Stock Market Analysis

by Norzarifah Kamarauzaman

Submission date: 22-Jun-2020 12:46PM (UTC+0800)
Submission ID: 1345328406
File name: WQD7003_Stock_Market_Analysis.pdf (1.23M)
Word count: 6454
Character count: 34207

Stock Market Analysis

- Norzarifah Kamarauzaman 17041741
- Nazrah Mustapa 17218099
- Fatin Nabilah Abdul Rahman 17201487
- Mohamad Azery Hussin 17200989

Table of Contents

Overview and Motivation:

Investing the "extra cash" once a person has a steady source of income and solid emergency fund is a truly mind-boggling matter. Should the money be kept in savings account or should it be invested in real estate, bonds, mutual funds or stocks? Inarguably, the matter of deciding the best place to stash the cash is not always easy and as quoted from *Warren Buffet*, who is regarded as the most successful investor in the world, ***Do not put all eggs in one basket***.

According to a report published by *Franklin Templeton Investments* (2020), stocks have averaged a 10.59% one-year return over the last half century (ended December 31, 2019), therefore this "basket" provide the highest potential returns in comparison to other kinds of investments. However, stocks tend to be the most-volatile investments thus carrying a higher degree of risk in comparison to other investment opportunities.

This project is about **forecasting time series data using machine learning algorithms** as the stock market data itself e.g. the daily opening and closing stock prices are time-series in nature. In the real world however, do note that stock prices are also affected by **financial news** (such as demonetization or company's activity like merger/demerger etc. which can be deciphered by applying Natural Language Processing (NLP) but this will not be convered in the project) and other intangible factors that can often be impossible to predict beforehand e.g. COVID-19 Pandemic.

Initial Questions:

So, stocks investment seems promising for wealth accumulation, but if a person wants to start investing today, these are definitely some questions that would come the mind, such as what is the best stock to invest in right now?, should I buy now or wait?...and so on.

To align with the abovementioned questions, our goal is **to predict the closing price of a selected stock at a given date, based on the past 60-days trading data**. The questions that we are trying to address from this project are (but not limited to):

- **Question 1:** Given a selection of companies, which stock should I buy according to my financial budget?
- **Question 2:** How volatile is the selected stock?
- **Question 3:** What is the expected closing price of the selected stock at a given date?

Related Work:

The idea to predict future stock price using machine learning is not novel as there are a lot of studies have been conducted on this topic especially in the recent decade, attributable to rapid technological advancement in computing.

A study conducted by Hegazy et al. (2013) proposed a machine learning model that *integrates Particle swarm optimization (PSO) and least square support vector machine (LS-SVM) algorithms i.e. LS-SVM-PSO* to predict the stock prices of several companies selected from a variety of sectors in S&P 500 stock market. The data were retrieved from Yahoo Finance within a period of January 2009 to January 2012. In summary, this study used financial technical indicators such as relative strength index (RSI), money flow index (MFI), exponential moving average (EMA), stochastic oscillator (SO), and moving average convergence/divergence (MACD) calculated from the historical raw datasets (i.e. feature extraction and selection). These features were then made as an input to model that leveraged on LS-SVM-PSO, LS-SVM, and ANN algorithms. Mean Square Error (MSE) performance function was used to evaluate the models and the overall results conveyed that LS-SVM optimized with PSO is the best one with lowest error value followed by LS-SVM algorithm.

The more recent publication related to stock market prediction using machine learning is *the application of deep learning methods to predict the intraday directional movements of S&P 500 index* Vargas et al. (2017). In their study, Vargas et al. (2017) leverage Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to perform Natural Language Processing (NLP) on financial news titles and a set of technical indicators to detect and analyze complex patterns and interactions present in the data to subsequentially accelerate the trading process. The results revealed that RNN surpassed CNN on catching the context information and modeling complex temporal characteristics for stock market forecasting although CNN performed better than RNN on catching semantic from texts.

Understand the Dataset and Get Data:

The dataset used in this project is retrieved from Kaggle Stock Market Dataset (https://www.kaggle.com/jacksoncrow/stock-market-dataset) which contains historical daily prices of all stocks and exchange-traded funds (ETFs) that are currently being traded on NASDAQ.

An article published in The Motley Fool (https://www.fool.com/investing/2020/04/24/surprise-most-faang-stocks-are-historically-cheap.aspx) has recommended the `FAANG stocks`. These stocks represent the five most popular and best-performing global technology companies which have led the market over the past decade:

- Facebook (FB)
- Amazon (AMZN)
- Apple (AAPL)
- Netflix (NFLX)
- Alphabet (GOOG & GOOGL)

The variables present in the dataset are:

Variables	Description
Date	The date at which the stock is traded
Open	The staring price at which the stock is traded on a particular day
High	The maximum price of the share for the day
Low	The minimum price of the share for the day
Close	The final price at which the stock is traded on a particular day
Adj_Close	The final price that include dividends, stock splits and new stock offerings
Volume	The number of shares bought or sold for the day

In [1]:

```
# import libraries to ignore warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings('ignore')

# import general libraries
import numpy as np
import pandas as pd
import math
import os

# import libaries for visualisation and define setting
import matplotlib.pyplot as plt
from matplotlib import gridspec
from pandas.plotting import lag_plot
plt.style.use("seaborn-white")
plt.rcParams["figure.figsize"]=16,10
plt.rcParams.update({'font.size': 14})

# import libraries for time series analysis
from datetime import datetime, date, timedelta
from sorted_months_weekdays import *
from sort_dataframeby_monthorweek import *
from pandas.tseries.holiday import USFederalHolidayCalendar
from pandas.tseries.offsets import CustomBusinessDay
from pandas.tseries.offsets import *
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose

# import libraries for modeling
from pmdarima import auto_arima
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

```
Using TensorFlow backend.
```

In [2]:
```
# define function to read files:

os.chdir('.\stocks')

