Exercise 10

10.1 Problem Statement:

Solve the stock price forecasting problem using statistical techniques – Maximum Likelihood estimation after understanding the distribution of the data.

10.2 Description of Machine Learning Algorithm:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

Naming the Variables. There are many names for a regression's dependent variable. It may be called an outcome variable, criterion variable, endogenous variable, or regressand. The independent variables can be called exogenous variables, predictor variables, or regressors.

Three major uses for regression analysis are (1) determining the strength of predictors, (2) forecasting an effect, and (3) trend forecasting.

Mathematically, we can represent a linear regression as:

$y= a0+a1x+ \varepsilon$

Y= Dependent Variable (Target Variable) X= Independent Variable (predictor Variable) a0= intercept of the line (Gives an additional degree of freedom) a1 = Linear regression coefficient (scale factor to each input value).

ε = random error

10.3 Description of Data Set:

Title of the data set: AAPL data

The dataset consists of the data of the Apple Inc. stock prices for 1257 days.

10.4 Data Preprocessing and Exploratory Data Analysis:

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Major Tasks in Data Preprocessing:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

Exploratory data analysis (EDA) is a technique that data professionals can use to understand a dataset before they start to model it. Some people refer to EDA as data exploration. The goal of conducting EDA is to

determine the characteristics of the dataset. Conducting EDA can help data analysts make predictions and assumptions about data. Often, EDA involves data visualization, including creating graphs like histograms, scatter plots and box plots.

Major Tasks in EDA:

1. Observe your dataset
2. Find any missing values
3. Categorize your values
4. Find the shape of your dataset
5. Identify relationships in your dataset
6. Locate any outliers in your dataset

10.5 Machine Learning Package Used for Model building:

For the linear regression model we use scikit-learn

Scikit-learn is an open source Machine Learning Python package that offers functionality supporting supervised and unsupervised learning. Additionally, it provides tools for model development, selection and evaluation as well as many other utilities including data pre-processing functionality.

More specifically, scikit-learn's main functionality includes classification, regression, clustering, dimensionality reduction, model selection and pre-processing. sThe library is very simple to use and most importantly efficient as it is built on **NumPy**, **SciPy** and **matplotlib.**

10.6 Implementation:

```
#importing the libraries
import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from google.colab import drive
drive.mount('/content/drive')

# Data Handling: Load CSV
df = pd.read_csv("/content/drive/MyDrive/ML Lab dataset/AAPL.csv")

# get to know list of features, data shape, stat. description.
print(df.shape)

print("First 5 lines:")
print(df.head(5))

print("describe: ")
print(df.describe())

print("info: ")
```

```
print(df.info())
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
(1258, 15)
First 5 lines:
   Unnamed: 0 symbol                       date    close     high     low  \
0           0   AAPL  2015-05-27 00:00:00+00:00  132.045  132.260  130.05
1           1   AAPL  2015-05-28 00:00:00+00:00  131.780  131.950  131.10
2           2   AAPL  2015-05-29 00:00:00+00:00  130.280  131.450  129.90
3           3   AAPL  2015-06-01 00:00:00+00:00  130.535  131.390  130.05
4           4   AAPL  2015-06-02 00:00:00+00:00  129.960  130.655  129.32

     open    volume    adjClose     adjHigh      adjLow     adjOpen  \
0  130.34  45833246  121.682558  121.880685  119.844118  120.111360
1  131.86  30733309  121.438354  121.595013  120.811718  121.512076
2  131.23  50884452  120.056069  121.134251  119.705890  120.931516
3  131.20  32112797  120.291057  121.078960  119.844118  120.903870
4  129.86  33667627  119.761181  120.401640  119.171406  119.669029

   adjVolume  divCash  splitFactor
0   45833246      0.0          1.0
1   30733309      0.0          1.0
2   50884452      0.0          1.0
3   32112797      0.0          1.0
4   33667627      0.0          1.0
describe:
        Unnamed: 0        close         high          low         open  \
count  1258.000000  1258.000000  1258.000000  1258.000000  1258.000000
mean    628.500000   167.723998   169.230475   166.039780   167.548266
std     363.297628    56.850796    57.500128    56.006773    56.612707
min       0.000000    90.340000    91.670000    89.470000    90.000000
25%     314.250000   116.327500   117.405000   115.602500   116.482500
50%     628.500000   160.485000   162.080000   158.974250   160.345000
75%     942.750000   199.785000   201.277500   198.170000   199.520000
max    1257.000000   327.200000   327.850000   323.350000   324.730000

             volume     adjClose      adjHigh       adjLow      adjOpen  \
count  1.258000e+03  1258.000000  1258.000000  1258.000000  1258.000000
mean   3.500397e+07   162.666715   164.131054   161.028013   162.493082
std    1.729100e+07    58.733820    59.402842    57.869246    58.494560
min    1.136204e+07    84.954351    86.205062    84.136216    84.634620
25%    2.359205e+07   109.484490   110.393556   107.962457   109.135002
50%    3.064771e+07   154.710645   156.091874   153.054341   154.410017
75%    4.100487e+07   196.960053   198.428438   195.281553   196.452903
max    1.622063e+08   326.337147   326.357095   322.497300   323.873661

          adjVolume      divCash  splitFactor
count  1.258000e+03  1258.000000       1258.0
mean   3.500397e+07     0.010477          1.0
std    1.729100e+07     0.083366          0.0
min    1.136204e+07     0.000000          1.0
25%    2.359205e+07     0.000000          1.0
50%    3.064771e+07     0.000000          1.0
75%    4.100487e+07     0.000000          1.0
max    1.622063e+08     0.820000          1.0
info:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Unnamed: 0    1258 non-null   int64  
 1   symbol        1258 non-null   object 
 2   date          1258 non-null   object 
 3   close         1258 non-null   float64
 4   high          1258 non-null   float64
 5   low           1258 non-null   float64
 6   open          1258 non-null   float64
 7   volume        1258 non-null   int64  
 8   adjClose      1258 non-null   float64
 9   adjHigh       1258 non-null   float64
 10  adjLow        1258 non-null   float64
 11  adjOpen       1258 non-null   float64
 12  adjVolume     1258 non-null   int64  
 13  divCash       1258 non-null   float64
 14  splitFactor   1258 non-null   float64
dtypes: float64(10), int64(3), object(2)
memory usage: 147.5+ KB
None
```

```
df
```

