Project Title: Stock Price Prediction for Netflix using LSTM

Problem Statement:

The stock market is a dynamic and complex system where investors aim to make informed decisions to maximize their returns. Predicting the stock price of a company like Netflix is particularly challenging due to its exposure to various market dynamics, competition, and global events. This project aims to develop a robust predictive model using Long Short-Term Memory (LSTM) neural networks to forecast the future stock prices of Netflix accurately.

Description:

Netflix is a leading streaming service provider, and its stock price is influenced by a myriad of factors, including subscriber growth, content releases, market sentiment, and economic conditions. This project will utilize historical stock price data, along with relevant features such as trading volume, macroeconomic indicators, and news sentiment analysis, to train an LSTM-based deep learning model.

Conclusion:

Summarize the project's findings and key results. Discuss the model's accuracy and its ability to predict Netflix stock prices. Highlight any challenges encountered during the project and potential areas for improvement. Emphasize the importance of using LSTM neural networks for time series forecasting tasks like stock price prediction.

Outline :

1 .Libraries and settings

2 .Analyze data

3 .Manipulate data

4 .Model and validate data

5 .Predictions

```
# Disable all warnings
import warnings
warnings.filterwarnings('ignore')
import numpy as np   # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. 
pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
import tensorflow
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
from keras.layers import SimpleRNN
from keras.layers import LSTM
from keras.layers import Dropout

# import all stock prices
data= pd.read_csv("/content/NFLX.csv", index_col = 0)
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1009 entries, 2018-02-05 to 2022-02-04
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       1009 non-null   float64
 1   High       1009 non-null   float64
 2   Low        1009 non-null   float64
 3   Close      1009 non-null   float64
 4   Adj Close  1009 non-null   float64
 5   Volume     1009 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 55.2+ KB

                  Open        High         Low       Close   Adj Close
\
Date

2018-02-05  262.000000  267.899994  250.029999  254.259995  254.259995

2018-02-06  247.699997  266.700012  245.000000  265.720001  265.720001

2018-02-07  266.579987  272.450012  264.329987  264.559998  264.559998

2018-02-08  267.079987  267.619995  250.000000  250.100006  250.100006

2018-02-09  253.850006  255.800003  236.110001  249.470001  249.470001


              Volume
Date
2018-02-05  11896100
2018-02-06  12595800
2018-02-07   8981500
2018-02-08   9306700
2018-02-09  16906900
```

```
data.tail()
```

```
                  Open        High         Low       Close   Adj Close
\
```

```
Date

2022-01-31  401.970001  427.700012  398.200012  427.140015  427.140015

2022-02-01  432.959991  458.480011  425.540009  457.130005  457.130005

2022-02-02  448.250000  451.980011  426.480011  429.480011  429.480011

2022-02-03  421.440002  429.260010  404.279999  405.600006  405.600006

2022-02-04  407.309998  412.769989  396.640015  410.170013  410.170013


              Volume
Date
2022-01-31  20047500
2022-02-01  22542300
2022-02-02  14346000
2022-02-03   9905200
2022-02-04   7782400
```

```
data.describe()
```

```
              Open         High          Low        Close    Adj Close
\
count  1009.000000  1009.000000  1009.000000  1009.000000  1009.000000

mean    419.059673   425.320703   412.374044   419.000733   419.000733

std     108.537532   109.262960   107.555867   108.289999   108.289999

min     233.919998   250.649994   231.229996   233.880005   233.880005

25%     331.489990   336.299988   326.000000   331.619995   331.619995

50%     377.769989   383.010010   370.880005   378.670013   378.670013

75%     509.130005   515.630005   502.529999   509.079987   509.079987

max     692.349976   700.989990   686.090027   691.690002   691.690002


             Volume
count  1.009000e+03
mean   7.570685e+06
std    5.465535e+06
min    1.144000e+06
25%    4.091900e+06
50%    5.934500e+06
75%    9.322400e+06
max    5.890430e+07
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1009 entries, 2018-02-05 to 2022-02-04
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Open       1009 non-null   float64
 1   High       1009 non-null   float64
 2   Low        1009 non-null   float64
 3   Close      1009 non-null   float64
 4   Adj Close  1009 non-null   float64
 5   Volume     1009 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 55.2+ KB
```

```
# Let's check the data, to see if there is duplicate data or not
data.duplicated()
```

```
Date
2018-02-05    False
2018-02-06    False
2018-02-07    False
2018-02-08    False
2018-02-09    False
              ...
2022-01-31    False
2022-02-01    False
2022-02-02    False
2022-02-03    False
2022-02-04    False
Length: 1009, dtype: bool
```

```
# Checking for missing values
data.isnull().sum()
```