def read_file(abbr):
    # read multiple .csv files in the directory
    file_list = []
    for file in os.listdir():
        if file.endswith('.csv'):
            df = pd.read_csv(file,)
            df['Symbol'] = file.replace(".csv", "")
            file_list.append(df)
    data = pd.concat(file_list, axis=0, ignore_index=True)
    if abbr == " ":
        data = data
    else:
        data = data[data['Symbol'].isin(abbr)]
    # rename columns
    data.columns = ['Date', 'Open', 'High', 'Low', 'Close', 'Adj_Close', 'Volu
me', 'Symbol']
    return(data)
```

In [3]:
```
# read data
stock_data = read_file(['AAPL', 'AMZN', 'FB', 'NFLX', 'GOOGL'])
```

In [4]:
```
# to check: display the first five rows of data
stock_data.head()
```

Out[4]:

	Date	Open	High	Low	Close	Adj_Close	Volume	Symbol
0	1980-12-12	0.513393	0.515625	0.513393	0.513393	0.406782	117258400	AAPL
1	1980-12-15	0.488839	0.488839	0.486607	0.486607	0.385558	43971200	AAPL
2	1980-12-16	0.453125	0.453125	0.450893	0.450893	0.357260	26432000	AAPL
3	1980-12-17	0.462054	0.464286	0.462054	0.462054	0.366103	21610400	AAPL
4	1980-12-18	0.475446	0.477679	0.475446	0.475446	0.376715	18362400	AAPL

In [5]:
```
# to check: display the last five rows of data
stock_data.tail()
```

Out[5]:

	Date	Open	High	Low	Close	Adj_Close	Volume	Symbol
25970	2020-02-27	371.459991	391.559998	370.600006	371.709991	371.709991	10967700	NFLX
25971	2020-02-28	364.209991	376.769989	356.799988	369.029999	369.029999	11178600	NFLX
25972	2020-03-02	373.109985	381.359985	364.500000	381.049988	381.049988	6997900	NFLX
25973	2020-03-03	381.029999	393.519989	367.399994	368.769989	368.769989	8364600	NFLX
25974	2020-03-04	377.769989	384.010010	370.510010	383.790009	383.790009	5481300	NFLX

Exploratory Data Analysis

In Time Series Analysis, the data must be indexed by timestamps, thus we will begin by creating a datetime index for the dataframe.

In [6]:
```
# set date as index
stock_data['Date'] = pd.to_datetime(stock_data['Date'], format='%Y-%m-%d')
stock_data.set_index('Date', inplace=True)
```

In [7]:
```
# to check: display a random sampling of 5 rows
stock_data.sample(5, random_state=0)
```

Out[7]:

Date	Open	High	Low	Close	Adj_Close	Volume	Symbol
2016-03-29	753.679993	767.179993	748.289978	765.890015	765.890015	2003100	GOOGL
2007-03-06	223.958954	229.729736	223.913910	229.003998	229.003998	15052300	GOOGL
2013-04-22	400.700714	402.382385	387.887878	400.455444	400.455444	5761000	GOOGL
2014-10-22	541.049988	550.760010	540.229980	542.690002	542.690002	2973700	GOOGL
2005-08-23	6.550000	6.585714	6.474286	6.534286	5.672027	73901100	AAPL

The closing price `Close` and the adjusted closing price `Adj_Close` refer to the different ways of stocks being valuated. The later represents a more complex analysis that using the closing price as a starting point.

> - The **closing price** is literally the price of a particular stock at the close of a trading day.
> - The **adjusted closing price** taking takes into account factors such as dividends, stock splits and new stock offerings.

Seeing that `Adj_Close` represents a more accurate reflection of a stock's value, we will make this variable as the target variable.

In [8]:
```
# subset Adj_Close from stock_data:
df_adj_close = stock_data[['Adj_Close', 'Symbol']]
```

In [9]:
```
# to check: display a random sampling of 5 rows
df_adj_close.sample(5, random_state=0)
```

Out[9]:

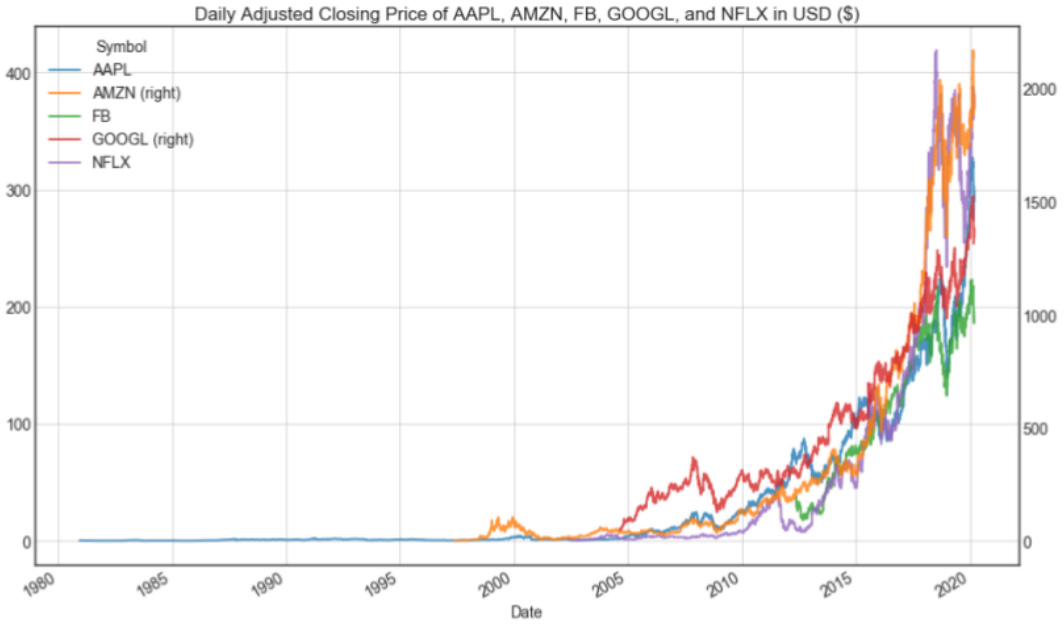
Date	Adj_Close	Symbol
2016-03-29	765.890015	GOOGL
2007-03-06	229.003998	GOOGL
2013-04-22	400.455444	GOOGL
2014-10-22	542.690002	GOOGL
2005-08-23	5.672027	AAPL

Line Plot: the first step to comprehend the data

The best way to understand the data is through visualisation. Let's plot the daily adjusted closing price of the selected companies using line charts and then compare each performances throughout the years.

In [10]:
```
# reshape the data frame for visualisation:
df_adj_close_viz = df_adj_close.pivot(columns='Symbol')
df_adj_close_viz.columns = df_adj_close_viz.columns.droplevel(0)
df_adj_close_viz.plot(secondary_y=['AMZN', 'GOOGL'], grid=True, linewidth=2, a
lpha=0.8)
plt.title('Daily Adjusted Closing Price of AAPL, AMZN, FB, GOOGL, and NFLX in
 USD ($)')
```

Out[10]:
```
Text(0.5, 1.0, 'Daily Adjusted Closing Price of AAPL, AMZN, FB, GOOGL, and NF
LX in USD ($)')
```



In [11]:
```
# to check: display a random sampling of 5 rows
# df_adj_close_viz.sample(5, random_state=0)
```

From data point of view, it is evident that these stocks share a common `end_date` i.e. 2020-03-04 whereas `start_date` differs from each other.

- **Question 1:** Given a selection of companies, which stock should I buy according to my financial budget?

To address the first question raised in this study, it seems like AMZN and GOOGL seem out of the league as the price per share for each stocks exceeds USD2500 and USD1400 respectively as of June 2020.

Missing Data Analysis

Although the data is claimed to have 'daily' frequency, we need to check if this is true or not. Is there any missing trading days in the data?

In [12]:
```
# is there any missing datapoints in the data?
df_adj_close.isna().sum()
```

Out[12]:
```
Adj_Close    0
Symbol       0
dtype: int64
```

In [13]:
```
# define function to identify the missing trading day(s) for each company:

def missing_dates(data):

    grouped = data.groupby('Symbol')
    output = []
    for key, value in grouped:

        data_dates = value.index.sort_values()
        trading_days = len(data_dates)

        start_date = data_dates[0]
        end_date = data_dates[-1]

        business_dates = pd.date_range(start_date, end_date, freq=BDay())
        total_bdays = len(business_dates)

        dates_missing = pd.to_datetime([item for item in business_dates if ite
m not in data_dates], format='%Y-%m-%d')
        missing_days = len(dates_missing)
        missing_perc = round(missing_days/total_bdays*100.0, 2)

        data_dict = {'Symbol':key,'Start_date':start_date, 'End_date':end_date
, 'Business_days':total_bdays,
                     'Trading_days':trading_days, 'Missing_days':missing_days,
'Missing_percentage':missing_perc,
                     'Missing_dates': dates_missing}
        output.append(data_dict)
        df = pd.DataFrame(output)
        df.sort_values('Start_date').reset_index(drop=True)

    return df
```

In [14]:
```
missing_dates(df_adj_close)
```

Out[14]:

	Symbol	Start_date	End_date	Business_days	Trading_days	Missing_days	Missing_percenta
0	AAPL	1980-12-12	2020-03-04	10234	9889	345	3.
1	AMZN	1997-05-15	2020-03-04	5950	5738	212	3.
2	FB	2012-05-18	2020-03-04	2034	1960	74	3.
3	GOOGL	2004-08-19	2020-03-04	4055	3912	143	3.
4	NFLX	2002-05-23	2020-03-04	4640	4476	164	3.

There is a very little percentage of missing datapoints in the data (on the basis of trading days are conducted during business days). Further inspection reveals that these dates are attributed to the public holidays in the United States, thus no stock trading are held on these days.

Date	Day	Event
1981-01-01	Thursday	New Year's Day
1997-05-26	Monday	Memorial Day
1997-07-04	Friday	Independence Day
2000-07-04	Tuesday	Independence Day
2012-05-28	Monday	Memorial Day
2012-07-04	Wednesday	Independence Day

Lag Plot: dechiper more information from a time series

Time series modeling assumes that **there is a relationship between an observation and the previous observation**. A `lag` is a fixed time displacement and a `lag plot` is useful in exploring the relationship between a selected observation and a lag of that observation. Data randomness, model suitability, presence of outliers, serial correlation, and seasonality are some of the properties of a time series that can be deciphered from the lag plot.

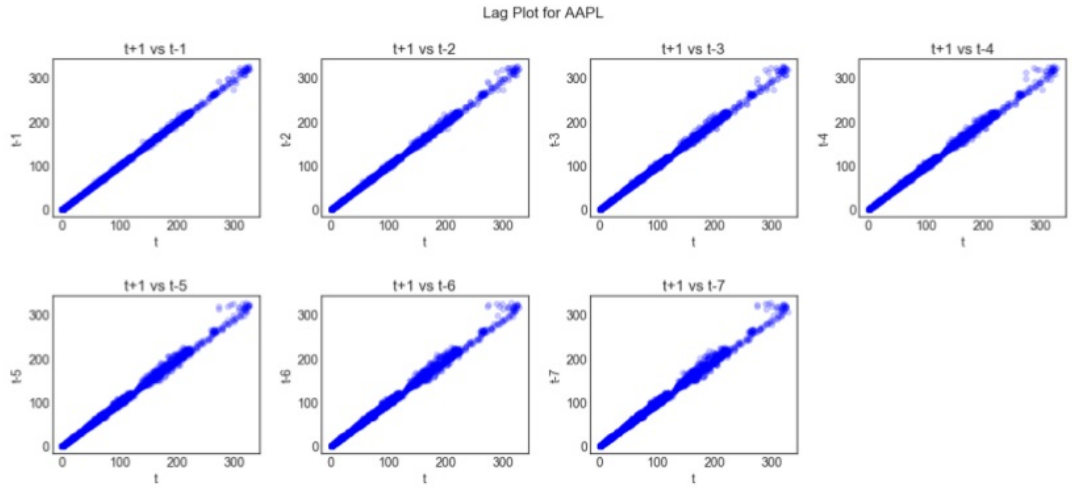
Given a data values of Y_1, Y_2, Y_3 ..., Y_n, the k-period (or k^{th}) lag of the value Y_i is defined as the data point that occurred k time points before time i:

$$Lag_k(Y_i) = Y_{i-k}$$

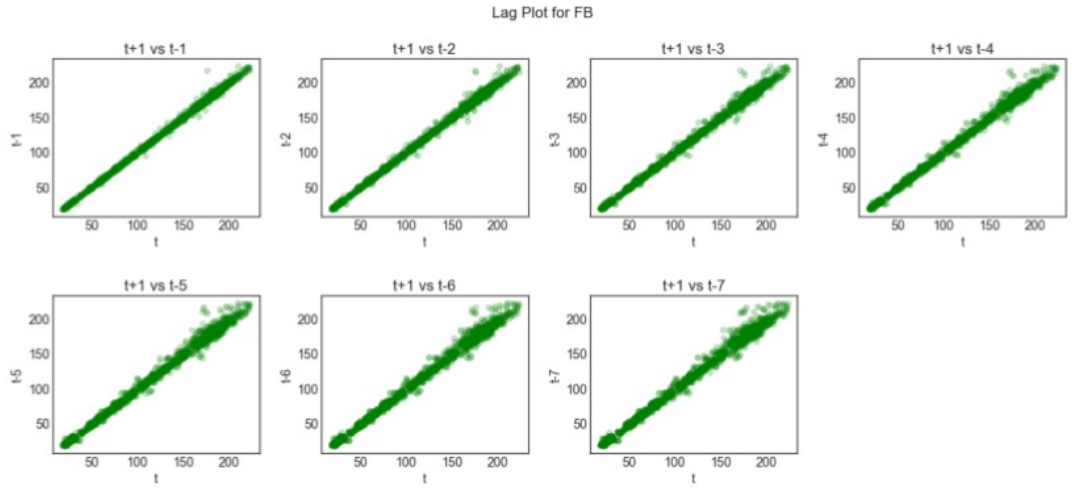
In [15]:

```
# define funtion to plot weekly lagging:

def lag_plot_weekly(data, company):

    grouped = data.groupby('Symbol')

    for key, value in grouped:
        if key == company:
            data = value[value['Symbol']== key]['Adj_Close']
            lags = 7
            columns = [data]
            for i in range(1,(lags + 1)):
                columns.append(data.shift(i))
            dataframe = pd.concat(columns, axis=1)
            columns = ['t+1']
            for i in range(1,(lags + 1)):
                columns.append('t-' + str(i))
            dataframe.columns = columns

            if key == 'AAPL':
                color = 'blue'
            elif key == 'FB':
                color = 'green'
            else:
                color = 'purple'

            plt.figure(figsize=(20,8))
            for i in range(1,(lags + 1)):
                ax = plt.subplot(240 + i)
                plt.subplots_adjust(hspace=0.5, wspace=0.3)
                ax.set_title('t+1 vs t-' + str(i))
                plt.scatter(x=dataframe['t+1'].values, y=dataframe['t-'+str(i
)].values, alpha=0.2, color=color)
                ax.set_ylabel('t-' +str(i))
                ax.set_xlabel('t')
            plt.suptitle('Lag Plot for {}'.format(key), fontdict = {'fontsize'
: 20})

        plt.show()
```

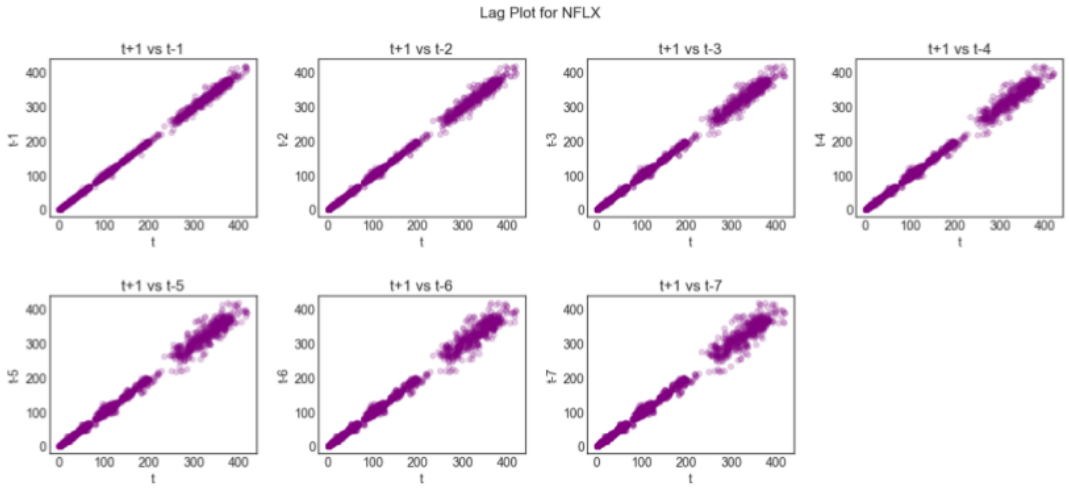
In [16]:
```
lag_plot_weekly(stock_data, 'AAPL')
```



In [17]:
```
lag_plot_weekly(stock_data, 'FB')
```



```
In [18]: lag_plot_weekly(stock_data, 'NFLX')
```



Explanation:

The shape of the lag plot provides clue about the underlying structure of the data; and it this case, the lag plots for the selected companies (AAPL, FB, and NFLX) exhibit **strong positive linear pattern** thus implying that the time series are strongly **non-random** and an **autoregressive model** is probably a better choice. Presence of **outliers** in the data is evident especially in the higher value ranges. Data with seasonality will repeat itself periodically in a sine or cosine-like wave but in our case, no such pattern can be observed, thus none of the data exhibits seasonality.

Although **postitive autocorrelation** is present, the strength of the autocorrelation weakens day by day seeing that the first lag has datapoints that are tightly clustered around the diagonal, but the datapoints begin to disperse away from the diagonal in the next lag and beyond. To be exact, the adjusted closing price in a particular day differs greatly from the price it was at the same day in the previous week, thus lead us to this question:.

- **Question 2:** How volatile are the selected stocks?

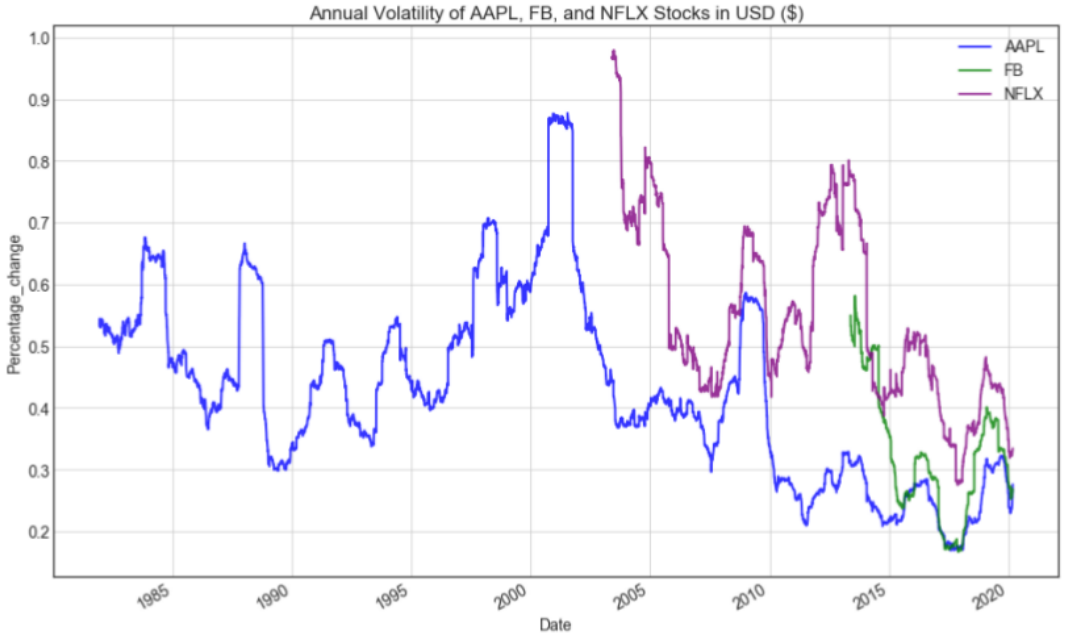
Volatility is a measure of dispersion around the mean or average return of a stock. Volatility can be measured using the **standard deviation**, which signals how tightly the price of a stock is grouped around the mean or moving average (MA). When prices are tightly bunched together, the standard deviation is small. Contrarily, when prices are widely spread apart, the standard deviation is large. **Rolling means** (or moving averages) are generally used to smooth out short-term fluctuations in time series data and highlight long-term trends.

Volatility = returns.std() * np.sqrt(days)

We use the `rolling()` method with `window = 25` 2 to compute the annual rolling mean of our daily stock price data to compare the annual volatility of the stocks.

In [19]:

```
# subset the selected stocks:
df = df_adj_close[df_adj_close['Symbol'].isin(['AAPL', 'FB', 'NFLX'])]