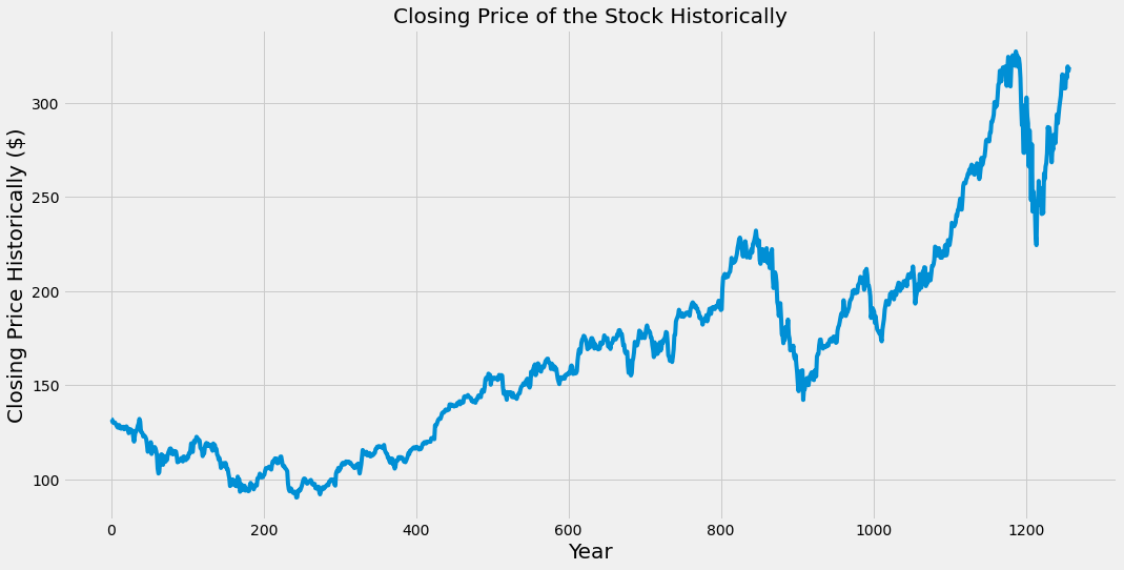
	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitFactor
0	0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.0500	130.34	45833246	121.682558	121.880685	119.844118	120.111360	45833246	0.0	1.0
1	1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.1000	131.86	30733309	121.438354	121.595013	120.811718	121.512076	30733309	0.0	1.0
2	2	AAPL	2015-05-29 00:00:00+00:00	130.280	131.450	129.9000	131.23	50884452	120.056069	121.134251	119.705890	120.931516	50884452	0.0	1.0
3	3	AAPL	2015-06-01 00:00:00+00:00	130.535	131.390	130.0500	131.20	32112797	120.291057	121.078960	119.844118	120.903870	32112797	0.0	1.0
4	4	AAPL	2015-06-02 00:00:00+00:00	129.960	130.655	129.3200	129.86	33667627	119.761181	120.401640	119.171406	119.669029	33667627	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1253	1253	AAPL	2020-05-18 00:00:00+00:00	314.960	316.500	310.3241	313.17	33843125	314.960000	316.500000	310.324100	313.170000	33843125	0.0	1.0
1254	1254	AAPL	2020-05-19 00:00:00+00:00	313.140	318.520	313.0100	315.03	25432385	313.140000	318.520000	313.010000	315.030000	25432385	0.0	1.0
1255	1255	AAPL	2020-05-20 00:00:00+00:00	319.230	319.520	316.2000	316.68	27876215	319.230000	319.520000	316.200000	316.680000	27876215	0.0	1.0
1256	1256	AAPL	2020-05-21 00:00:00+00:00	316.850	320.890	315.8700	318.66	25672211	316.850000	320.890000	315.870000	318.660000	25672211	0.0	1.0
1257	1257	AAPL	2020-05-22 00:00:00+00:00	318.890	319.230	315.3500	315.77	20450754	318.890000	319.230000	315.350000	315.770000	20450754	0.0	1.0