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

Data Visualization

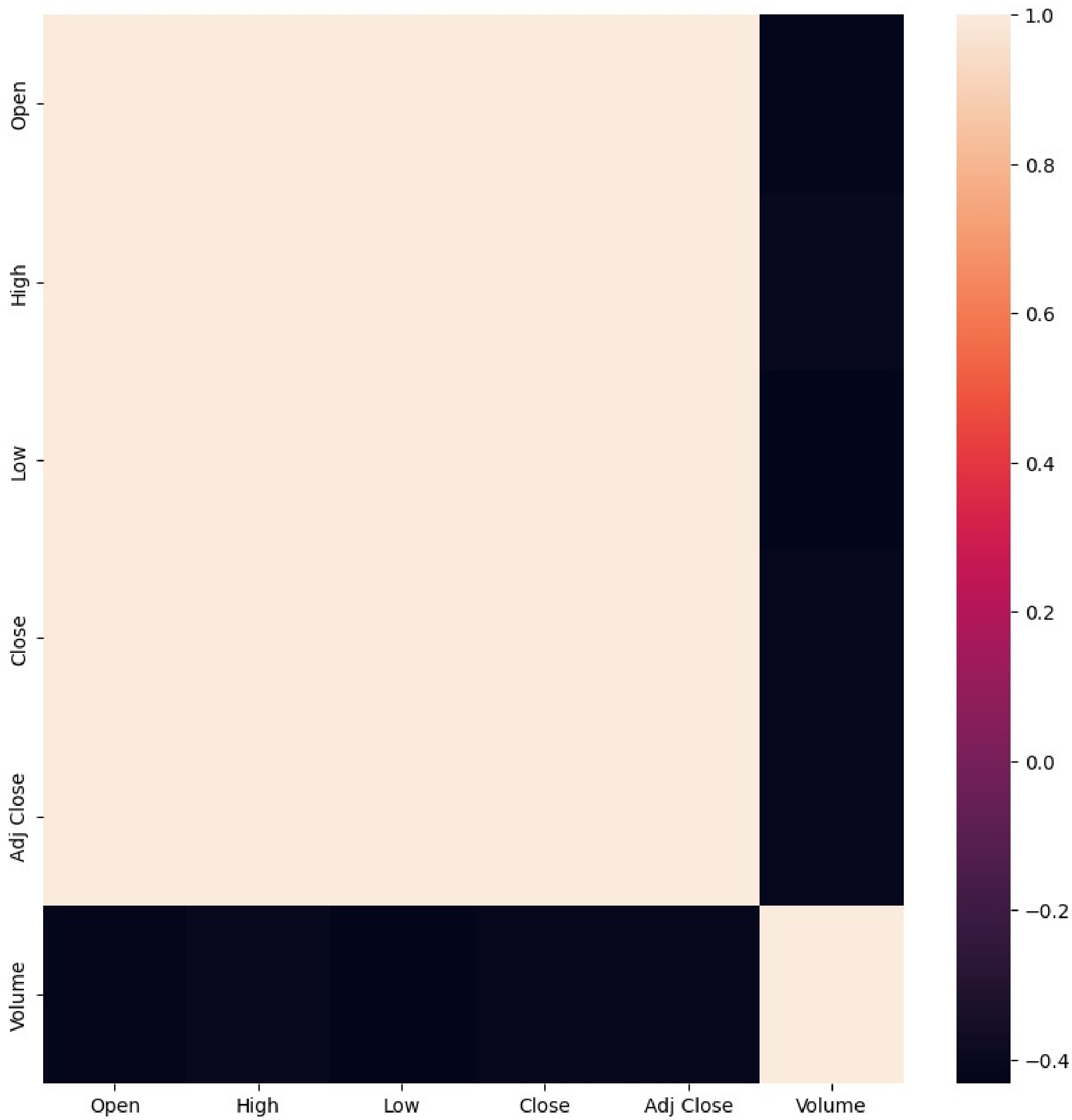
```
df = data.copy()
df
```

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

              Volume
Date
2018-02-05  11896100
2018-02-06  12595800
2018-02-07   8981500
2018-02-08   9306700
2018-02-09  16906900
...              ...
2022-01-31  20047500
2022-02-01  22542300
2022-02-02  14346000
2022-02-03   9905200
2022-02-04   7782400

[1009 rows x 6 columns]

# Matrix form for correlation data
drrr= data.corr()
drrr

              Open      High       Low     Close  Adj Close    Volume
Open      1.000000  0.998605  0.998508  0.996812   0.996812 -0.415838
High      0.998605  1.000000  0.998203  0.998551   0.998551 -0.400699
Low       0.998508  0.998203  1.000000  0.998544   0.998544 -0.432116
```