# calculate percentage change and volatility
for key, value in df.groupby('Symbol'):
    min_periods = 252  # the average trading days per year
    percentage_change = value['Adj_Close'].pct_change()
    volatility = percentage_change.rolling(min_periods).std()*np.sqrt(min_peri
ods)
    if key == 'AAPL':
        color = 'blue'
    elif key == 'FB':
        color = 'green'
    else:
        color = 'purple'
    # plot annual stock volatility
    volatility.plot(grid=True, linewidth=2, alpha=0.8, color=color)
    plt.legend(['AAPL', 'FB', 'NFLX'])
    plt.ylabel('Percentage_change')
    plt.title('Annual Volatility of AAPL, FB, and NFLX Stocks in USD ($)')
```



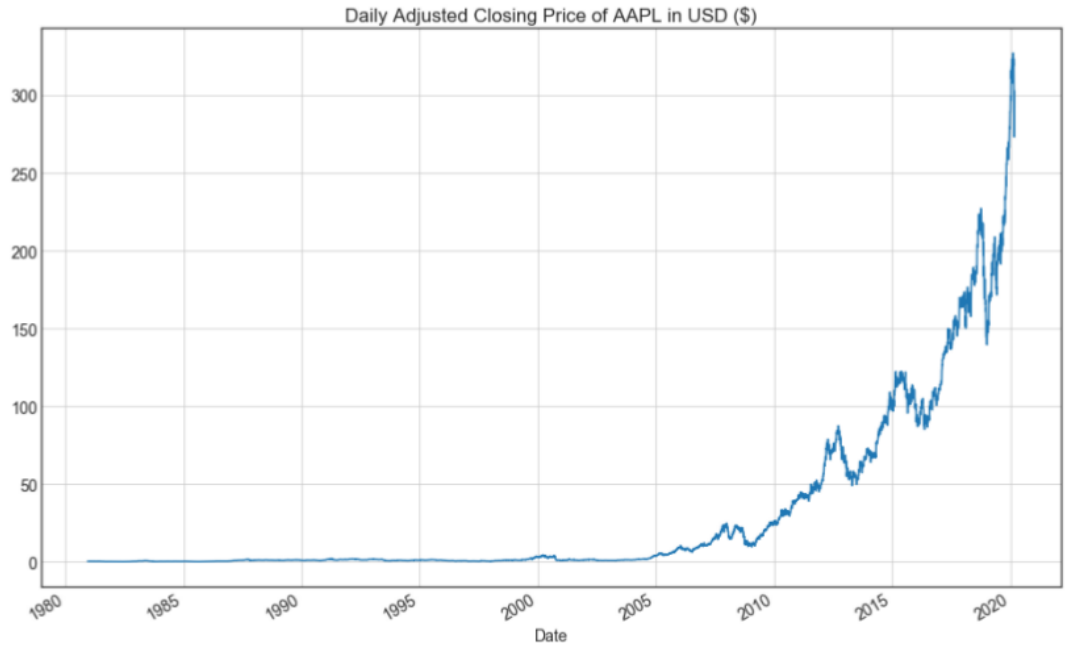
Confirmatory Data Analysis

For modeling purpose, we will chose **Apple (AAPL)** data as it has most datapoints due to longer years of trading and despite disruptions in society and business caused by the recent coronavirus pandemic has battered the stock market, AAPL is uniquely positioned to perform well. Regarded as one of the Big Four technology companies along with Google, Amazon, and Microsoft, Apple's stock price meteoric rise everytime the company releases its new product.

To recall, the line plot shows the adjusted closing price of AAPL *increases in general* since its commencement in the U.S stock market.

In [20]:
```
df_adj_close_viz['AAPL'].plot(grid=True)
plt.title('Daily Adjusted Closing Price of AAPL in USD ($)')
```

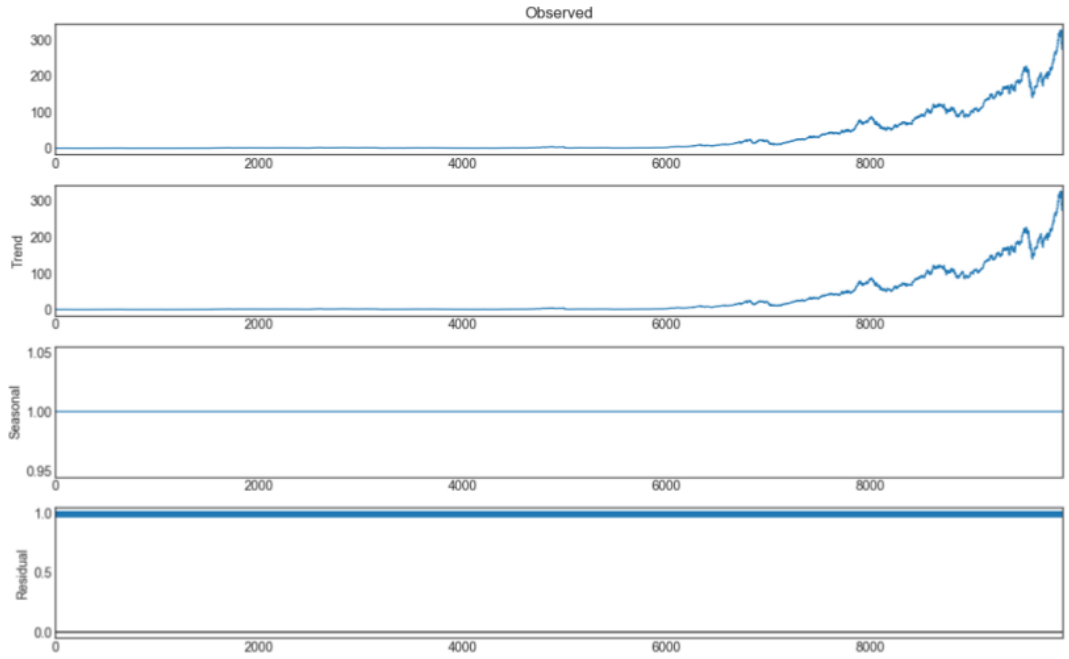
Out[20]:
```
Text(0.5, 1.0, 'Daily Adjusted Closing Price of AAPL in USD ($)')
```



However, there is no obvious pattern in the fluctuation of the AAPL stock price. When decomposing the time series into its components i.e. Trend, Seasonal and Residual, AAPL data shows an obvious upward trend but has insignificant seasonality (be it daily or annually) and unstable variance.

Time Series Components

In [21]:
```
# subset data for the company of interest
company_data = stock_data[stock_data['Symbol']=='AAPL']

# create a new dataframe contain only the 'Adj_Close' column, then convert the
dataframe to a series
data = company_data.filter(['Adj_Close'])
series = data.values

# decompose time series into its component
result = seasonal_decompose(series, model='multiplicative', freq=1)
result.plot()
plt.show()
```



To stabilize the variance in a time series (hence make it stationary), we can use logarithm transformation on the original data. Time series is stationary when its mean and variance are not a function of time (i.e. they are constant through time). Stationarity is important because most of the statistical methods to perform analysis and forecasting work on the assumption that the statistical properties (mean, variance, correlation, etc.) of the series are constant in time.

Time Series Stationarity

To test (and measure) the stationarity of a time series, ones need to check how the statistical properties vary in time. This can be done either by visual inspection of the datapoints or by performing **Dickey-Fuller test** - a statistical test that is normally used in Time Series Analysis to check for the stationarity of a univariate series by deciphering the presence of unit root in the series. Parameters such as **p-value**, **test statistic** and **crtitical values** derived from the test will be used to interpret the test result.

The null and alternate hypothesis of this test are:

- **H_0:** The time series has a unit root, thus it is non-stationary.
- **H_a:** The time series has no unit root, thus it is stationary.

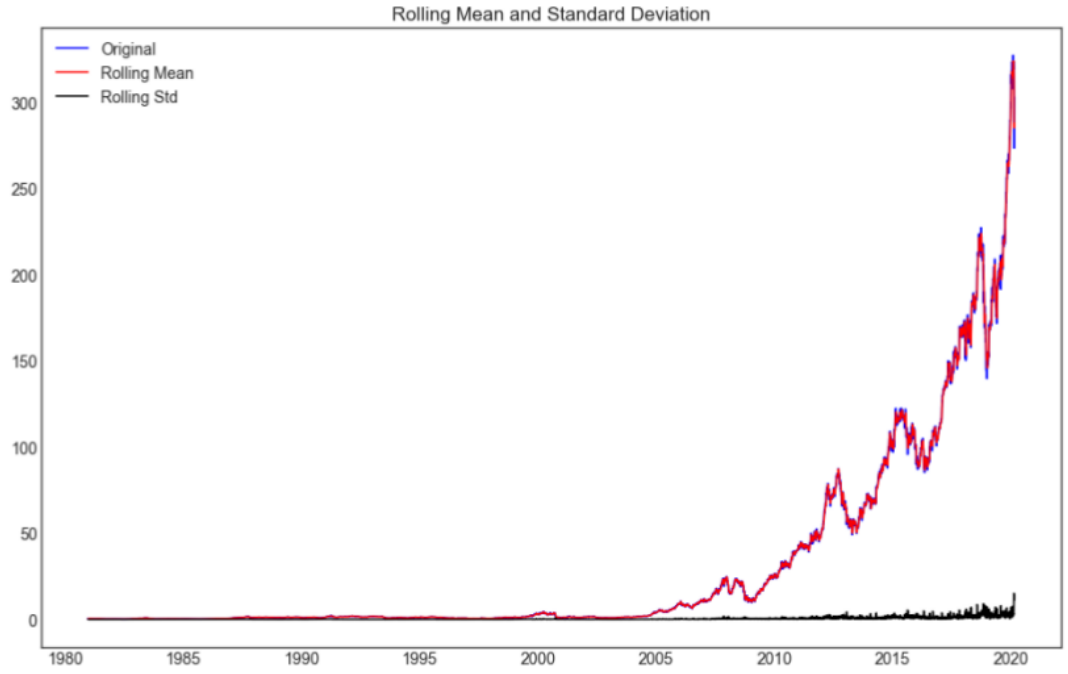
If `test statistic > critical value` , the **H_0** is accepted thus the series is regarded as non-stationary.

H_0 is rejected when `test statistic < critical value` , thus implying that the series is stationary.

In [22]:
```
# define function for ADF test

def adf_test(timeseries):
    # Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
```

In [23]:
```
# define function for testing data stationarity

def stationarity_test(timeseries):

    # calculate rolling statistics
    rolmean = timeseries.rolling(window=5).mean()
    rolstd = timeseries.rolling(window=5).std()

    # plot rolling statistics:
    plt.plot(timeseries, color='blue', label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    adf_test(timeseries)
```

In [24]:
```
# apply function to test data stationarity
stationarity_test(data)
```



20