1258 rows × 15 columns

Data Preprocessing

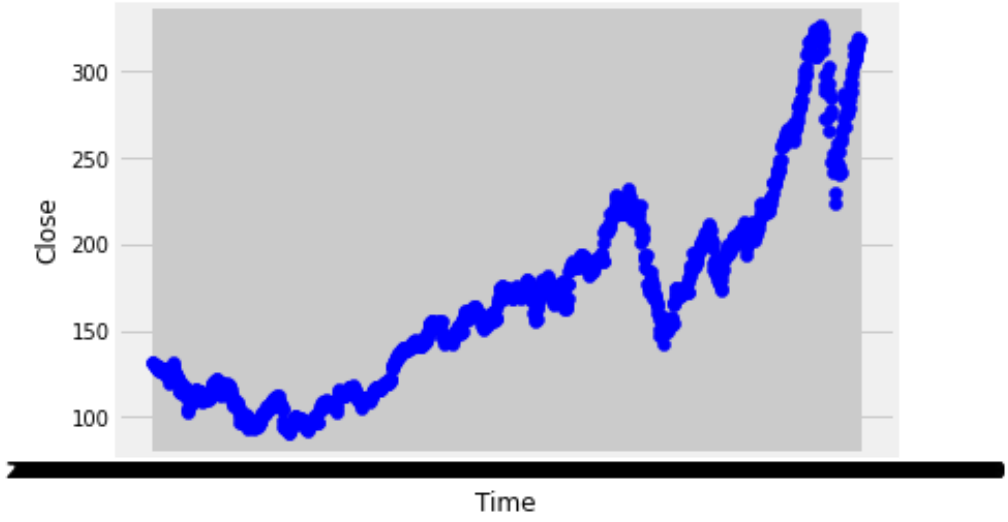
```
df.isnull().sum()
Unnamed: 0     0
symbol         0
date           0
close          0
high           0
low            0
open           0
volume         0
adjClose       0
adjHigh        0
adjLow         0
adjOpen        0
adjVolume      0
divCash        0
splitFactor    0
dtype: int64
```

```
df = df.loc[(df['symbol'] == 'AAPL')]
df = df.drop(columns=['symbol'])
df = df[['date','open','close','low','volume','high']]
plt.figure(figsize=(16,8))
plt.title('Closing Price of the Stock Historically')
plt.plot(df['close'])
plt.xlabel('Year', fontsize=20)
plt.ylabel('Closing Price Historically ($)', fontsize=20)
plt.show()
```



