STOCK PRICE PREDICTION

DATA PREPROCESSING – PHASE 1

Submitted by:

Adhithya Sree Mohan	AM.EN. U4CSE20002
Akhilesh Rajesh	AM.EN. U4CSE20005
Tarun Raveesh	AM.EN. U4CSE20016
Golla Ajay Kumar	AM.EN. U4CSE20026
Karuturi Paavani	AM.EN. U4CSE20036

IMPORTING LIBRARIES

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import string
import csv
import re

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import scatter_matrix
from collections import Counter
from sklearn import metrics

warnings.filterwarnings("ignore")
```

✓ 0.1s

EXTRACTING DATA - Reading the CSV File:

```
Data = pd.read_csv("C:\\Users\\NFLX.csv")
print(Data)
```

✓ 0.4s

```
           Date        Open        High         Low       Close   Adj Close  \
0     2018-02-05  262.000000  267.899994  250.029999  254.259995  254.259995
1     2018-02-06  247.699997  266.700012  245.000000  265.720001  265.720001
2     2018-02-07  266.579987  272.450012  264.329987  264.559998  264.559998
3     2018-02-08  267.079987  267.619995  250.000000  250.100006  250.100006
4     2018-02-09  253.850006  255.800003  236.110001  249.470001  249.470001
...          ...         ...         ...         ...         ...         ...
1004  2022-01-31  401.970001  427.700012  398.200012  427.140015  427.140015
1005  2022-02-01  432.959991  458.480011  425.540009  457.130005  457.130005
1006  2022-02-02  448.250000  451.980011  426.480011  429.480011  429.480011
1007  2022-02-03  421.440002  429.260010  404.279999  405.600006  405.600006
1008  2022-02-04  407.309998  412.769989  396.640015  410.170013  410.170013

       Volume
0     11896100
1     12595800
2      8981500
3      9306700
4     16906900
...        ...
1004  20047500
1005  22542300
1006  14346000
1007   9905200
1008   7782400

[1009 rows x 7 columns]
```

DATA PREPROCESSING

```
# Finding size of dataset (i.e number of rows & columns of the dataset.)
print("Rows: ",Data.shape[0])
print("Columns: ",Data.shape[1])
```

✓ 0.7s

```
Rows:  1009
Columns:  7
```

```
Data.head() # head() function by default showcases first five rows
```

✓ 0.4s

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900

```
Data.shape
```

✓ 0.4s

```
(1009, 7)
```

```
Data.describe()
```

✓ 0.7s

	Open	High	Low	Close	Adj Close	Volume
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	1.009000e+03
mean	419.059673	425.320703	412.374044	419.000733	419.000733	7.570685e+06
std	108.537532	109.262960	107.555867	108.289999	108.289999	5.465535e+06
min	233.919998	250.649994	231.229996	233.880005	233.880005	1.144000e+06
25%	331.489990	336.299988	326.000000	331.619995	331.619995	4.091900e+06
50%	377.769989	383.010010	370.880005	378.670013	378.670013	5.934500e+06
75%	509.130005	515.630005	502.529999	509.079987	509.079987	9.322400e+06
max	692.349976	700.989990	686.090027	691.690002	691.690002	5.890430e+07

```
Data.columns
```

✓ 0.4s

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

Different Data Types In Datasets:

