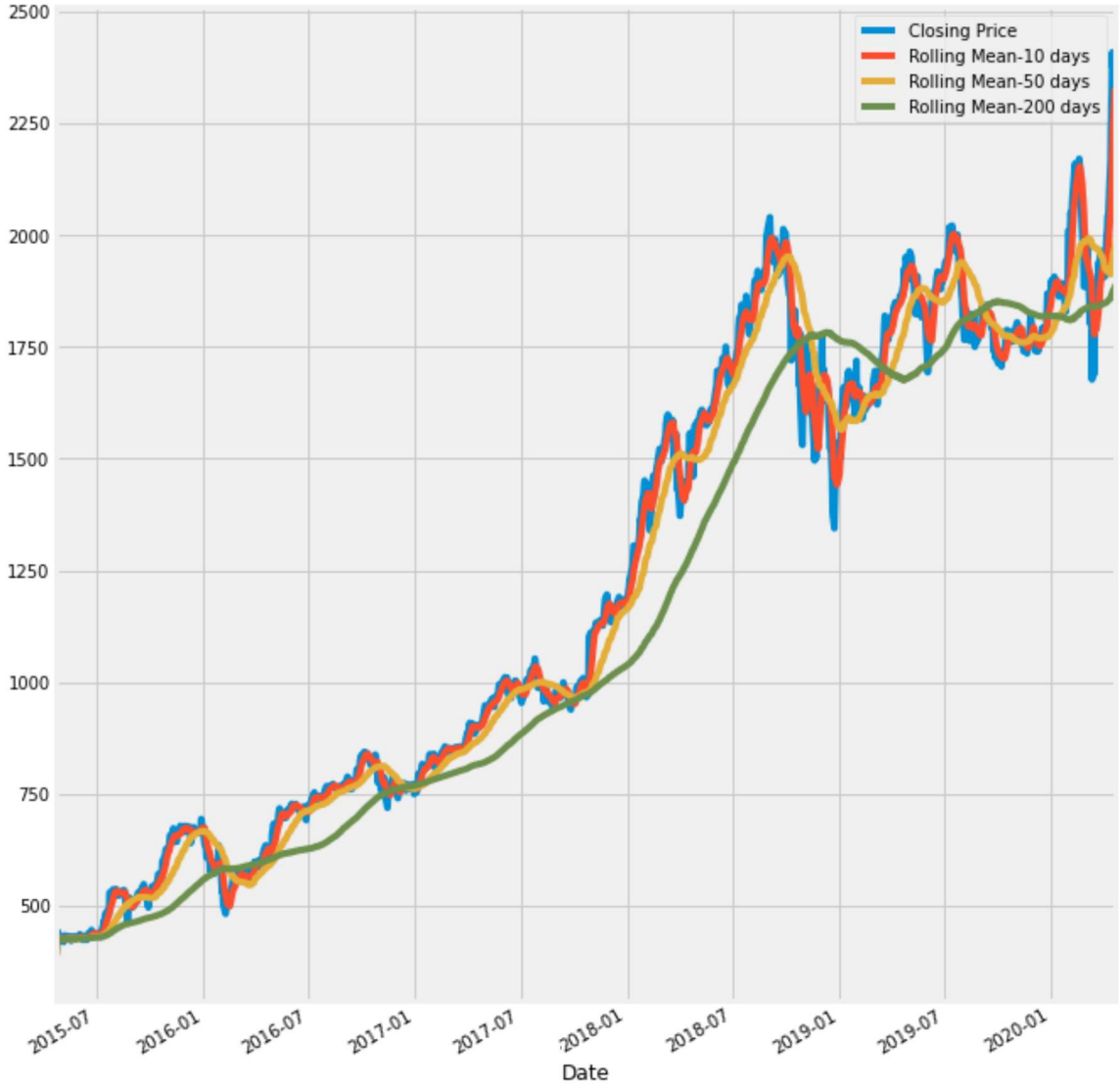
Microsoft is right behind Amazon and is another giant majorly due to its cloud computing business making headlines.

Facebook stocks although is right behind, fell in 2018 due to cambridge analytica and its public hearing.

As pointed out earlier, Last quarter of 2018 was depressing for all companies equally with S & P 500 taking a huge hit and Tech companies coming under scrutiny for data privacy.

In [17]:

```
#Rolling mean comparisons to 10 days, 60 days and 200 days average/closing price
rolling_am10 = AMZN.Close.rolling('10D').mean()
rolling_am50 = AMZN.Close.rolling('50D').mean()
rolling_am200 = AMZN.Close.rolling('200D').mean()
AMZN.Close.plot(figsize=(10,12))
rolling_am10.plot(figsize=(10,12))
rolling_am50.plot(figsize=(10,12))
rolling_am200.plot(figsize=(10,12))
plt.legend(['Closing Price','Rolling Mean-10 days','Rolling Mean-50 days','Rolling Mean
-200 days'])
# Plotting a rolling mean of 10 , 50 ,200 day window with original Closing price attrib
ute of Amazon stocks
plt.show()
```


Closing Price
Rolling Mean-10 days
Rolling Mean-50 days
Rolling Mean-200 days
2500
2250
2000
1750
1500
1250
1000
750
500
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
2018-07
2019-01
2019-07
2020-01
Date

Above you can see the Moving Averages of the 10-days, 50-days and 200-days rolling mean plotted along with the closing price. The Idea here is to buy or sell your stock based on the moving averages. The 10-day average is beneficial to short term trader whereas a 200 day moving average is for a long term trader giving insights to future trends than follwoing the closing price closely.