Regression Model

```
from sklearn import metrics
%matplotlib inline
import matplotlib.pyplot as plt
import math
plt.scatter(df.date, df.close,  color='blue')
plt.xlabel("Time")
plt.ylabel("Close")
plt.show()
```



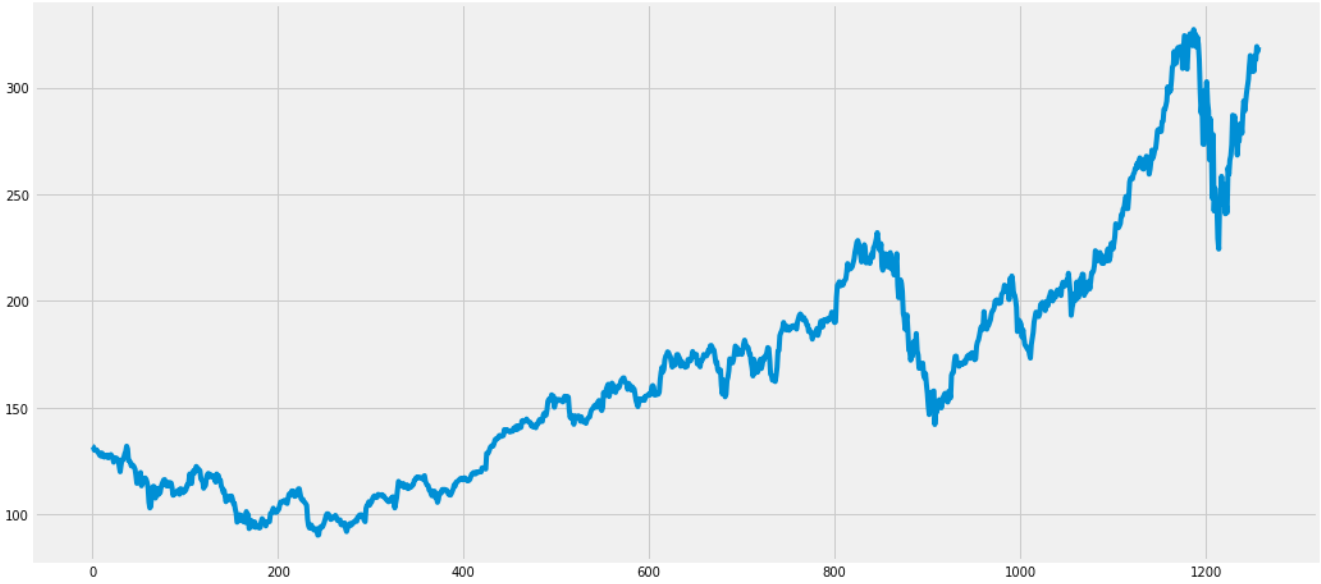
```
df.shape
(1258, 6)
```

```
df['date'] = pd.to_datetime(df.date)
df.head()
```

	date	open	close	low	volume	high
0	2015-05-27 00:00:00+00:00	130.34	132.045	130.05	45833246	132.260
1	2015-05-28 00:00:00+00:00	131.86	131.780	131.10	30733309	131.950
2	2015-05-29 00:00:00+00:00	131.23	130.280	129.90	50884452	131.450
3	2015-06-01 00:00:00+00:00	131.20	130.535	130.05	32112797	131.390
4	2015-06-02 00:00:00+00:00	129.86	129.960	129.32	33667627	130.655

```
print(len(df))
1258
```

```
df['close'].plot(figsize=(16,8))
<matplotlib.axes._subplots.AxesSubplot at 0x7f69b37da310>
```




```
x1 = df[['open', 'high','low', 'volume']]
y1 = df['close']
from sklearn.model_selection import train_test_split
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, random_state =
0)
x1_train.shape
(943, 4)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score
regression = LinearRegression()
regression.fit(x1_train, y1_train)
LinearRegression()
```

```
print(regression.coef_)
[-5.72199532e-01  6.87109453e-01  8.86463283e-01 -1.08492815e-09]

print(regression.intercept_)
0.11094825488586935

predicted=regression.predict(x1_test)
print(x1_test)
     open     high       low    volume
5     130.660  130.940  129.9000  30983542
494   152.450  154.070  152.3100  25596687
52    116.530  119.990  116.5300  54951597
985   206.830  207.760  205.1200  18543206
186    96.310   96.900   95.9200  34280758
...       ...      ...       ...       ...
744   186.550  187.400  185.2200  23211241
1002  183.520  184.349  180.2839  38612290
922   154.110  154.480  151.7400  25441549
459   139.845  141.600  139.7600  25860165
911   149.560  151.820  148.5200  41025314

[315 rows x 4 columns]

predicted.shape
(315,)

dframe = pd.DataFrame(y1_test,predicted)
dfr=pd.DataFrame({'Actual_Price':y1_test, 'Predicted_Price':predicted})
print(dfr)
      Actual_Price  Predicted_Price
5            130.12       130.435435
494          153.95       153.731535
52           119.72       119.118748
985          205.28       206.328009
186           96.88        96.575683
...             ...              ...
744          187.36       186.296984
1002         183.09       181.541997
922          152.70       152.558283
459          141.42       141.250455
911          150.75       150.462761

[315 rows x 2 columns]

dfr.head(10)
```