```
Data.dtypes
```

✓ 0.5s

```
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

```
Data.info()
```

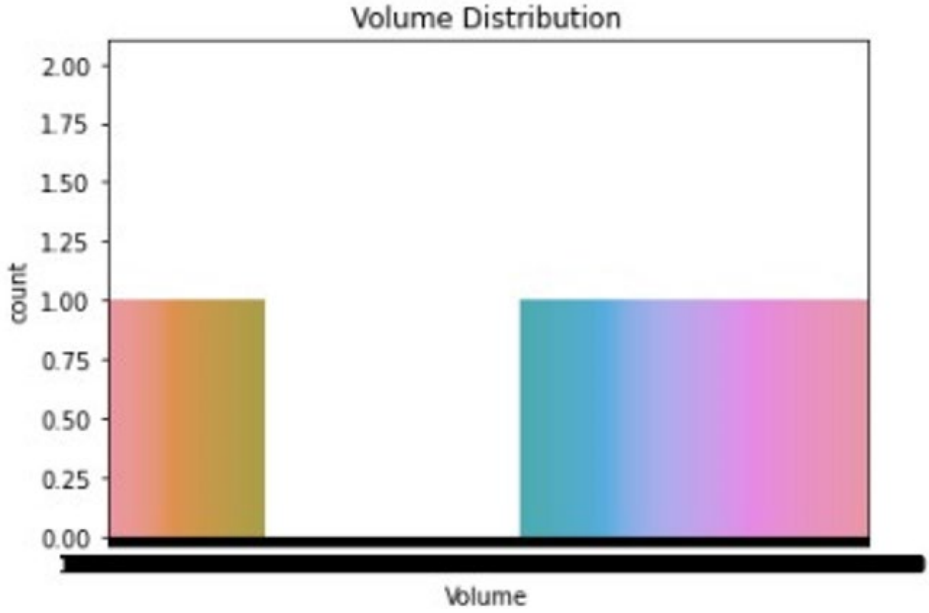
✓ 0.8s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              1009 non-null   object
 1   Open              1009 non-null   float64
 2   High              1009 non-null   float64
 3   Low               1009 non-null   float64
 4   Close             1009 non-null   float64
 5   Adj Close         1009 non-null   float64
 6   Volume            1009 non-null   float64
 7   bin_of_Adj Close  1009 non-null   category
dtypes: category(1), float64(6), object(1)
memory usage: 56.5+ KB
```

```
# Finding Number of samples under target variable
print(f"Number of samples under target value: \n{Data['Volume'].value_counts()}")
sns.countplot(Data.Volume).set_ylim(0, 2)
plt.show()
```

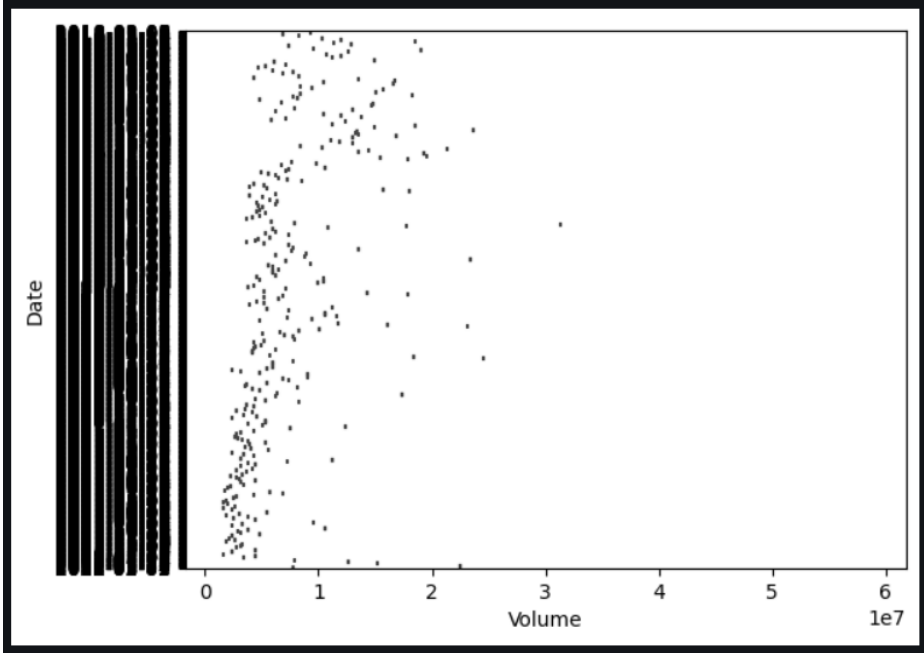
✓ 0.1s

```
Number of samples under target value:
6717700    2
5439200    2
3732200    2
6997900    2
4408200    1
          ..
5019000    1
5358200    1
5428500    1
5667200    1
7782400    1
Name: Volume, Length: 1005, dtype: int64
```