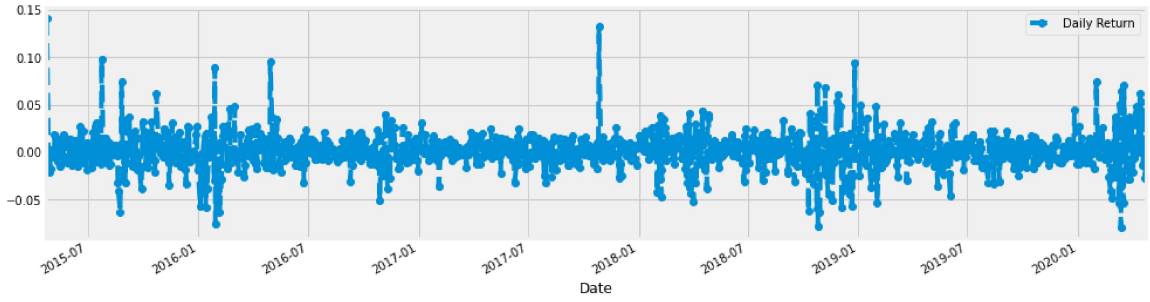
The strategy is to apply two moving averages to a chart: one longer and one shorter. When the shorter MA crosses above the longer-term MA, it's a buy signal known as the "golden cross" whereas vice versa is a sell signal which is called deadcross.

Resources: https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp (https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp)

Additional Feature of Daily Stock Returns is added below, which is the percentage change based on previous days closing price. Maximum percentage change is recorded on 0.15. A stock with lower positive and negative daily returns is preferrable contributing to its lesser risk associated with it factor. We can notice a trend here, that since 2020 there are larger daily fluctuations in daily returns signalling a doubt or 'volatility' on a daily basis in the market compared to the other years. This could be due to the corona virus pandemic dualling doubts to the investors. Although this still suggests Amazon is a short term stock performer if invested correctly (Hypothesis 4).

In [18]:

```
AMZN['Daily Return']=AMZN['Close'].pct_change()
AMZN['Daily Return'].plot(figsize=(15,4),legend=True,linestyle='--',marker='o')
plt.ioff()
```



In [19]:

```
AMZN.info() # except for the current day , all data on Daily Return is available.
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 7 columns):
Open            1259 non-null float64
High            1259 non-null float64
Low             1259 non-null float64
Close           1259 non-null float64
Adj Close       1259 non-null float64
Volume          1259 non-null int64
Daily Return    1258 non-null float64
dtypes: float64(6), int64(1)
memory usage: 118.7 KB
```

Adding an additional feature column mon_fri which returns 1 if the day is monday and friday in the working days and 0 if its tuesday,wednesday,thursday. This is done by creating a column day of week and assigning 1 for day of week 0 and 4.

In [20]:

```
# Adding a new column for differentiating between active working days, Monday and Frida
y.
am_df=AMZN.copy()
am_df['DayofWeek']= pd.DatetimeIndex(am_df.index).dayofweek
am_df['mon_fri'] = 0
for i in range(0,len(am_df)):
    if (am_df['DayofWeek'][i] == 0 or am_df['DayofWeek'][i] == 4):
        am_df['mon_fri'][i] = 1
    else:
        am_df['mon_fri'][i] = 0
am_df1 = am_df.sort_index(ascending=True, axis=0)
am_df1
```

Out[20]:

Date	Open	High	Low	Close	Adj Close	Volume	Daily Return
2015-04-23	390.209991	391.880005	386.149994	389.989990	389.989990	7980000	NaN
2015-04-24	439.000000	452.649994	439.000000	445.100006	445.100006	17176900	0.14131
2015-04-27	443.859985	446.989990	437.410004	438.559998	438.559998	5430900	-0.01469
2015-04-28	438.510010	439.000000	428.040009	429.309998	429.309998	4140500	-0.02109
2015-04-29	426.750000	434.239990	426.029999	429.369995	429.369995	3621700	0.00014
...	...	...	...	...	...	...	..
2020-04-16	2346.000000	2461.000000	2335.000000	2408.189941	2408.189941	12038200	0.04355
2020-04-17	2372.330078	2400.000000	2316.020020	2375.000000	2375.000000	7930000	-0.01378
2020-04-20	2389.949951	2444.979980	2386.050049	2393.610107	2393.610107	5770700	0.00783
2020-04-21	2416.610107	2428.310059	2279.659912	2328.120117	2328.120117	7476700	-0.02736
2020-04-22	2369.000000	2394.000000	2351.000000	2363.489990	2363.489990	4212200	0.01519

1259 rows × 9 columns

In [21]:

```
am_df1.fillna(am_df1.mean(), inplace=True)# Filling the missing value in daily return w
ith mean of values.
am_df1.head()
```

Out[21]:

Date	Open	High	Low	Close	Adj Close	Volume	Daily Return	Day
2015-04-23	390.209991	391.880005	386.149994	389.989990	389.989990	7980000	0.001613	
2015-04-24	439.000000	452.649994	439.000000	445.100006	445.100006	17176900	0.141311	
2015-04-27	443.859985	446.989990	437.410004	438.559998	438.559998	5430900	-0.014693	
2015-04-28	438.510010	439.000000	428.040009	429.309998	429.309998	4140500	-0.021092	
2015-04-29	426.750000	434.239990	426.029999	429.369995	429.369995	3621700	0.000140	