```
Results of Dickey-Fuller Test:
Test Statistic                    4.684072
p-value                           1.000000
#Lags Used                       37.000000
Number of Observations Used    9851.000000
Critical Value (1%)              -3.431014
Critical Value (5%)              -2.861833
Critical Value (10%)             -2.566926
dtype: float64
```

By plotting the `standard deviation` and `mean` along with the original data points, we can see that both are time-dependent.

Result:

- **Test Statistic = 2.61** i.e. `test statistic > critical value`
- The null hypothesis is failed to be rejected thus clearly implying that the original time series is non-stationary.

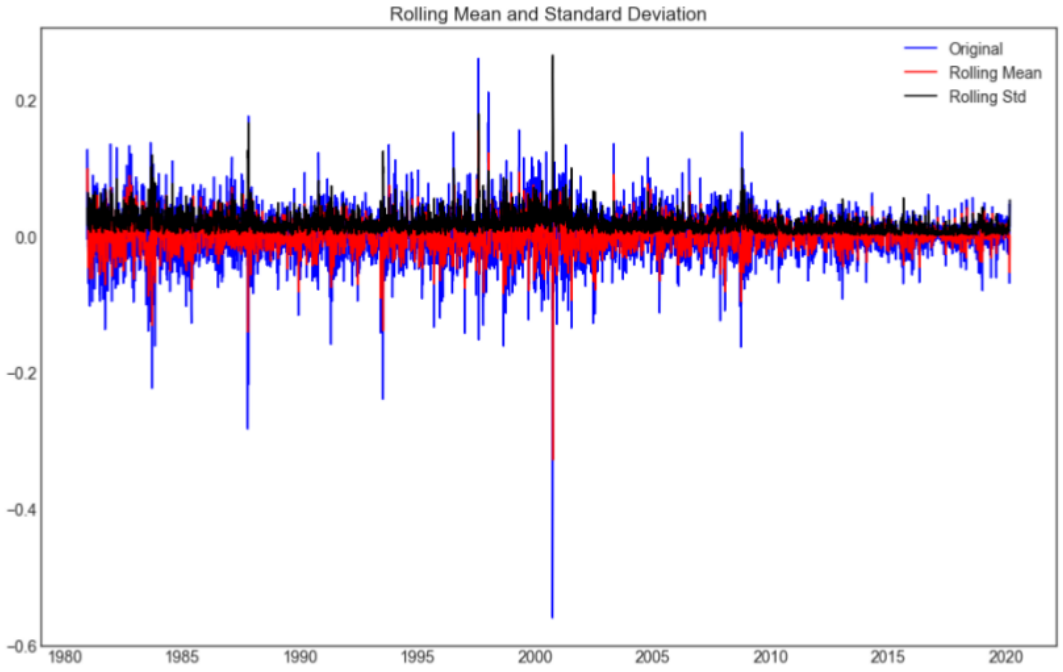
To make a time series stationary, **Transformation** and **Differencing** are the two common methods that can be applied. Trend and seasonality need to be removed from the data and in our case, trend is main culprit that is causing the non-stationarity.

- Transformation (log, square root, cube root, etc.) will reduce the trend by penalising higher values more than smaller values.
- Differencing method will take the difference of an observation at a particular instant with that at the previous point in time.

First, let's apply **Log Transformation** and **Moving Average Smoothing** to the original time series and check the stationarity of the output. Application of smoothing technique over the transformed data will remove noise that may be present.

In [25]:
```
# log-transformed the data
ts_log = np.log(data)

# replace infs with NaN
ts_log.replace([np.inf, -np.inf], np.nan, inplace=True)

# remove all the NaN values
ts_log.dropna(inplace=True)

# get the moving average of the series
moving_avg = ts_log.rolling(5).mean()

# subtract the moving average of the log-transformed dataframe
ts_log_moving_avg_diff = ts_log - moving_avg

# remove all the NaN values
ts_log_moving_avg_diff.dropna(inplace=True)
```

In [26]:
```
# apply function to test data stationarity on the log-transformed series
stationarity_test(ts_log_moving_avg_diff)
```



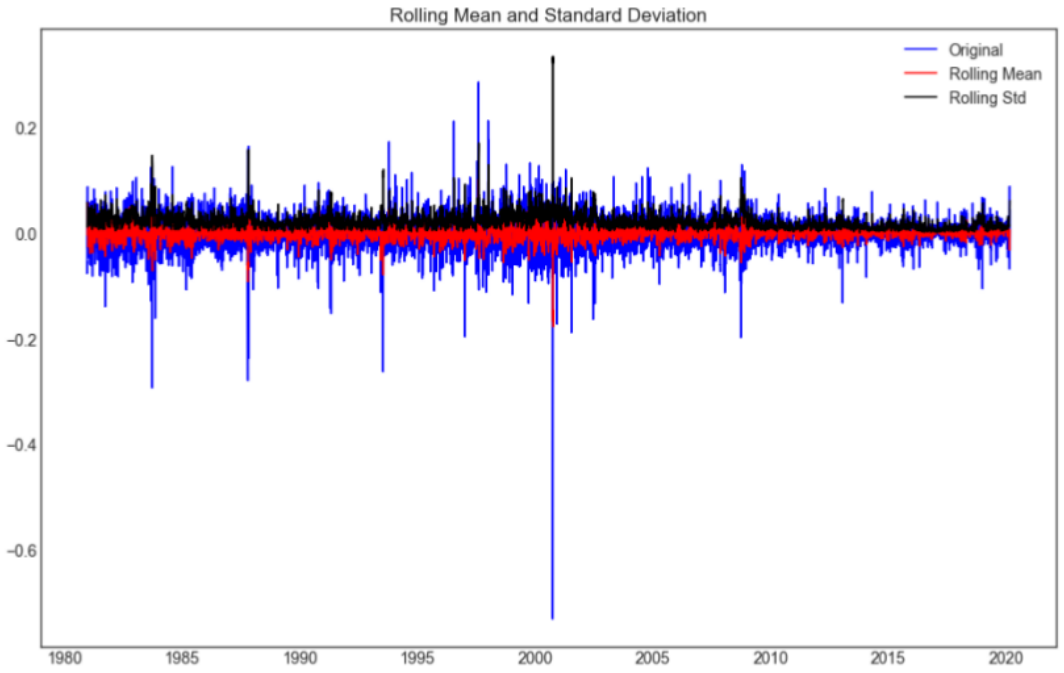

```
Results of Dickey-Fuller Test:
Test Statistic                  -25.027991
p-value                           0.000000
#Lags Used                       11.000000
Number of Observations Used    9873.000000
Critical Value (1%)              -3.431013
Critical Value (5%)              -2.861833
Critical Value (10%)             -2.566926
dtype: float64
```

Result:

- **Test Statistic = -12.09** i.e. `test statistic < critical value`
- The null hypothesis is rejected thus implying that the log-transformed time series is stationary.

Next, let's apply first order Differencing to the log-transformed data and check its stationarity:

In [27]:
```
# apply first order differencing to the log-transformed data
ts_log_diff = ts_log - ts_log.shift()
ts_log_diff.dropna(inplace=True)

# apply function to test data stationarity on the differencing series
stationarity_test(ts_log_diff)
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -22.636068
p-value                           0.000000
#Lags Used                       16.000000
Number of Observations Used    9871.000000
Critical Value (1%)              -3.431013
Critical Value (5%)              -2.861833
Critical Value (10%)             -2.566926
dtype: float64
```

Result:

- **Test Statistic = -12.86** i.e. `test statistic < critical value`
- The null hypothesis is rejected thus implying that the time series is stationary.

With combination of log transformation and differencing, the Test Statistic becomes more negative therefore there is stronger evidence to reject the null hypothesis of a unit root.

Predictive Model

Defining metrics for model evaluation

To evaluate model performance, ones can measure the 'forecast error' that is the difference between the actual values (y) and the predicted values (ŷ) output from the model. Forecast error is also termed as `loss function` .

In general, there are two kinds of forecast errors; *scale-dependent errors* and *percentage errors*. For our project, we will use `root mean squared error (RMSE) and mean absolute percentage error (MAPE)` as the model metrics. Each are the scale-dependent errors and the percentage errors respectively.

$$RMSE = \sqrt{\frac{1}{n}\sum e_t^2}$$

$$MAPE = \frac{1}{n}\sum \frac{|e_t|}{d_t}$$

In [28]:
```
# define functions to output metrics for model evaluation:

def root_mean_squared_error(actual, pred):
    rmse = np.sqrt(np.mean(pred-actual)**2)
    return rmse