```
sns.boxplot(x = "Volume", y = "Date", data = Data)
plt.show()
```

✓ 9.6s

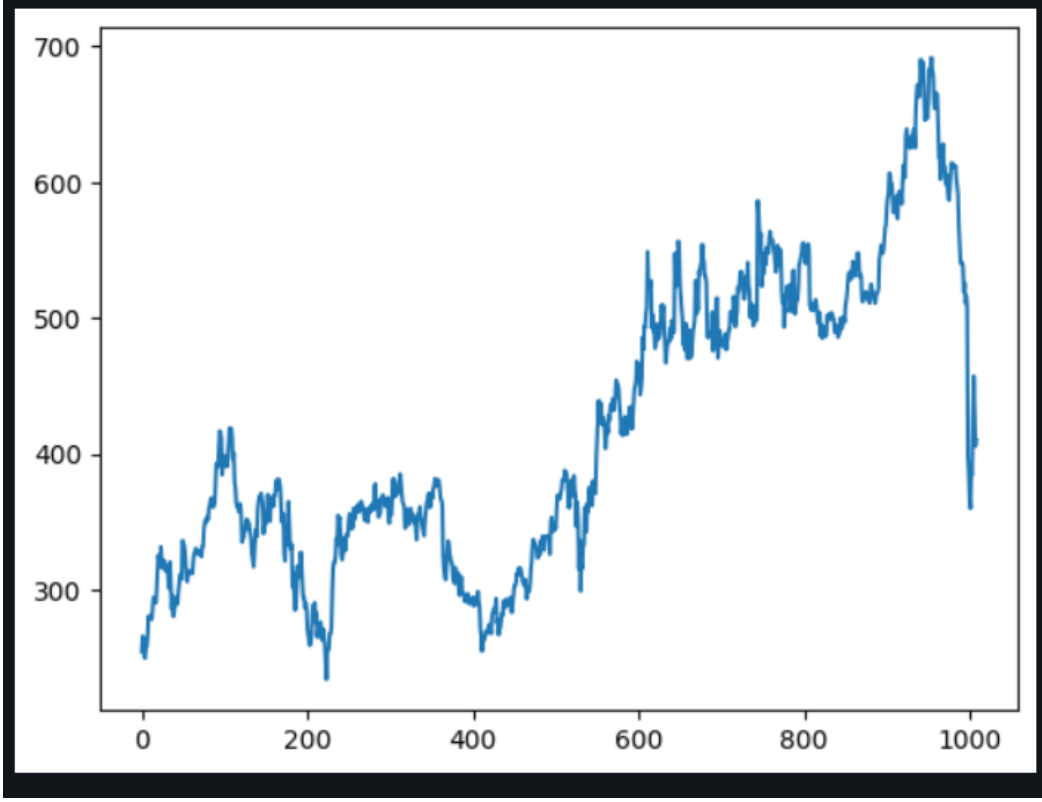


The Adjusted Close Value is the final output value that will be forecasted using the Machine Learning model

```
# Plot the True Adj Close Value
Data['Adj Close'].plot()
```

✓ 0.2s

```
<AxesSubplot: >
```



IMPUTING DATA VALUES

There were no missing values in the datasets. So, there was no replacement and missing values.

```
Data.isnull().values.any()  # Checking whether we have any missing values in dataset
```

✓ 0.3s

```
False
```

```
Data.isnull().sum()
```

✓ 0.4s

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

NORMALIZATION

```
norm = MinMaxScaler()
Data["Volume"] = norm.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Normalisation :")
Data.head()
```

✓ 0.4s

```
After Normalisation :
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	0.186150
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	0.198264
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	0.135690
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	0.141320
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	0.272902

Making data available for various machine learning model through normalization.

STANDARIZATION

```
Data.head()
```

✓ 0.4s

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	0.186150
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	0.198264
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	0.135690
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	0.141320
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	0.272902

```
Data['Volume'][:5]
```

✓ 0.4s

```
0    0.186150
1    0.198264
2    0.135690
3    0.141320
4    0.272902
Name: Volume, dtype: float64
```

```
scalar = StandardScaler(copy=True, with_mean=True, with_std=True)
Data["Volume"] = scalar.fit_transform(Data["Volume"].values.reshape(-1,1))
print ("After Standardisation : \n")
Data.head()
```

✓ 0.6s

```
After Standardisation :
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	0.791791
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	0.919875
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	0.258257
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	0.317787
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	1.709045

DISCRETIZATION