In [22]:

```
am=pd.DataFrame()
am['DayofWeek'] = am_df1['DayofWeek'].astype('category') # Converting Day of week to a
 category before visualization.
am['ClosedPrice']=am_df1['Close']
am.head()
```

Out[22]:

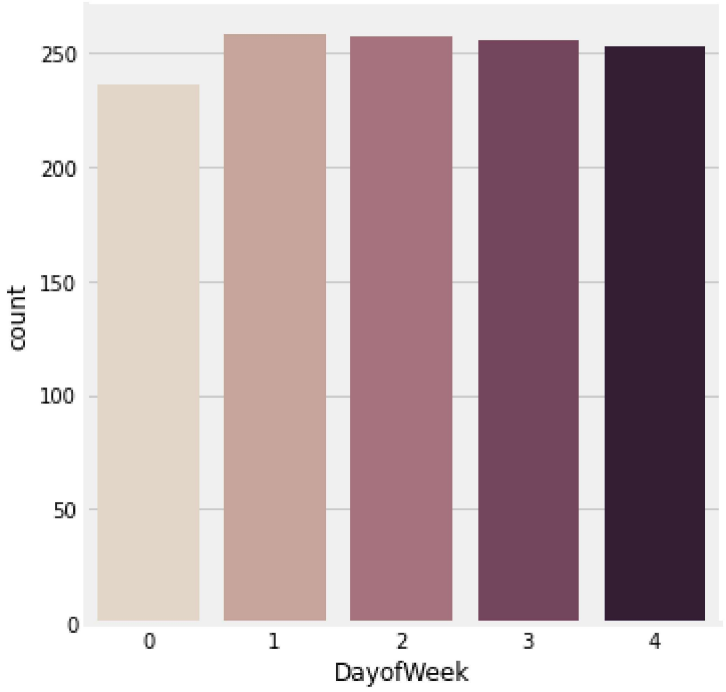
Date	DayofWeek	ClosedPrice
2015-04-23	3	389.989990
2015-04-24	4	445.100006
2015-04-27	0	438.559998
2015-04-28	1	429.309998
2015-04-29	2	429.369995

In [23]:

```
import seaborn as sns
sns.catplot(x="DayofWeek", kind="count", palette="ch:.25", data=am)
```

Out[23]:

```
<seaborn.axisgrid.FacetGrid at 0x1dfd13f4108>
```



This adds proof to Hypothesis 2 that Tuesday,wednesday and thrusday are better performers in Working days.

In [24]:

```
am_df1.isnull().sum() # checking for any null values
```

Out[24]:

```
Open           0
High           0
Low            0
Close          0
Adj Close      0
Volume         0
Daily Return   0
DayofWeek      0
mon_fri        0
dtype: int64
```

In [25]:

```
am_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 9 columns):
Open            1259 non-null float64
High            1259 non-null float64
Low             1259 non-null float64
Close           1259 non-null float64
Adj Close       1259 non-null float64
Volume          1259 non-null int64
Daily Return    1259 non-null float64
DayofWeek       1259 non-null int64
mon_fri         1259 non-null int64
dtypes: float64(6), int64(3)
memory usage: 98.4 KB
```

In [26]:

```
am_df1.shape
```

Out[26]:

```
(1259, 9)
```

Data Modelling and Testing

As mentioned in the Literary Review., the data is split into Training and Validation for the time series with 4 years of data representing the training set and the remaining for validation. Columns being considered are the closing price for prediction based on the dates. The concept of moving averages is also applied on our dataset where the prediction being an average of previous value , dropping the previous oldest value in every loop.

Algorithms used for Testing and Predictions

Moving- Average (Baseline Algorithm)

Auto-Regressive Integrated Moving Averages (ARIMA)

Long Short Term Memory (LSTM)

Prophet from Facebook

In [27]:

```
data = am_df1.drop(['Open', 'High', 'Low','Adj Close','Volume', 'DayofWeek','Daily Retu
rn','mon_fri'], axis=1)
data.head()
```

Out[27]:

Date	Close
2015-04-23	389.989990
2015-04-24	445.100006
2015-04-27	438.559998
2015-04-28	429.309998
2015-04-29	429.369995

1) Moving Average Method (Baseline Algorithm)

In [28]:

```
# splitting into train and validation
train = data[:1007]
valid = data[1007:]
pred = []
valid.shape
for i in range(0,valid.shape[0]):
    a = train['Close'][len(train)-252+i:].sum() + sum(pred)
    b = a/252
    pred.append(b)
rms=np.sqrt(np.mean(np.power((np.array(valid['Close'])-pred),2)))
print('\n RMSE value on validation set(base value):')
print(rms)
```

```
 RMSE value on validation set(base value):
195.34875658857368
```

In [29]:

```
valid['Predictions'] = 0
valid['Predictions'] = pred
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
```

Out[29]:

```
[<matplotlib.lines.Line2D at 0x1dfd18ff348>,
 <matplotlib.lines.Line2D at 0x1dfd18ff508>]
```

The RMSE base value is close to 195 . The predicted values are showing an average value trend following the same range as the training values.

Next we do the AutoARIMA algorithm.