def mean_absolute_percentage_error(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    mape = np.mean(np.abs((actual - pred) / actual)) * 100
    return mape
```

Modeling Method 1: Auto-ARIMA

ARIMA is a very popular statistical method for time series forecasting. The model take into account the past values to predict the future values.

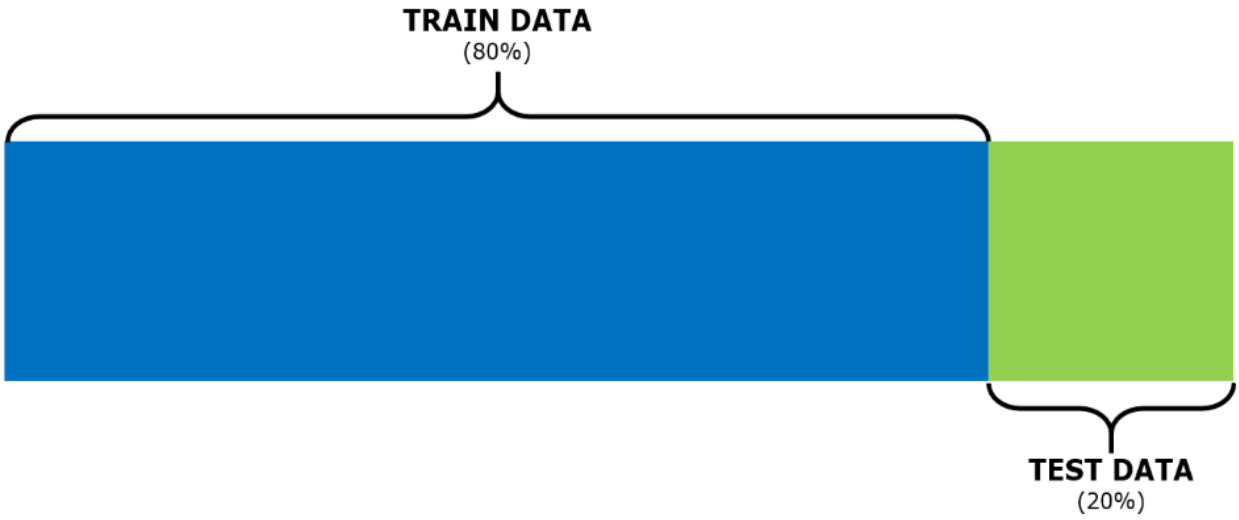
Three important components in ARIMA:

- **p** : associated with the auto-regressive (AR) aspect of the model, which incorporates past values to forecast the next value.
- **d** : associated with the integrated (I) part of the model, which is related to the order of differencing to apply to a time series.
- **q** : associated with the moving average (MA) part of the model, which uses past forecast errors to predict the future value.

Auto-ARIMA (https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html) automatically selects the best combination of (p,q,d) that provides the least error, plus it does not consume a lot of time than the manual parameter tuning in ARIMA. `Auto-ARIMA` automatically conducts differencing tests (i.e. Augmented Dickey-Fuller (ADF) test) to determine the order of differencing, d, and then fitting models within ranges of defined `start_p` , `max_p` , `start_q` , `max_q` ranges.

If the seasonal option is enabled, auto-ARIMA also seeks to identify the optimal `P` and `Q` hyper-parameters after conducting the Canova-Hansen to determine the optimal order of seasonal differencing, `D` .

In [29]:
```
# Get the number of rows to train the model on
training_data_len = math.ceil(len(data) * .8)

# subset training data
train_data = data[0:training_data_len]

# subset testing data
test_data = data[training_data_len:]

# convert each dataset to a numpy array
train, test = train_data.values, test_data.values
```


TRAIN DATA
(80%)
TEST DATA
(20%)

In [30]:
```
# initialize Auto-ARIMA model:
model = auto_arima(train, start_p=1, start_q=1, max_p=3, max_q=3,
                   m=12, start_P=0, seasonal=False, d=1, D=1, trace=True,
                   error_action='ignore', suppress_warnings=True)

# fit model to the training series
model.fit(train)
```

```
Performing stepwise search to minimize aic
Fit ARIMA: (1, 1, 1)x(0, 0, 0, 0) (constant=True); AIC=762.096, BIC=790.000,
Time=3.297 seconds
Fit ARIMA: (0, 1, 0)x(0, 0, 0, 0) (constant=True); AIC=790.269, BIC=804.221,
Time=0.520 seconds
Fit ARIMA: (1, 1, 0)x(0, 0, 0, 0) (constant=True); AIC=785.152, BIC=806.080,
Time=0.882 seconds
Fit ARIMA: (0, 1, 1)x(0, 0, 0, 0) (constant=True); AIC=785.326, BIC=806.254,
Time=0.541 seconds
Fit ARIMA: (0, 1, 0)x(0, 0, 0, 0) (constant=False); AIC=797.589, BIC=804.565,
Time=0.211 seconds
Fit ARIMA: (2, 1, 1)x(0, 0, 0, 0) (constant=True); AIC=773.010, BIC=807.890,
Time=3.712 seconds
Fit ARIMA: (1, 1, 2)x(0, 0, 0, 0) (constant=True); AIC=772.739, BIC=807.619,
Time=3.547 seconds
Fit ARIMA: (0, 1, 2)x(0, 0, 0, 0) (constant=True); AIC=786.151, BIC=814.055,
Time=0.915 seconds
Fit ARIMA: (2, 1, 0)x(0, 0, 0, 0) (constant=True); AIC=786.022, BIC=813.926,
Time=1.196 seconds
Fit ARIMA: (2, 1, 2)x(0, 0, 0, 0) (constant=True); AIC=757.038, BIC=798.894,
Time=6.819 seconds
Fit ARIMA: (3, 1, 2)x(0, 0, 0, 0) (constant=True); AIC=759.185, BIC=808.017,
Time=7.903 seconds
Fit ARIMA: (2, 1, 3)x(0, 0, 0, 0) (constant=True); AIC=754.159, BIC=802.991,
Time=8.486 seconds
Fit ARIMA: (1, 1, 3)x(0, 0, 0, 0) (constant=True); AIC=774.347, BIC=816.203,
Time=4.979 seconds
Fit ARIMA: (3, 1, 3)x(0, 0, 0, 0) (constant=True); AIC=728.877, BIC=784.685,
Time=7.939 seconds
Total fit time: 50.949 seconds
```

Out[30]:
```
ARIMA(maxiter=50, method='lbfgs', order=(3, 1, 3), out_of_sample_size=0,
      scoring='mse', scoring_args=None, seasonal_order=(0, 0, 0, 0),
      start_params=None, suppress_warnings=True, trend=None,
      with_intercept=True)
```

AIC and **BIC** are both penalized-likelihood criteria:

- **AIC** is an estimate of a constant plus the relative distance between the unknown true likelihood function of the data and the fitted likelihood function of the model, so that a lower AIC means a model is considered to be closer to the truth.
- **BIC** is an estimate of a function of the posterior probability of a model being true, under a certain Bayesian setup, so that a lower BIC means that a model is considered to be more likely to be the true model

In [31]:
```
# make predictions based on the fitted model
predictions = model.predict(n_periods = test.shape[0])
```

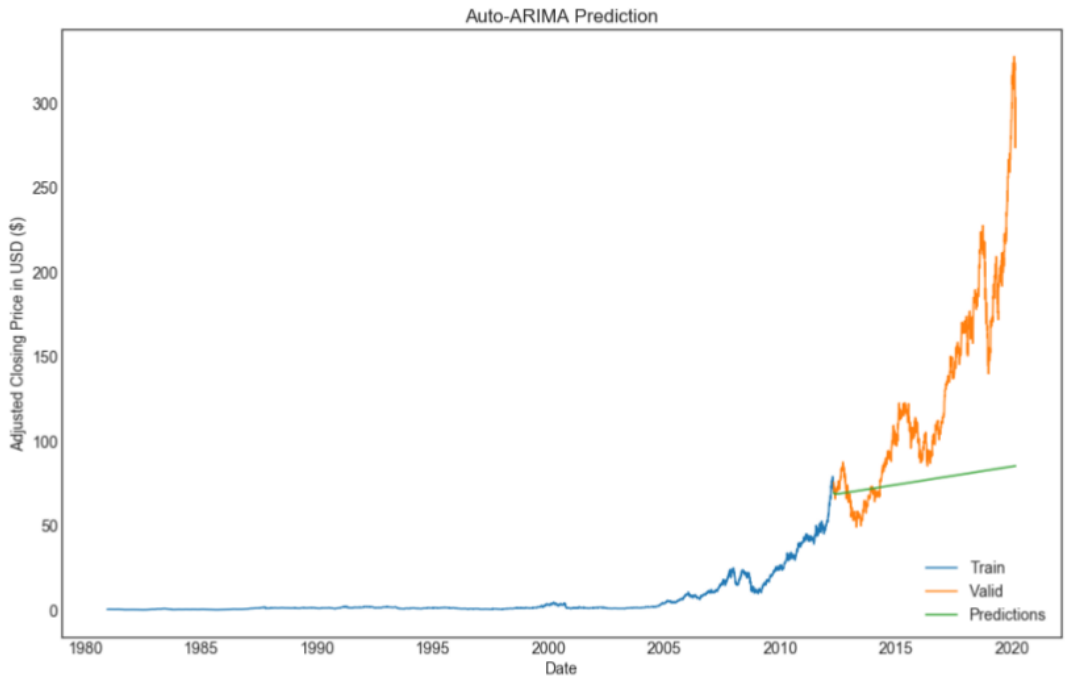
In [32]:
```
# output model metrics for model evaluation
arima_rmse = root_mean_squared_error(test, predictions)
arima_mape = mean_absolute_percentage_error(test, predictions)

output=[]
output.append({'Model': 'Auto-Arima', 'RMSE': arima_rmse, 'MAPE': arima_mape,
'Predictions': predictions})
arima_metrics = pd.DataFrame(output)
arima_metrics
```

Out[32]:

	Model	RMSE	MAPE	Predictions
0	Auto-Arima	49.698489	35.799058	[69.2334352975805, 69.2701611222995, 69.205991...

In [33]:
```
# plot Train, Valid and Predictions data:
pred_data = pd.DataFrame(predictions, index=test_data.index, columns=['Predict
ion'])