```
Close      0.996812  0.998551  0.998544  1.000000   1.000000 -0.413362
Adj Close  0.996812  0.998551  0.998544  1.000000   1.000000 -0.413362
Volume    -0.415838 -0.400699 -0.432116 -0.413362  -0.413362  1.000000
```

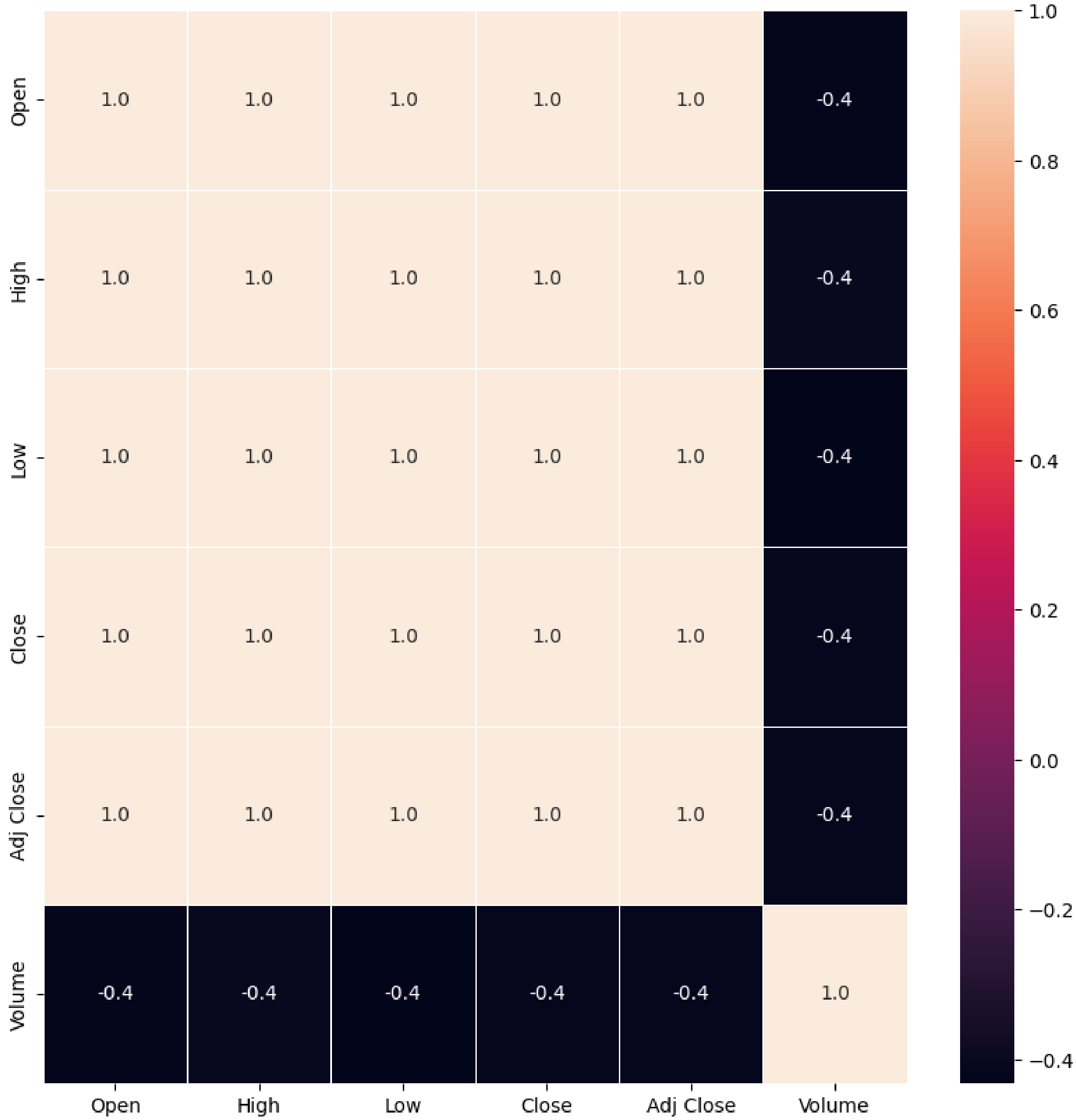
```
# We here looking at the data Visualization by heatmap.
plt.figure(figsize=(10,10))
sns.heatmap(drrr)
```