2) Auto -ARIMA

As discussed in the Literary Review for algorithms

AR term refers to the past values used for forecasting the next value. The AR term is defined by the parameter ‘p’ in arima.

MA term is used to defines number of past forecast errors used to predict the future values. The parameter ‘q’ in arima represents the MA term.

In [30]:

```
from pmdarima.arima import auto_arima
train = data[:1007]
valid = data[1007:]

training = train['Close']
validation = valid['Close']

model = auto_arima(training, start_p=1, start_q=1,max_p=3, max_q=3, m=12,start_P=0, sea
sonal=True,d=1, D=1, trace=True,error_action='ignore',suppress_warnings=True)
model.fit(training)

forecast = model.predict(n_periods=252)
forecast = pd.DataFrame(forecast,index = valid.index,columns=['Prediction'])
```

```
Performing stepwise search to minimize aic
Fit ARIMA: (1, 1, 1)x(0, 1, 1, 12) (constant=True); AIC=9056.145, BIC=908
0.654, Time=3.786 seconds
Fit ARIMA: (0, 1, 0)x(0, 1, 0, 12) (constant=True); AIC=9678.340, BIC=968
8.143, Time=0.086 seconds
Fit ARIMA: (1, 1, 0)x(1, 1, 0, 12) (constant=True); AIC=9440.770, BIC=946
0.377, Time=1.315 seconds
Fit ARIMA: (0, 1, 1)x(0, 1, 1, 12) (constant=True); AIC=9056.895, BIC=907
6.502, Time=1.504 seconds
Near non-invertible roots for order (0, 1, 1)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (0, 1, 0)x(0, 1, 0, 12) (constant=False); AIC=9676.344, BIC=968
1.246, Time=0.034 seconds
Fit ARIMA: (1, 1, 1)x(0, 1, 0, 12) (constant=True); AIC=9680.811, BIC=970
0.418, Time=1.111 seconds
Fit ARIMA: (1, 1, 1)x(1, 1, 1, 12) (constant=True); AIC=9057.366, BIC=908
6.776, Time=12.248 seconds
Near non-invertible roots for order (1, 1, 1)(1, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (1, 1, 1)x(0, 1, 2, 12) (constant=True); AIC=9057.311, BIC=908
6.722, Time=15.010 seconds
Near non-invertible roots for order (1, 1, 1)(0, 1, 2, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (1, 1, 1)x(1, 1, 0, 12) (constant=True); AIC=9412.191, BIC=943
6.700, Time=7.245 seconds
Near non-invertible roots for order (1, 1, 1)(1, 1, 0, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (1, 1, 1)x(1, 1, 2, 12) (constant=True); AIC=9060.142, BIC=909
4.454, Time=15.946 seconds
Near non-invertible roots for order (1, 1, 1)(1, 1, 2, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (1, 1, 0)x(0, 1, 1, 12) (constant=True); AIC=9056.912, BIC=907
6.519, Time=3.533 seconds
Near non-invertible roots for order (1, 1, 0)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (2, 1, 1)x(0, 1, 1, 12) (constant=True); AIC=9057.678, BIC=908
7.088, Time=6.499 seconds
Near non-invertible roots for order (2, 1, 1)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (1, 1, 2)x(0, 1, 1, 12) (constant=True); AIC=9057.700, BIC=908
7.111, Time=5.527 seconds
Near non-invertible roots for order (1, 1, 2)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (0, 1, 0)x(0, 1, 1, 12) (constant=True); AIC=9055.526, BIC=907
0.232, Time=1.988 seconds
Near non-invertible roots for order (0, 1, 0)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (0, 1, 2)x(0, 1, 1, 12) (constant=True); AIC=9057.227, BIC=908
1.736, Time=2.400 seconds
Near non-invertible roots for order (0, 1, 2)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
```

```
Fit ARIMA: (2, 1, 0)x(0, 1, 1, 12) (constant=True); AIC=9057.377, BIC=908
1.886, Time=2.079 seconds
Near non-invertible roots for order (2, 1, 0)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Fit ARIMA: (2, 1, 2)x(0, 1, 1, 12) (constant=True); AIC=9053.636, BIC=908
7.948, Time=10.151 seconds
Near non-invertible roots for order (2, 1, 2)(0, 1, 1, 12); setting score
to inf (at least one inverse root too close to the border of the unit circ
le: 1.000)
Total fit time: 90.473 seconds
```

In [31]:

```
rms_ARIMA=np.sqrt(np.mean(np.power((np.array(valid['Close'])-np.array(forecast['Predict
ion'])),2)))
rms_ARIMA
```

Out[31]:

```
351.3994908291147
```

High value of RMSE value than the base line. Also on plotting the below graph you can notice an increased trend in prediction based on the previous moving averages. It neither take any trend or seasonality into account

In [32]:

```
plt.plot(train['Close'])
plt.plot(valid['Close'])
plt.plot(forecast['Prediction'])
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0x1dfa1b62208>]
```

3) LSTM Algorithm

Next we implement the LSTM model , a deep learning algorithm which is the best when it comes to predicting sequential problems. It acts as an improvised version of RNN where it stores only the information that is needed based on which predictions are done. The forget gate filters the information thats not needed anymore, apart from having an input gate and output gate.