plt.plot(train_data)
plt.plot(test_data)
plt.plot(pred_data)
plt.title('Auto-ARIMA Prediction')
plt.xlabel('Date')
plt.ylabel('Adjusted Closing Price in USD ($)')
plt.legend(['Train', 'Valid', 'Predictions'], loc='lower right')
```

Out[33]: <matplotlib.legend.Legend at 0x1af5fea6a08>



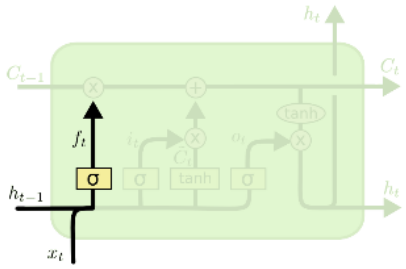
An auto-ARIMA model uses past data to understand the pattern in the time series. Using these values, the model captured an increasing trend in the series (but does not focus on the seasonal part). Although the predictions using this technique are far better than ARIMA or other common machine learning models, these predictions are still not close to the real values.

Modeling Method 3: Long short-term memory (LSTM)

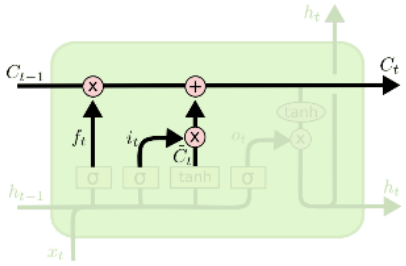
Long Short Term Memory networks or `LSTM` is an advanced version of Recurrent Neural Network (RNN) that is capable of learning long-term dependencies by storing important past information and forgetting the information that is not important. LSTM has been widely used for sequence prediction problems is are proven to be extremely effective.

Architecturally, LSTM consists of cells with three gates that regulate the 'movement' of information in the network:

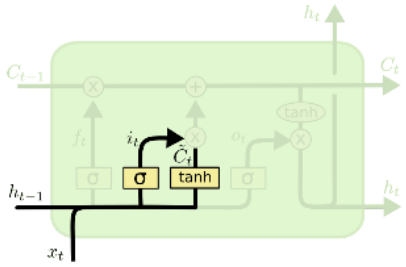
- **Input gate** adds information to the cell state
- **Forget gate** removes information that is no longer required by the model
- **Output gate** selects the information to be shown as output



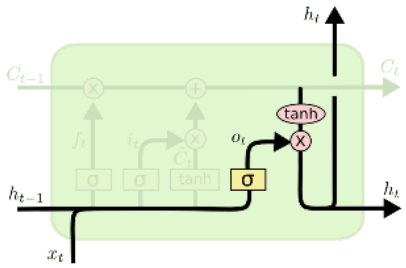
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



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



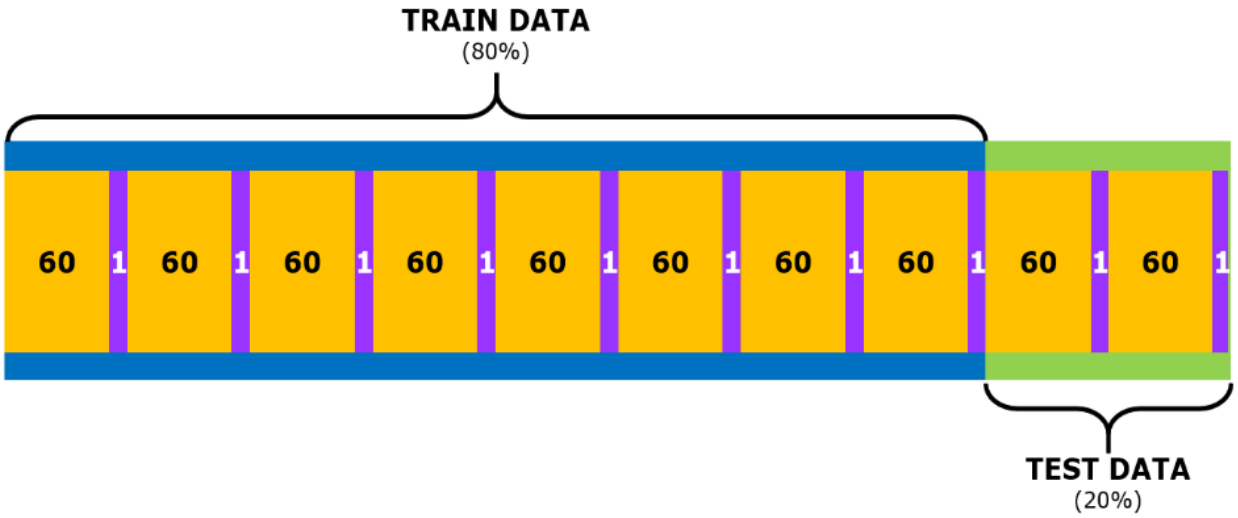
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



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

Christopher Olah (https://colah.github.io/posts/2015-08-Understanding-LSTMs/) has paraphrased the complex concept of LSTM into digestible information.

In [34]:
```
# convert the dataframe to a numpy array
dataset = data.values

# Get the number of rows to train the model on
training_data_len = math.ceil(len(dataset) * .8)

# scale the data
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)

# create training dataset
# scale the train data
train_data = scaled_data[0:training_data_len, :]
# append the past-60-day values to X_train dataset and every 61st-day value to
y_train
X_train = []
y_train = []
for i in range(60, len(train_data)):
    X_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i,0])
# convert X_train and y_train to numpy arrays
X_train, y_train = np.array(X_train), np.array(y_train)
# reshape X_train dataset
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

# create testing dataset
# scale the test data and create X_test and y_test dataset
test_data = scaled_data[training_data_len-60:,:]

# Create the dataset X_test and y_test
X_test = []
y_test = dataset[training_data_len:,:]
for i in range(60, len(test_data)):
    X_test.append(test_data[i-60:i, 0])

# convert X_test and y_test to numpy array
X_test = np.array(X_test)

# reshape X_test dataset
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```


TRAIN DATA
(80%)
60
1
60
1
60
1
60
1
60
1
60
1
60
1
60
1
60
1
60
1
TEST DATA
(20%)

In [35]:
```
# Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, batch_size=1, epochs=4)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0622 12:21:09.992004 22404 deprecation_wrapper.py:119] From C:\Users\norzari
fah.k\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\backend
\tensorflow_backend.py:74: The name tf.get_default_graph is deprecated. Pleas
e use tf.compat.v1.get_default_graph instead.

W0622 12:21:10.003489 22404 deprecation_wrapper.py:119] From C:\Users\norzari
fah.k\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\backend
\tensorflow_backend.py:517: The name tf.placeholder is deprecated. Please use
tf.compat.v1.placeholder instead.

W0622 12:21:10.005518 22404 deprecation_wrapper.py:119] From C:\Users\norzari
fah.k\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\backend
\tensorflow_backend.py:4138: The name tf.random_uniform is deprecated. Please
use tf.random.uniform instead.

W0622 12:21:10.330811 22404 deprecation_wrapper.py:119] From C:\Users\norzari
fah.k\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\optimize
rs.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v
1.train.Optimizer instead.

W0622 12:21:10.483398 22404 deprecation.py:323] From C:\Users\norzarifah.k\Ap
pData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\python\ops
\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflow.py
thon.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
W0622 12:21:11.206403 22404 deprecation_wrapper.py:119] From C:\Users\norzari
fah.k\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\backend
\tensorflow_backend.py:986: The name tf.assign_add is deprecated. Please use
tf.compat.v1.assign_add instead.

W0622 12:21:11.292897 22404 deprecation_wrapper.py:119] From C:\Users\norzari
fah.k\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\backend
\tensorflow_backend.py:973: The name tf.assign is deprecated. Please use tf.c
ompat.v1.assign instead.


Epoch 1/4
7852/7852 [==============================] - 320s 41ms/step - loss: 3.9987e-0
5
Epoch 2/4
7852/7852 [==============================] - 312s 40ms/step - loss: 1.6999e-0
5
Epoch 3/4
7852/7852 [==============================] - 317s 40ms/step - loss: 1.1649e-0
5
Epoch 4/4
7852/7852 [==============================] - 333s 42ms/step - loss: 7.1671e-0
6
```

In [36]:

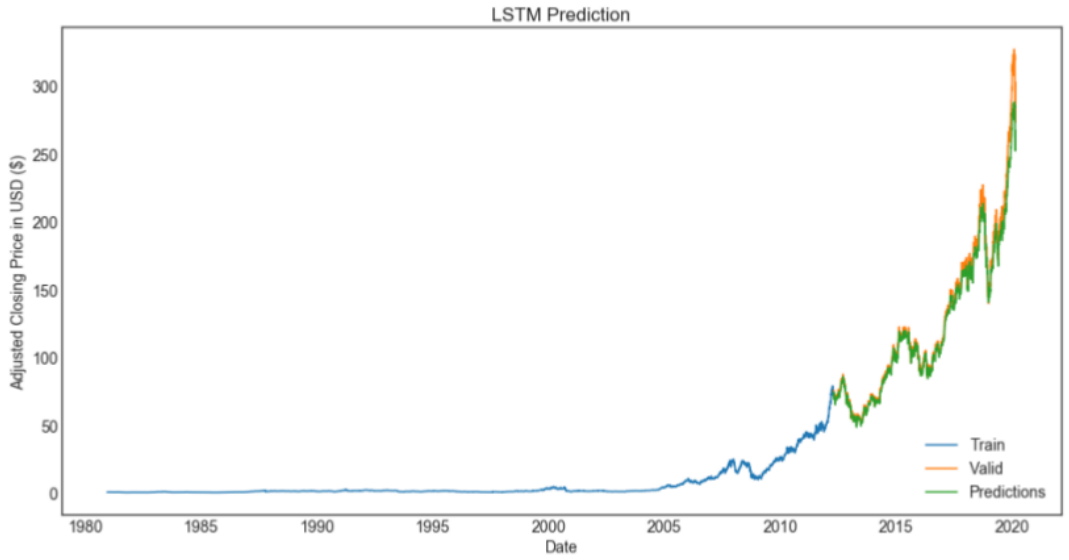
```
# Get the predicted price value from the model
predictions = model.predict(X_test)

# Transform the scaled predictions to the actual value
predictions = scaler.inverse_transform(predictions)
```

In [37]:
```
# output model metrics for model evaluation
lstm_rmse = root_mean_squared_error(y_test, predictions)
lstm_mape = mean_absolute_percentage_error(y_test, predictions)

output=[]
output.append({'Model': 'LSTM', 'RMSE': lstm_rmse, 'MAPE': lstm_mape, 'Predict
ions': predictions})
lstm_metrics = pd.DataFrame(output)
lstm_metrics
```

Out[37]:

	Model	RMSE	MAPE	Predictions
0	LSTM	4.422914	2.959188	[[69.09267], [73.1252], [74.214424], [73.65232...