```
Data['Adj Close'].unique()
```

✓ 0.7s

```
Output exceeds the size limit. Open the full output data in a text editor
array([254.259995, 265.720001, 264.559998, 250.100006, 249.470001,
       257.950012, 258.269989, 266.      , 280.269989, 278.519989,
       278.549988, 281.040009, 278.140015, 285.929993, 294.160004,
       290.609985, 291.380005, 290.390015, 301.049988, 315.      ,
       325.220001, 321.160004, 317.      , 331.440002, 321.299988,
       315.880005, 321.549988, 321.089996, 318.450012, 313.480011,
       317.5     , 316.480011, 306.700012, 300.940002, 320.350006,
       300.690002, 285.769989, 295.350006, 280.290009, 283.670013,
       288.940002, 293.970001, 288.850006, 289.929993, 298.070007,
       303.670013, 309.25    , 311.649994, 307.779999, 336.059998,
       334.519989, 332.700012, 327.769989, 318.690002, 307.019989,
       305.76001 , 313.980011, 311.76001 , 312.459991, 313.299988,
       313.359985, 311.690002, 320.089996, 326.26001 , 326.890015,
       330.299988, 329.600006, 326.459991, 328.529999, 326.130005,
       328.190002, 324.179993, 331.820007, 331.619995, 344.720001,
       349.290009, 351.290009, 349.730011, 353.540009, 351.600006,
       359.929993, 361.809998, 365.799988, 367.450012, 361.399994,
       360.570007, 361.450012, 363.829987, 379.929993, 392.869995,
       391.980011, 390.399994, 404.980011, 416.76001 , 415.440002,
       411.089996, 384.480011, 399.390015, 390.390015, 395.420013,
       391.429993, 398.179993, 390.519989, 398.390015, 408.25    ,
       418.970001, 415.630005, 418.649994, 413.5     , 395.799988,
       400.480011, 379.480011, 375.130005, 364.230011, 361.049988,
       362.660004, 357.320007, 362.869995, 363.089996, 355.209991,
       334.959991, 337.450012, 338.380005, 344.5     , 343.089996,
```

```
...
       591.150024, 567.52002 , 553.289978, 541.059998, 539.849976,
       540.840027, 537.219971, 519.200012, 525.690002, 510.799988,
       515.859985, 508.25    , 397.5     , 387.149994, 366.420013,
       359.700012, 386.700012, 384.359985, 427.140015, 457.130005,
       429.480011, 405.600006, 410.170013])
```

```
print(Data['Adj Close'].max())
print(Data['Adj Close'].min())
```

✓ 0.4s

```
691.690002
233.880005
```

```
Data['bin_of_Adj Close'] = pd.cut(Data['Adj Close'], [200,300,400,500,600,700],
                                  labels = ['200-300', '300-400', '400-500', '500-600', '600-700'])
```

✓ 0.6s

```
Data.groupby([Data["bin_of_Adj Close"]]).count()
```

✓ 0.4s

bin_of_Adj Close	Date	Open	High	Low	Close	Adj Close	Volume
200-300	136	136	136	136	136	136	136
300-400	410	410	410	410	410	410	410
400-500	169	169	169	169	169	169	169
500-600	231	231	231	231	231	231	231
600-700	63	63	63	63	63	63	63

```
for column in Data.columns:
    print("------  " + column + "  ------")
    print(Data[column].value_counts())
```

✓ 0.4s

```
Output exceeds the size limit. Open the full output data in a text editor
------  Date  ------
2018-02-05    1
2020-10-14    1
2020-09-25    1
2020-09-28    1
2020-09-29    1
             ..
2019-06-14    1
2019-06-17    1
2019-06-18    1
2019-06-19    1
2022-02-04    1
Name: Date, Length: 1009, dtype: int64
```

```
------  Open  ------
365.000000    4
359.000000    3
355.000000    3
295.000000    3
425.000000    2
              ..
378.290009    1
378.190002    1
379.059998    1
382.769989    1
407.309998    1
...
400-500     169
200-300     136
600-700      63
Name: bin_of_Adj Close, dtype: int64
```

Making the values group wise and making continuous values as discrete.

TEXT PREPROCESSING

There were no strings in the datasets, so no modifications required.

IMAGE PREPROCESSING

There were no images given in data sets so there will be no use of image pre-processing and conversion of color image to grey scale image.

DATA SUMMARIZATION