In [33]:

```
#LSTM

from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM

new_data = data.copy()

#creating train and test sets
dataset = new_data.values

train = dataset[0:1007,:]
valid = dataset[1007:,:]

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

# 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=1, batch_size=1, verbose=2)

#predicting 246 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)

rms_LSTM=np.sqrt(np.mean(np.power((valid-closing_price),2)))
rms_LSTM
```

```
Using TensorFlow backend.

Epoch 1/1
 - 28s - loss: 0.0021
```

Out[33]:

```
164.85108192567327
```

This is the best RMSE value (164.85) so far as compared to the base value(195) and AutoARIMA (385) also as plotted in the below graph.It can be tuned for the number of LSTM layers, increasing the epochs numbers etc.

However this might also be treated as too perfect again.The predicted values seem to perfectly map the validation curve. LSTM does not involve seasonalities or any other unexpected trend cycle in them which is required for our objective to predict long term trends.

In [34]:

```
train = new_data[:1007]
valid = new_data[1007:]
valid['Predictions'] = closing_price
plt.plot(train['Close'])
plt.plot(valid[['Close','Predictions']])
```

Out[34]:

```
[<matplotlib.lines.Line2D at 0x1df893bc688>,
 <matplotlib.lines.Line2D at 0x1df893bc808>]
```

4) Prophet from Facebook

Prophet is a statistical approach to forecasting developed by Facebook that takes seasonality, trends and holidays into accounts while doing time forecasting.

In [35]:

```
from fbprophet import Prophet
```

In [36]:

```
data['Date']=data.index
data.head()
```

Out[36]:

	Close	Date
Date		
2015-04-23	389.989990	2015-04-23
2015-04-24	445.100006	2015-04-24
2015-04-27	438.559998	2015-04-27
2015-04-28	429.309998	2015-04-28
2015-04-29	429.369995	2015-04-29

In [37]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1259 entries, 2015-04-23 to 2020-04-22
Data columns (total 2 columns):
Close    1259 non-null float64
Date     1259 non-null datetime64[ns]
dtypes: datetime64[ns](1), float64(1)
memory usage: 29.5 KB
```

In [38]:

```
#fit the model

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

train = data[:1007]
valid = data[1007:]

model = Prophet()
model.fit(train)

#predictions
close_prices = model.make_future_dataframe(periods=len(valid))
forecast = model.predict(close_prices)
#rmse
forecast_valid = forecast['yhat'][1007:]
rms_FB=np.sqrt(np.mean(np.power((np.array(valid['y'])-np.array(forecast_valid)),2)))
rms_FB
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasona
lity=True to override this.
```

Out[38]:

```
198.53080585507763
```

The RMSE value of Prophet (198.5) matches with that of the base value.

In [39]:

```
valid['Predictions'] = 0
valid['Predictions'] = forecast_valid.values

plt.plot(train['y'])
plt.plot(valid[['y', 'Predictions']])
```

Out[39]:

```
[<matplotlib.lines.Line2D at 0x1df8c00de88>,
 <matplotlib.lines.Line2D at 0x1df8c00d748>]
```

The prediction curve for prophet highlights that the value initially tends to follow the moving average and slightly drops down in 2020. This is not the perfect solution in terms of a prediction, however the objective of the coursework is to find a long term pattern in 2020-21 period for the closing price predictions.

Algorithm Evaluation and Inference

We have already discussed the Individual models and its inferences in their respective segments. This section combines all the details and discusses on which model is chosen finally and why based on the information.

Moving Average (Base Algorithm) : RMSE Value 195 - Predictions are graphed to be an average representation of the Training set.

Auto -ARIMA : RMSE value 385 - Predictions are graphed as incrementing curve compared to thetraining set - Does not take any trends/ seasonality or unpredicted changes in stock prices into account - Not the ideal model to be chosen

LSTM : RMSE value of 164.85 - Predictions are in line with the validation curve perfectly - Marking the best value among all the algorithms - Does not take trends/ seasonality or unpredicted changes in stock prices into account - Preferred choice of algorithm for short term predictions.

Prpohet : RMSE Value 198 - Predictions are graphed more realistically to overall trends - Preferred choice of algorithm for long term predicitions - Can graph predictions on a weekly , monthly and yearly basis till 2021.

Objective of the course work is to do long term predictions on the closing price taking trends or unpredictive nature of stocks into account. Based on this information, for the particular coursework Prophet might make more sense to implement.

We can go ahead with Prophet despite that giving a higher score than LSTM model which gave the best RMSE scores , as prophet can give you a lot insight into data in terms of forecasting based on seasons , trends and other factors which is required on the long term predictions as per our objective.

Forecast Results in detail

Creating a forecasting dataframe

In the next section we are going to make use of the make_future_dataframe with a period of 365 days grasping based on which we will forecast our future prices. The dataframe has a forecasted value in terms of yhat with a lower and upper limit of uncertainities.

In [40]:

```
# Create Future dates
future_prices = model.make_future_dataframe(periods=365)

# Predict Prices
forecast = model.predict(future_prices)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

Out[40]:

	ds	yhat	yhat_lower	yhat_upper
1367	2020-04-18	1756.960607	1415.412845	2097.090375
1368	2020-04-19	1758.939317	1406.715065	2106.180440
1369	2020-04-20	1737.839931	1400.601086	2075.804737
1370	2020-04-21	1745.129301	1394.346196	2084.027767
1371	2020-04-22	1746.968299	1407.291222	2089.684814