	Actual_Price	Predicted_Price
5	130.12	130.435435
494	153.95	153.731535
52	119.72	119.118748
985	205.28	206.328009
186	96.88	96.575683
18	127.61	127.767290
317	106.94	107.034249
511	154.45	155.399809
364	111.59	112.135240
571	163.35	163.113563

```
from sklearn.metrics import confusion_matrix, accuracy_score
regression.score(x1_test, y1_test)
```

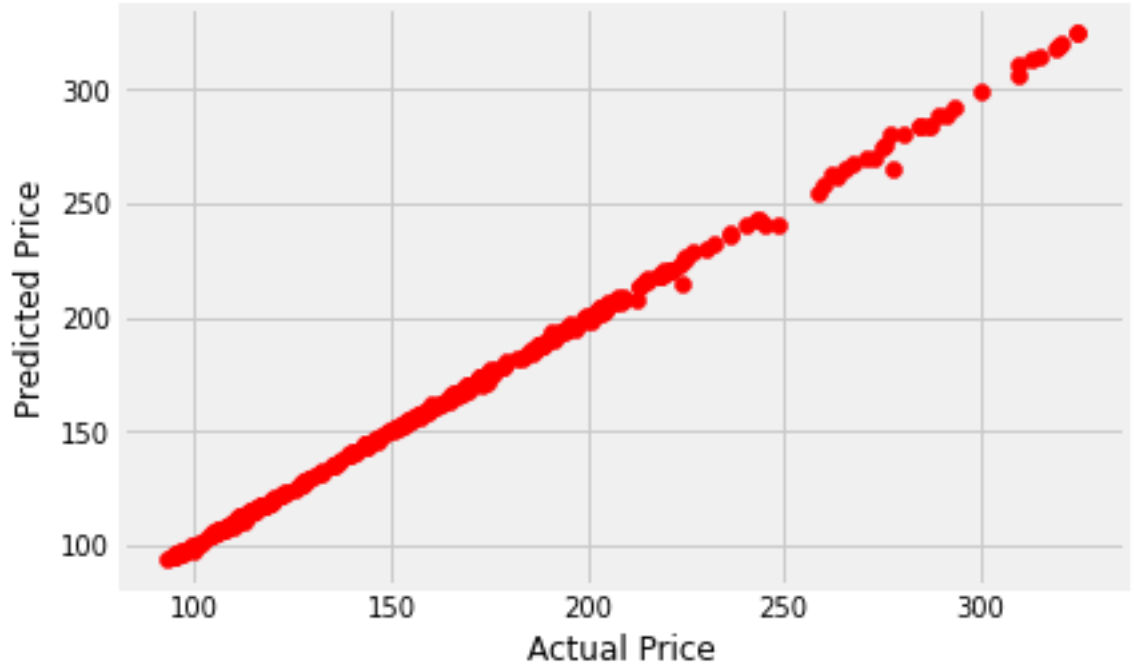
```
0.9993380579129287
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y1_test, predicted))
print('Mean Squared Error:', metrics.mean_squared_error(y1_test, predicted))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y1_test, p
redicted)))
```

```
Mean Absolute Error: 0.761370236045298
Mean Squared Error: 1.976697062918357
Root Mean Squared Error: 1.4059505904968
```

```
plt.scatter(dfr.Actual_Price, dfr.Predicted_Price,  color='red')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")

plt.show()
```



10.7 Results and Discussion:

The stock price forecasting problem has been solved using Linear Regression. The Linear Regression model is 99% accurate.