```
<Axes: >
```

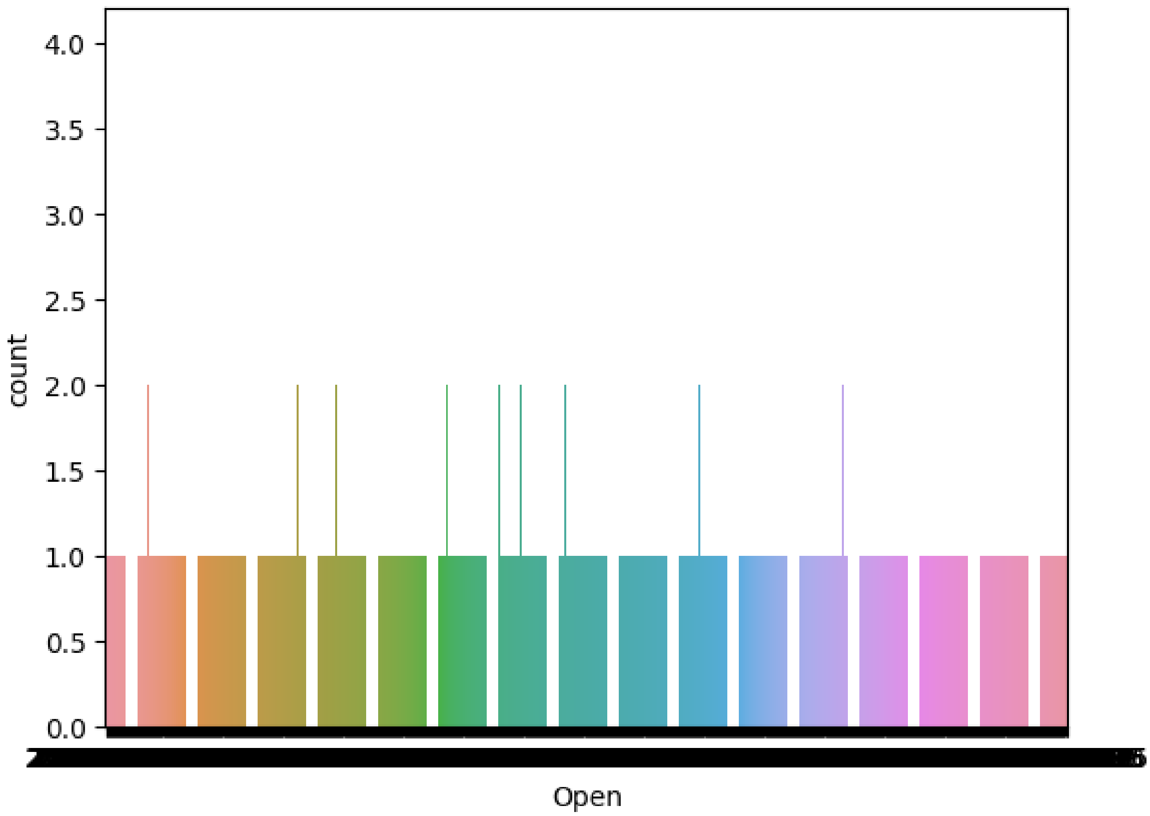


```python
f,ax = plt.subplots(figsize=(10, 10))
sns.heatmap(drrr, annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