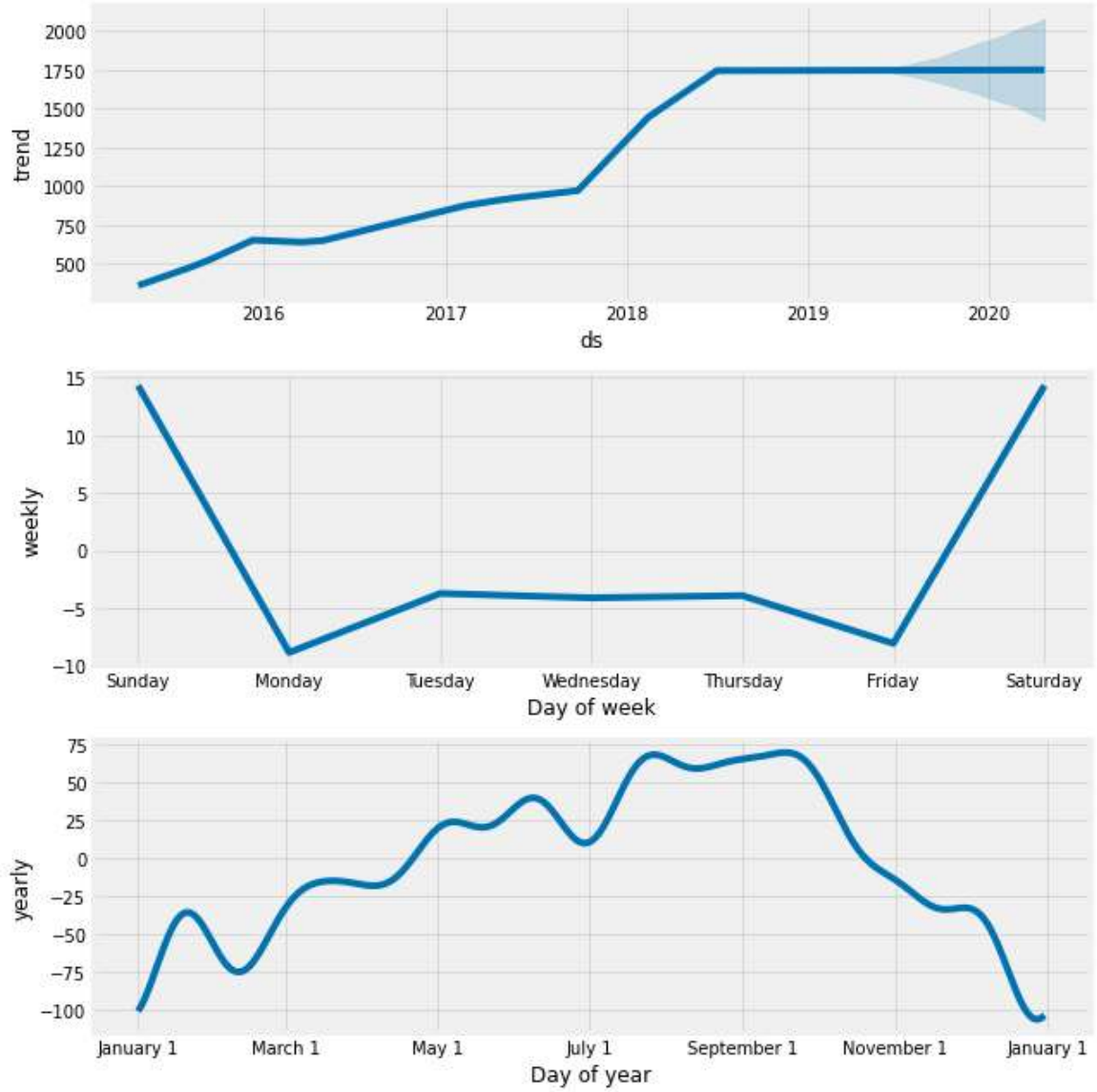
In [47]:

```
import matplotlib.dates as mdates
import datetime as dt
# Dates,
starting_date = dt.datetime(2019, 4, 7)
starting_date1 = mdates.date2num(starting_date)
trend_date = dt.datetime(2019, 6, 7)
trend_date1 = mdates.date2num(trend_date)

pointing_arrow = dt.datetime(2019, 4, 23)
pointing_arrow1 = mdates.date2num(pointing_arrow)

# Learn more Prophet tomorrow and plot the forecast for amazon.
fig = model.plot(forecast)
ax1 = fig.add_subplot(111)
ax1.set_title("Amazon Stock Price Forecast", fontsize=16)
ax1.set_xlabel("Date", fontsize=12)
ax1.set_ylabel("Close Price", fontsize=12)

# Forecast initialization arrow
ax1.annotate('Forecast \n Initialization', xy=(pointing_arrow1, 1350), xytext=(starting
_date1,1700),
            arrowprops=dict(facecolor='#ff7f50', shrink=0.1),
            )

# Trend emphasis arrow
ax1.annotate('Upward Trend', xy=(trend_date1, 1225), xytext=(trend_date1,950),
            arrowprops=dict(facecolor='#6cff6c', shrink=0.1),
            )

ax1.axhline(y=1260, color='b', linestyle='-')

plt.show()
```

Initializing your forecasting trend from the date 4/23/19 , picking up information from trends set at 6/7/2019 we are plotting our forecast. You can see the various components of the future dataframe we created below on basis on days , day of week , day of year.