```
print(Data.shape)
```

✓ 0.3s

```
(1009, 8)
```

```
Data.describe()
```

✓ 0.4s

	Open	High	Low	Close	Adj Close	Volume
count	1009.000000	1009.000000	1009.000000	1009.000000	1009.000000	1.009000e+03
mean	419.059673	425.320703	412.374044	419.000733	419.000733	-2.816820e-17
std	108.537532	109.262960	107.555867	108.289999	108.289999	1.000496e+00
min	233.919998	250.649994	231.229996	233.880005	233.880005	-1.176440e+00
25%	331.489990	336.299988	326.000000	331.619995	331.619995	-6.368105e-01
50%	377.769989	383.010010	370.880005	378.670013	378.670013	-2.995126e-01
75%	509.130005	515.630005	502.529999	509.079987	509.079987	3.206609e-01
max	692.349976	700.989990	686.090027	691.690002	691.690002	9.396897e+00

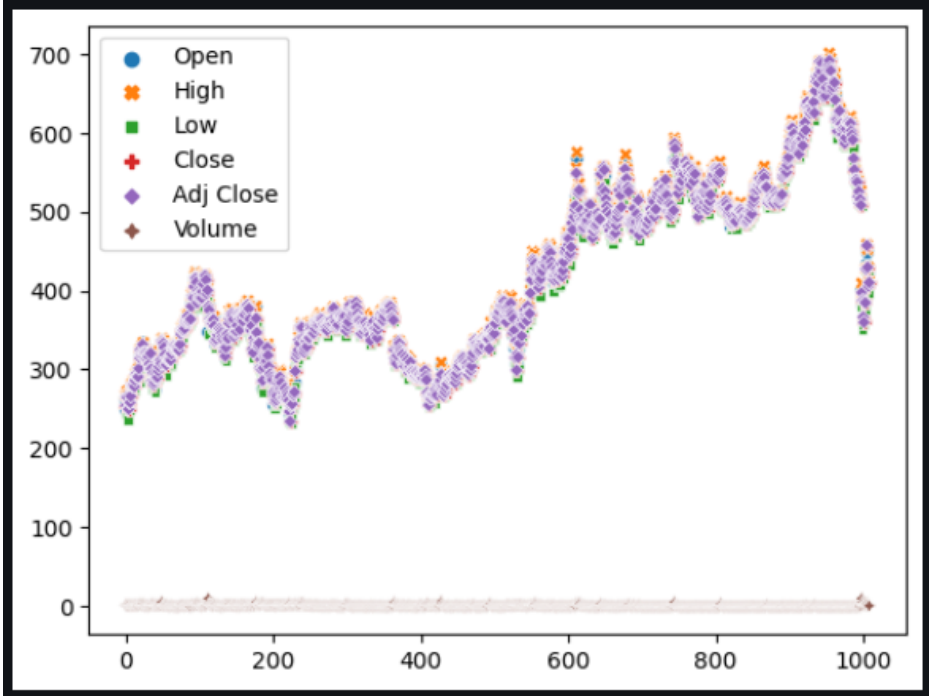
DATA VISUALIZATION

Scatter Plot