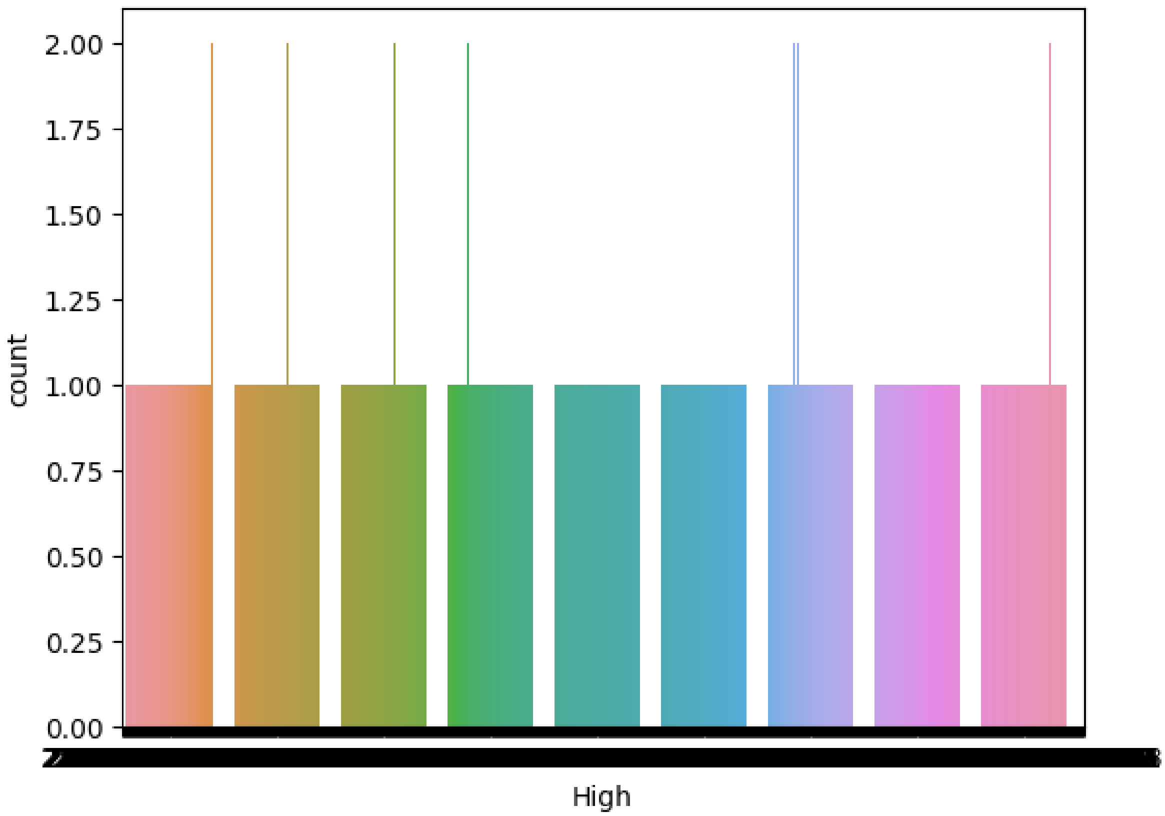
```
<Axes: >
```



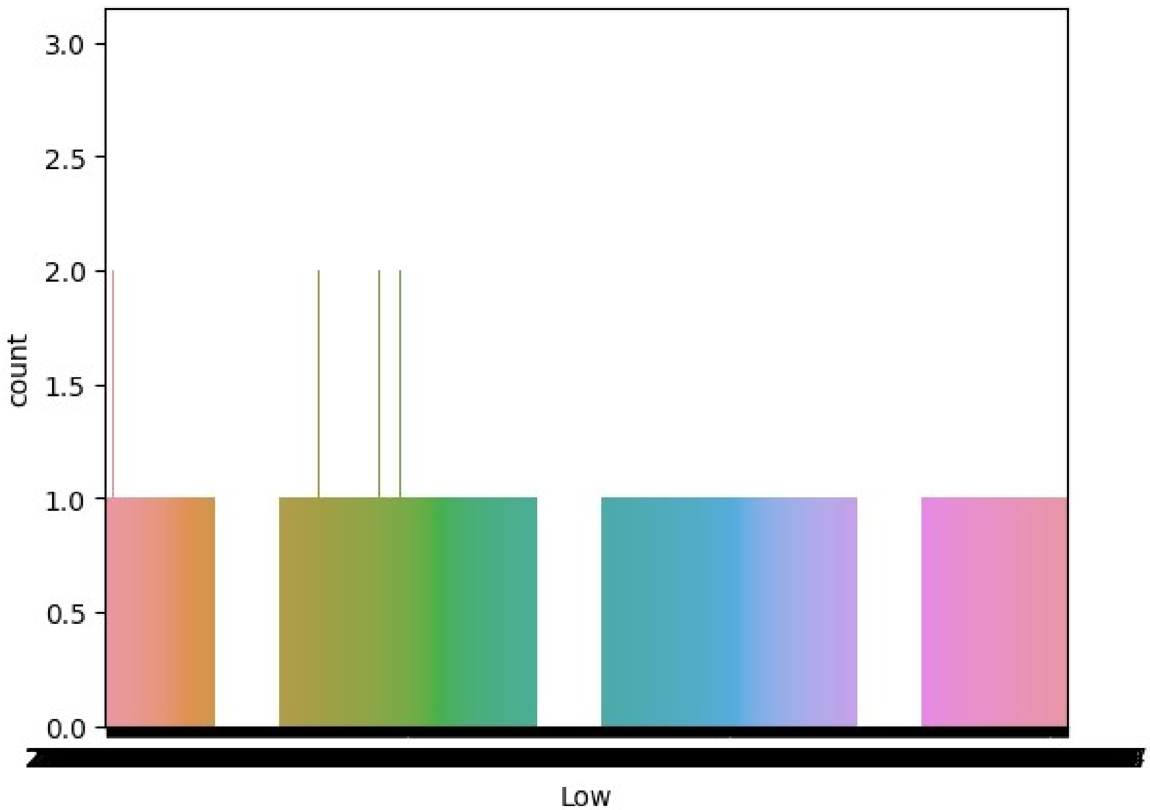

```python
# Original data.
for i in df.loc[:, ~df.columns.isin(["Date", "Volume"])]:
    f = sns.countplot(x=df[i]);
    plt.figure(figsize=(10,10))
    plt.show()
```