In [48]:

```
model.plot_components(forecast)
plt.show()
```

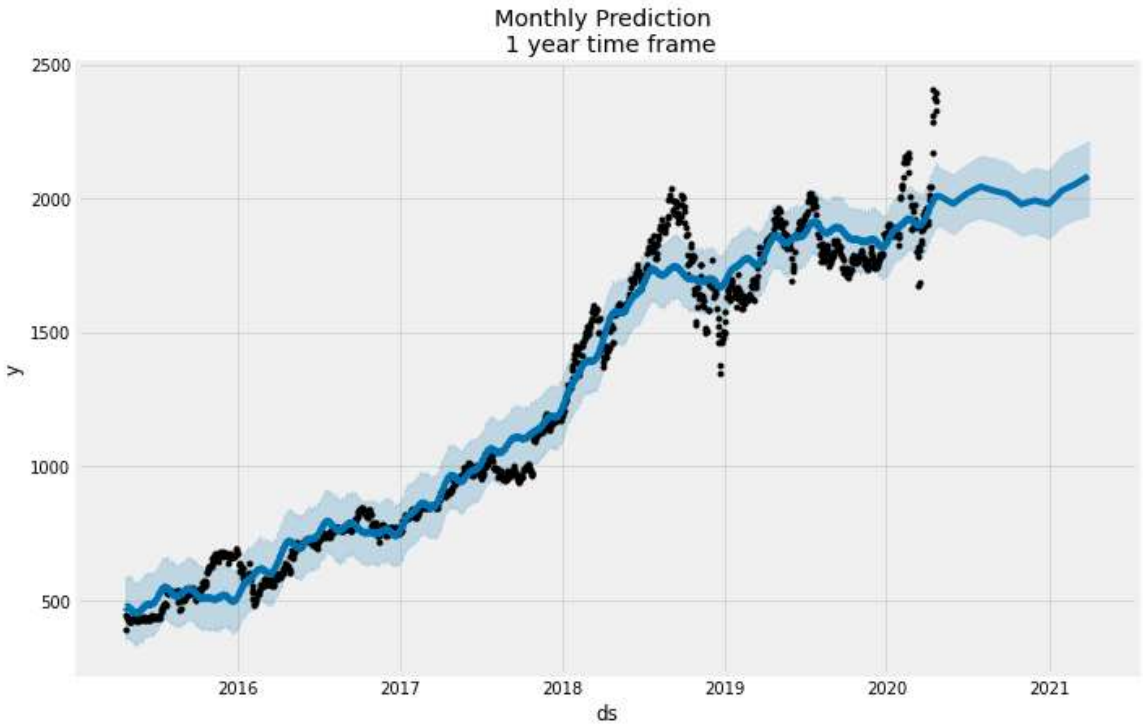


The next step involves fitting our historical data into prophet and creating a dataframe using the same method as above. Objective is to do a monthly prediction on a 1 year time frame that is 2020-21. Lets plot our predictions based on it.

In [43]:

```
# Monthly Data Predictions
m = Prophet(changepoint_prior_scale=0.01).fit(data)
future = m.make_future_dataframe(periods=12, freq='M')
fcst = m.predict(future)
fig = m.plot(fcst)
plt.title("Monthly Prediction \n 1 year time frame")

plt.show()
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasona
lity=True to override this.
```



Based on the predictions above and the below plot components from prophet ,We can come to the following conclusions from our Hypothesis 2,4 and 5