```
sns.scatterplot(Data)
```

0.4s

```
<AxesSubplot: >
```

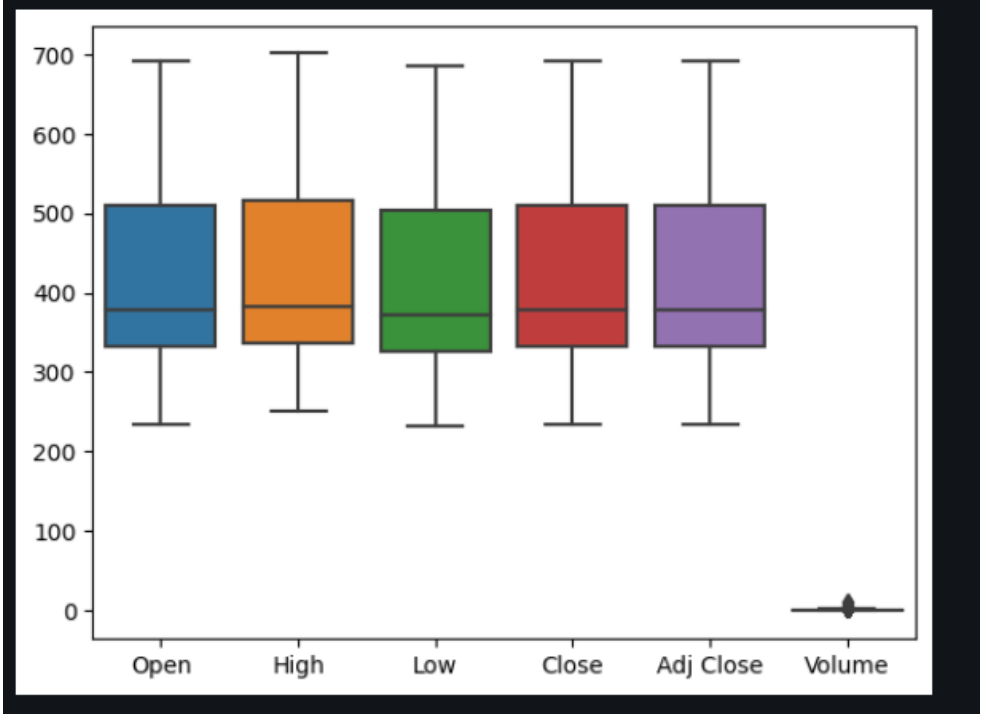


Box Plot

```
sns.boxplot(Data)
```

0.1s

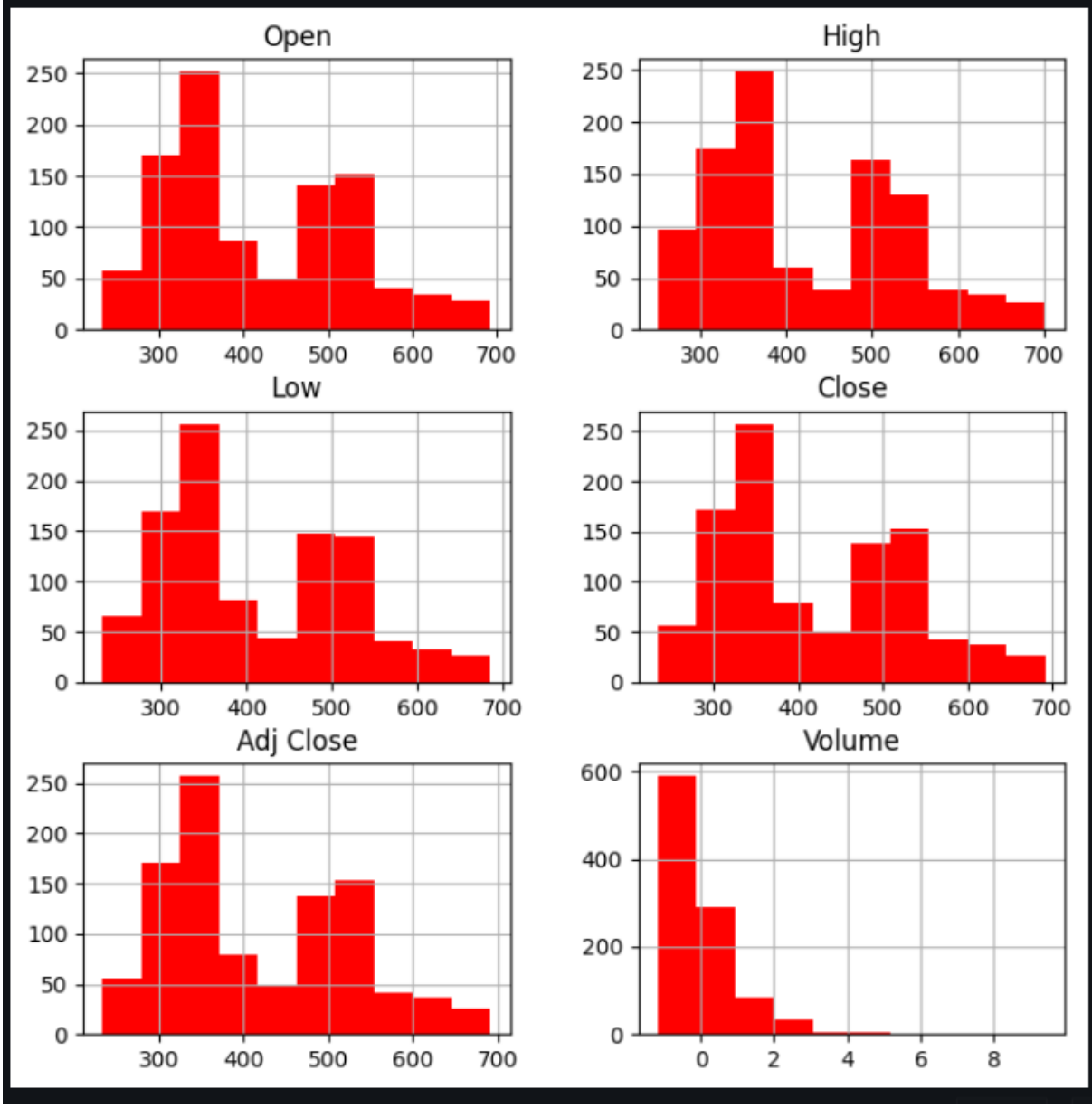
```
<AxesSubplot: >
```



Histogram

```
Data.hist(color = "red", figsize = (8,8))
plt.show()
```

✓ 1.1s



These are the various visualizations of data, and we normalized data. So, we can use data to apply for various models.

*****