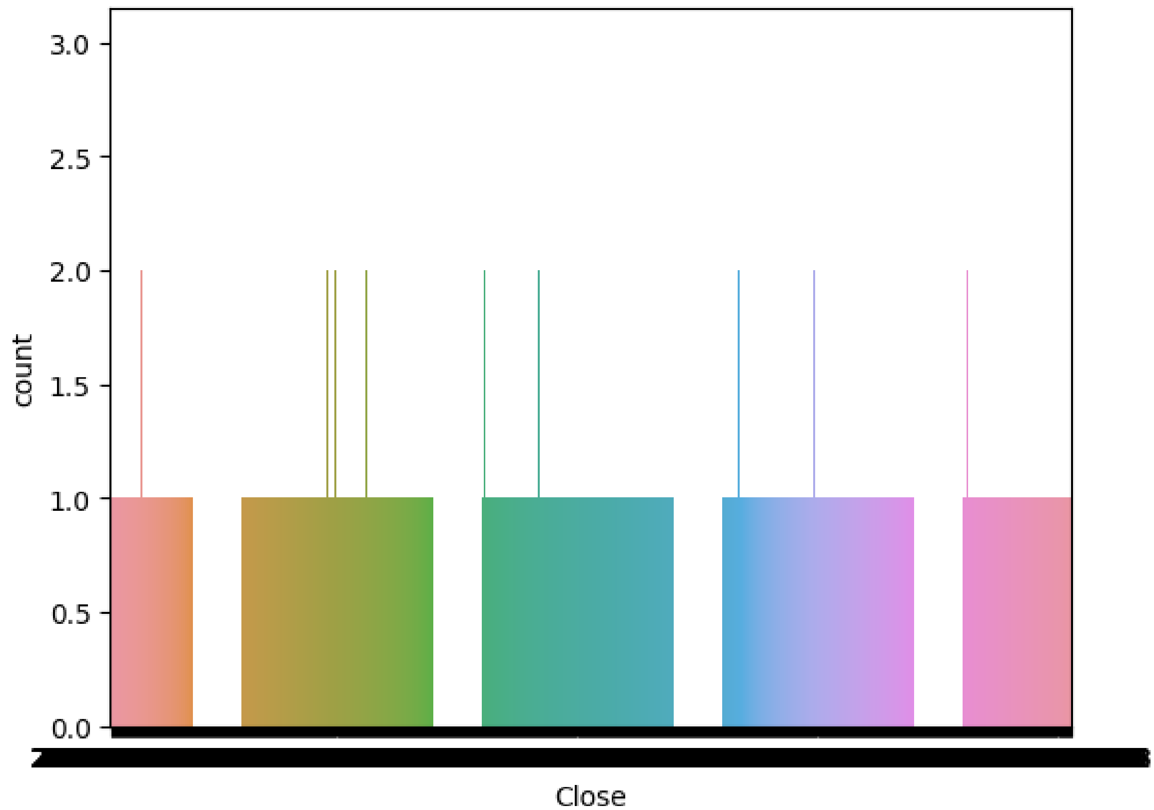

```
<Figure size 1000x1000 with 0 Axes>
```



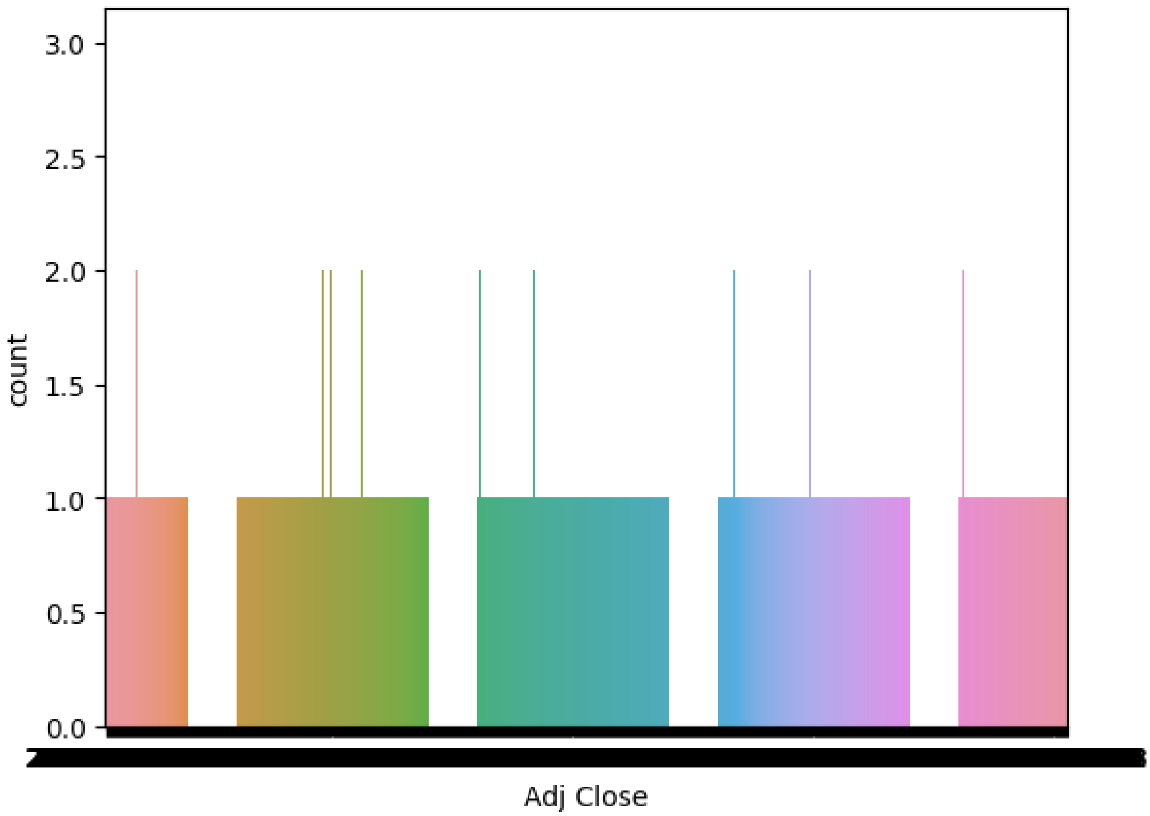

```
<Figure size 1000x1000 with 0 Axes>
```