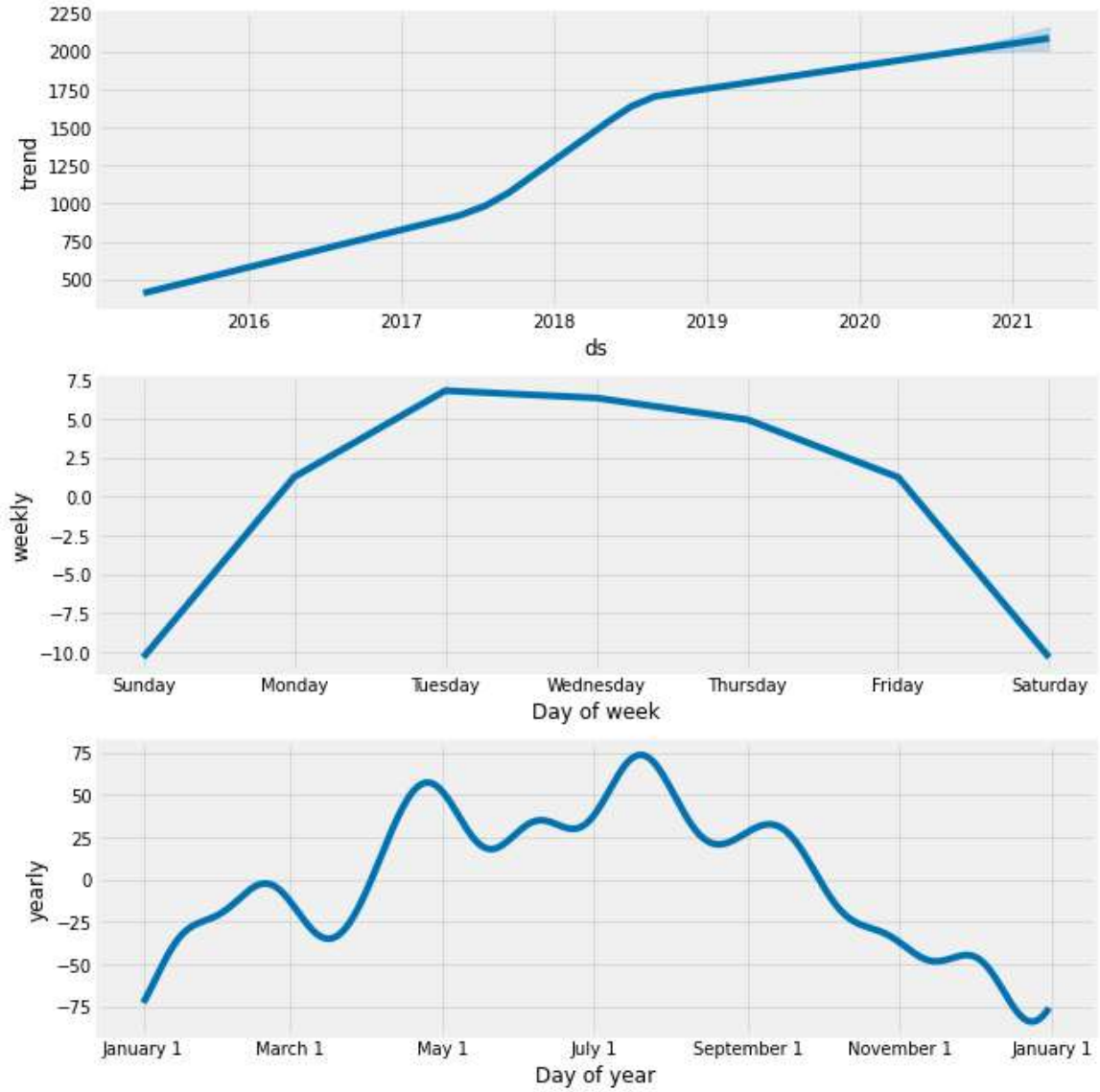
Overall stock performance is lower on the first and last working day (Monday and Friday).

Stock performance post Corona pandemic is still good for Amazon. Please note the gradual predicted stock price increase from 2020 to 2021 which might gradually pick up post 21.

Stock performance of second and third quarter of a year should outweigh the first and last quarter. Stock prices seem to show higher patterns from April to September with July/August peaking in terms of predictions.

In [44]:

```
m.plot_components(fcst)
plt.show()
```



Conclusion

Despite LSTM giving the best RMSE compared to the baseline, we went with Prophet due to its ability to take seasons and trends into consideration. Stock price predictions are highly volatile due to its very nature of considering sentiments of the investors and news related to market conditions.

Amazon was the best performer (Hypothesis 1), whose closing price was the basis of analysis for the remaining Hypothesis.

As shown in the future predictions by prophet as well as the historical data visualization, Tuesday, wednesday and thursday peaks in terms of stock prices. (Hypothesis 2)

Amazon stocks have increased by huge numbers since President Trump took to office (Hypothesis 3). This is contrary to his open clashes with Amazon related to Tax issues, JEDI contract and multiple other factors. They are still an investor favorite considering that they lead retail businesses and stepped up cloud computing with AWS.

Resource: https://www.marketwatch.com/press-release/amazon-stock-has-risen-170-since-trump-took-office-2020-02-18 (https://www.marketwatch.com/press-release/amazon-stock-has-risen-170-since-trump-took-office-2020-02-18)

Their streaming services and cloud platforms are over the top, with its ecommerce sectors as well during the pandemic period. This is proved as well in the future prediction by prophet (Hypothesis 4) with March to May showing a good rise in stock prices till 2021.

Stock performance of second and third quarter of a year should outweigh the first and last quarter. Stock prices seem to show higher patterns from April to September with July/August peaking in terms of predictions. (Hypothesis 5)

What most of these statements do infer is that a lot of stock trading is dependent on investor sentiment. The belief that cloud solutions or e-commerce would definitely prevail in such market conditions, and with Amazon digging deeper into technological solutions gives an overall positive consumer sentiment, despite threats such as data privacy or other news factors doing the runs.

To concur the same the numbers do show positive trend in sentiments as well in our algorithms.

Resource: https://www.investors.com/news/technology/amazon-stock-buy-now/ (https://www.investors.com/news/technology/amazon-stock-buy-now/)

Resource: https://www.fool.com/investing/2020/04/23/covid-19-crisis-cant-sink-3-e-commerce-stocks.aspx (https://www.fool.com/investing/2020/04/23/covid-19-crisis-cant-sink-3-e-commerce-stocks.aspx)

Amazon is followed by Microsoft , largely owing to its cloud, teams and skype usage surging especially in the current period. Large scale enterprises do use Azure for its datawarehousing and cloud computing purposes. The foundation is also active in its fight against the pandemic, which sets its trend positive in the investors mind.

Resource: https://www.fool.com/investing/2020/04/03/microsoft-poised-to-emerge-from-the-covid-19-pande.aspx (https://www.fool.com/investing/2020/04/03/microsoft-poised-to-emerge-from-the-covid-19-pande.aspx)

As a conclusive statement, it's good to know that no machine learning algorithm is the ultimate decision maker when it comes to time series forecasting especially in stock predictions. There are lots of factors involving

investor sentiments that needs to be covered including time being an unknown and unpredictable components that could change overnight. This is an interesting field of study that still needs to be developed.