In [38]:
```
# Plot the data
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions

# Visualise the data
plt.figure(figsize=(16,8))
plt.title('LSTM Prediction')
plt.xlabel('Date')
plt.ylabel('Adjusted Closing Price in USD ($)')
plt.plot(train['Adj_Close'])
plt.plot(valid[['Adj_Close', 'Predictions']])
plt.legend(['Train', 'Valid', 'Predictions'], loc='lower right')
plt.show()
```



Evaluation and Analysis

In comparison to Auto-ARIMA, LSTM gives better predictions; supported by the metrics values:

In [39]:
```
# dataframe for model metrics
frames = [arima_metrics, lstm_metrics]
compare_metrics = pd.concat(frames).reset_index(drop=True)
compare_metrics
```

Out[39]:

	Model	RMSE	MAPE	Predictions
0	Auto-Arima	49.698489	35.799058	[69.2334352975805, 69.2701611222995, 69.205991...
1	LSTM	4.422914	2.959188	[[69.09267], [73.1252], [74.214424], [73.65232...

Taking this into account, we will pick LSTM for model validation thus addressing this question:

- **Question 3:** What is the expected closing price of the selected stock at a given date?

In [42]:
```
# Predict the stock price for 2020-02-04

company_subset = data.loc['2010-01-01':'2020-03-02']

# Get the last 60-day closing price values and convert the dataframe to an arr
ay
last_60_days = company_subset[-60:].values

# Scale the data to be values between 0 and 1
last_60_days_scaled = scaler.transform(last_60_days)

# Create a list for the the last 60-day
X_test = []
X_test.append(last_60_days_scaled)
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Fit in the model to get the predicted scaled price
pred_price = model.predict(X_test)

# Unscale the data
pred_price = scaler.inverse_transform(pred_price)
print('The predicted stock price of AAPL on 2020-03-03')
print(pred_price)

# actual stock price for 2019-12-18
print()
print('The actual stock price of AAPL on 2020-03-03')
data.loc['2020-03-02':'2020-03-04']
```

```
The predicted stock price of AAPL on 2020-03-03
[[267.16962]]

The actual stock price of AAPL on 2020-03-03
```

Out[42]:

Date	Adj_Close
2020-03-02	298.809998
2020-03-03	289.320007
2020-03-04	302.739990

Explanation:

Although the predicted value **USD267.17** of the adjusted closing price varies from the actual value **USD289.32** at the selected date of **March 3rd 2020**, the predicted price is actually decreases than the actual adjusted closing price of the previous day, hence the model somehow able to detect the decreasing pattern of the stock price.

The End

Stock Market Analysis

ORIGINALITY REPORT

43%	30%	12%	41%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Sydney Student Paper	3%
2	Submitted to Harrisburg University of Science and Technology Student Paper	3%
3	www.analyticsvidhya.com Internet Source	3%
4	Submitted to Middlesex University Student Paper	3%
5	Submitted to University of Melbourne Student Paper	3%
6	methodology.psu.edu Internet Source	1%
7	Submitted to University of Huddersfield Student Paper	1%
8	machinelearningmastery.com Internet Source	1%
9	towardsdatascience.com	

	Internet Source	1%
10	www.investopedia.com Internet Source	1%
11	Submitted to University of Technology, Sydney Student Paper	1%
12	Danish Haroon. "Python Machine Learning Case Studies", Springer Science and Business Media LLC, 2017 Publication	1%
13	alkaline-ml.com Internet Source	1%
14	medium.com Internet Source	1%
15	gitlab.fdmci.hva.nl Internet Source	1%
16	www.researchgate.net Internet Source	1%
17	Submitted to City University Student Paper	1%
18	Submitted to Asian Institute of Management Student Paper	1%
19	Submitted to Bahcesehir University Student Paper	<1%

20	ashwin-ks.github.io Internet Source	<1%
21	Submitted to University of Brighton Student Paper	<1%
22	Submitted to University of Nevada, Las Vegas Student Paper	<1%
23	Submitted to University of Glasgow Student Paper	<1%
24	Submitted to CSU, San Jose State University Student Paper	<1%
25	Submitted to National College of Ireland Student Paper	<1%
26	Submitted to University of Oklahoma Student Paper	<1%
27	stackoverflow.com Internet Source	<1%
28	Submitted to California State University, Fresno Student Paper	<1%
29	Submitted to Bocconi University Student Paper	<1%
30	Submitted to University of Strathclyde Student Paper	<1%
31	docs.h2o.ai Internet Source	<1%

32	mc.ai Internet Source	<1%
33	Submitted to University of Warwick Student Paper	<1%
34	Submitted to Regis University Student Paper	<1%
35	Submitted to Queen Mary and Westfield College Student Paper	<1%
36	Submitted to Imperial College of Science, Technology and Medicine Student Paper	<1%
37	dspace.bracu.ac.bd Internet Source	<1%
38	Manuel R. Vargas, Beatriz S. L. P. de Lima, Alexandre G. Evsukoff. "Deep learning for stock market prediction from financial news articles", 2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2017 Publication	<1%
39	Submitted to University of Westminster Student Paper	<1%
40	ybchen.com Internet Source	<1%

41	Ervin Varga. "Practical Data Science with Python 3", Springer Science and Business Media LLC, 2019 Publication	<1%
42	www.ijert.org Internet Source	<1%
43	Submitted to University of Utah Student Paper	<1%
44	www.hillsclerk.com Internet Source	<1%
45	Submitted to Boston University Student Paper	<1%
46	Submitted to National Institute of Technology Delhi Student Paper	<1%
47	Submitted to Napier University Student Paper	<1%
48	Submitted to Rikkyo University Student Paper	<1%
49	kanoki.org Internet Source	<1%
50	Submitted to University of Missouri, Kansas City Student Paper	<1%
51	bioinformaticsinstitute.ru	

	Internet Source	<1%
52	Submitted to University College London Student Paper	<1%
53	Submitted to Liverpool John Moores University Student Paper	<1%
54	data.cityoftacoma.org Internet Source	<1%
55	mutualfunds.com Internet Source	<1%
56	Submitted to Coventry University Student Paper	<1%
57	Submitted to Infile Student Paper	<1%
58	budgeting.thenest.com Internet Source	<1%
59	Submitted to University of West London Student Paper	<1%
60	www.mdpi.com Internet Source	<1%
61	Submitted to Bilkent University Student Paper	<1%
62	Submitted to National University of Ireland,	<1%

	Maynooth Student Paper	
63	Manohar Swamynathan. "Mastering Machine Learning with Python in Six Steps", Springer Science and Business Media LLC, 2019 Publication	<1%
64	Submitted to North West University Student Paper	<1%
65	Submitted to Monash University Student Paper	<1%
66	Submitted to Sim University Student Paper	<1%
67	www.toptal.com Internet Source	<1%
68	Ekaba Bisong. "Building Machine Learning and Deep Learning Models on Google Cloud Platform", Springer Science and Business Media LLC, 2019 Publication	<1%
69	Submitted to UT, Dallas Student Paper	<1%
70	Submitted to De Montfort University Student Paper	<1%
71	Submitted to Universiti Tenaga Nasional Student Paper	<1%

72	Submitted to The Hong Kong Polytechnic University Student Paper	<1%
73	Submitted to University of Hull Student Paper	<1%
74	www.czxa.top Internet Source	<1%
75	Submitted to The University of Manchester Student Paper	<1%
76	git.fh-muenster.de Internet Source	<1%
77	Submitted to University of Leicester Student Paper	<1%
78	Submitted to Queen's University of Belfast Student Paper	<1%
79	Submitted to Indian School of Mines Student Paper	<1%
80	necromuralist.github.io Internet Source	<1%
81	Submitted to Trinity College Dublin Student Paper	<1%
82	Cavalcante, Rodolfo C., Rodrigo C. Brasileiro, Victor L.F. Souza, Jarley P. Nobrega, and Adriano L.I. Oliveira. "Computational Intelligence	<1%

and Financial Markets: A Survey and Future Directions", Expert Systems with Applications, 2016.
Publication

83	Submitted to University of Hong Kong Student Paper	<1%
84	Submitted to Indian Institute of Technology, Ropar Student Paper	<1%
85	Submitted to University of Edinburgh Student Paper	<1%
86	Submitted to Mondragon Unibertsitatea Student Paper	<1%
87	Submitted to The Robert Gordon University Student Paper	<1%
88	Submitted to University of Kent at Canterbury Student Paper	<1%
89	Submitted to Noroff University College Student Paper	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		