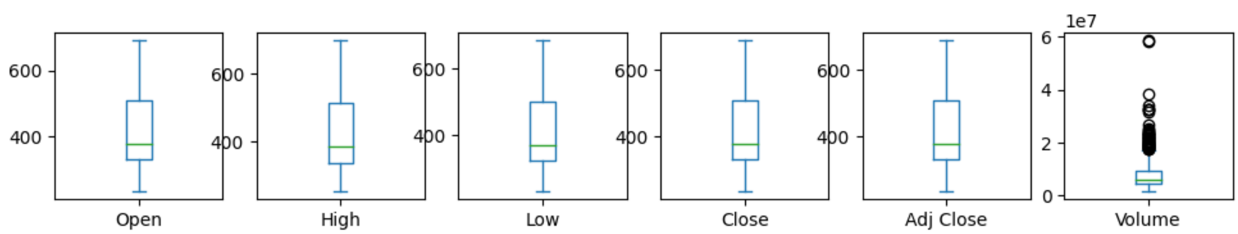

```
<Figure size 1000x1000 with 0 Axes>
```




```
<Figure size 1000x1000 with 0 Axes>
```




```
<Figure size 1000x1000 with 0 Axes>

df.plot(kind='box',subplots=True,layout=(10,10),figsize=(20,20))
plt.show()
```



Splitting

```
train_data, test_data = train_test_split(data, test_size=0.2,
random_state=42)
train_data.shape, test_data.shape

((807, 6), (202, 6))
```

Preprocessing

```
scaler = MinMaxScaler()
scalerr = MinMaxScaler()
train_data_scaled = scaler.fit_transform(train_data)
train_data_scaled
```

```
array([[0.17860959, 0.14960206, 0.16969522, 0.17439697, 0.17439697,
        0.14992077],
       [0.28898633, 0.26224888, 0.27997145, 0.27364546, 0.27364546,
        0.10854764],
       [0.30501932, 0.28738923, 0.30877797, 0.30631203, 0.30631203,
        0.06049766],
       ...,
       [0.67691906, 0.66287127, 0.6698401 , 0.66183209, 0.66183209,
        0.02491101],
       [0.09626332, 0.07914141, 0.10282939, 0.10512013, 0.10512013,
        0.08610638],
       [0.33054124, 0.33309287, 0.3316983 , 0.35996755, 0.35996755,
        0.1189803 ]])
```

```
test_data_scaled = scaler.transform(test_data)
test_data_scaled
```

```
array([[0.57747095, 0.5857591 , 0.59624089, 0.60416715, 0.60416715,
        0.07499163],
       [0.59230423, 0.58091143, 0.56955252, 0.57150058, 0.57150058,
        0.08023833],
       [0.5964488 , 0.59491335, 0.59831441, 0.58493084, 0.58493084,
        0.05016219],
       ...,
       [0.11825144, 0.10166625, 0.09696552, 0.09081345, 0.09081345,
        0.19982609],
       [0.28200602, 0.27101986, 0.29424097, 0.29827134, 0.29827134,
        0.04948878],
       [0.2564841 , 0.23776243, 0.25747474, 0.25381769, 0.25381769,
        0.14632288]])
```

```
#Creating a data structure with 50 timesteps and 1 output, timestep is
our memory size
#In this function we are creating our train data with 50x stock price
and next one is 1 scrolled data.
#for example X_tain[0] will be our data's 0 to 49. values
#X_train[1] will be our data's 1 to 50. values
#this 50 is our memory size, it will remember this way what we had
before.
X_train=[]
y_train=[]
timesteps=10
for i in range(timesteps,len(train_data_scaled)):
    X_train.append(train_data_scaled[i-timesteps:i,0])
    y_train.append(train_data_scaled[i,0])
```

```
X_train,y_train=np.array(X_train),np.array(y_train)
real_stock_price=test_data.loc[:,["Open"]].values
```

Model

Recurrent Neural Network (RNN)

```
regressor=Sequential()
#Adding the first RNN Layer and some Dropout Regularisation
regressor.add(SimpleRNN(units=10,activation="tanh",return_sequences=True,input_shape=(X_train.shape[1],1)))
regressor.add(Dropout(0,2))

#Adding the second RNN Layer and some Dropout Regularisation
regressor.add(SimpleRNN(units=10,activation="tanh",return_sequences=True))
regressor.add(Dropout(0,2))

#Adding the third RNN Layer and some Dropout Regularisation
regressor.add(SimpleRNN(units=10,activation="tanh",return_sequences=True))
regressor.add(Dropout(0,2))

#Adding the third RNN Layer and some Dropout Regularisation
regressor.add(SimpleRNN(units=10,activation="tanh",return_sequences=True))
regressor.add(Dropout(0,2))

#Adding the fourth RNN Layer and some Dropout Regularisation
regressor.add(SimpleRNN(units=10))
regressor.add(Dropout(0,2))

#Adding the output Layer
regressor.add(Dense(units=1))

regressor.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_15 (SimpleRNN)   (None, 50, 10)            120

 dropout_21 (Dropout)        (None, 50, 10)            0

 simple_rnn_16 (SimpleRNN)   (None, 50, 10)            210

 dropout_22 (Dropout)        (None, 50, 10)            0
```

```
 simple_rnn_17 (SimpleRNN)    (None, 50, 10)            210

 dropout_23 (Dropout)         (None, 50, 10)            0

 simple_rnn_18 (SimpleRNN)    (None, 50, 10)            210

 dropout_24 (Dropout)         (None, 50, 10)            0

 simple_rnn_19 (SimpleRNN)    (None, 10)                210

 dropout_25 (Dropout)         (None, 10)                0

 dense_4 (Dense)              (None, 1)                 11

=================================================================
Total params: 971 (3.79 KB)
Trainable params: 971 (3.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#Compling the RNN
regressor.compile(optimizer="adam",loss="mean_squared_error")
#Fitting the RNN to the Training set
regressor.fit(X_train,y_train,epochs=10,batch_size=32)
```

```
Epoch 1/10
24/24 [==============================] - 5s 45ms/step - loss: 0.1493
Epoch 2/10
24/24 [==============================] - 2s 67ms/step - loss: 0.0667
Epoch 3/10
24/24 [==============================] - 2s 79ms/step - loss: 0.0613
Epoch 4/10
24/24 [==============================] - 1s 45ms/step - loss: 0.0593
Epoch 5/10
24/24 [==============================] - 1s 44ms/step - loss: 0.0589
Epoch 6/10
24/24 [==============================] - 1s 45ms/step - loss: 0.0580
Epoch 7/10
24/24 [==============================] - 1s 45ms/step - loss: 0.0591
Epoch 8/10
24/24 [==============================] - 1s 44ms/step - loss: 0.0575
Epoch 9/10
24/24 [==============================] - 1s 45ms/step - loss: 0.0569
Epoch 10/10
24/24 [==============================] - 1s 45ms/step - loss: 0.0567

<keras.src.callbacks.History at 0x7f37d12f4eb0>
```

Long Short Term Memory (LSTM)

```
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units = 1))

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 50, 50)            10400

 dropout_26 (Dropout)        (None, 50, 50)            0

 lstm_7 (LSTM)               (None, 50, 50)            20200

 dropout_27 (Dropout)        (None, 50, 50)            0

 lstm_8 (LSTM)               (None, 50, 50)            20200

 dropout_28 (Dropout)        (None, 50, 50)            0

 lstm_9 (LSTM)               (None, 50, 50)            20200

 dropout_29 (Dropout)        (None, 50, 50)            0

 lstm_10 (LSTM)              (None, 50, 50)            20200

 dropout_30 (Dropout)        (None, 50, 50)            0

 lstm_11 (LSTM)              (None, 50)                20200

 dropout_31 (Dropout)        (None, 50)                0

 dense_5 (Dense)             (None, 1)                 51

=================================================================
```

```
Total params: 111451 (435.36 KB)
Trainable params: 111451 (435.36 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

model.compile(loss = 'mean_squared_error', optimizer = 'Adam')
model.fit(X_train, y_train, epochs = 10, batch_size = 32)

Epoch 1/10
24/24 [==============================] - 20s 149ms/step - loss: 0.0890
Epoch 2/10
24/24 [==============================] - 4s 187ms/step - loss: 0.0603
Epoch 3/10
24/24 [==============================] - 4s 160ms/step - loss: 0.0607
Epoch 4/10
24/24 [==============================] - 4s 152ms/step - loss: 0.0601
Epoch 5/10
24/24 [==============================] - 4s 156ms/step - loss: 0.0598
Epoch 6/10
24/24 [==============================] - 5s 188ms/step - loss: 0.0588
Epoch 7/10
24/24 [==============================] - 4s 148ms/step - loss: 0.0593
Epoch 8/10
24/24 [==============================] - 4s 151ms/step - loss: 0.0587
Epoch 9/10
24/24 [==============================] - 5s 202ms/step - loss: 0.0590
Epoch 10/10
24/24 [==============================] - 4s 151ms/step - loss: 0.0587

<keras.src.callbacks.History at 0x7f37cf5c2980>

dataset_total=pd.concat((train_data["Open"],test_data["Open"]),
axis=0)
inputs= dataset_total[len(dataset_total)-len(train_data)-
timesteps:].values.reshape(-1,1)
inputs=scalerr.fit_transform(inputs)

#prediction
X_test=[]
for i in range(timesteps,timesteps+len(test_data)):
    X_test.append(inputs[i-timesteps:i,0])
X_test=np.array(X_test)
X_test=np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
predicted_stock_price=model.predict(X_test)
predicted_stock_price=scalerr.inverse_transform(predicted_stock_price)
#we had scaled between 0-1 data, inversing it

7/7 [==============================] - 4s 40ms/step

#visualising
plt.plot(real_stock_price,color="red",label="Real Netflix Stock
```

```
Price")
plt.plot(predicted_stock_price,color="blue",label="Predicted Netflix
Stock Price")
plt.title("Netflix Stock Price Prediction")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```