```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
# Load Dataset
pathway="/content/YesBank_StockPrices (2).csv"
df = pd.read_csv(pathway)
```

```
df
```

	Date	Open	High	Low	Close
0	Jul-05	13.00	14.00	11.25	12.46
1	Aug-05	12.58	14.88	12.55	13.42
2	Sep-05	13.48	14.87	12.27	13.30
3	Oct-05	13.20	14.47	12.40	12.99
4	Nov-05	13.35	13.88	12.88	13.41
...	...	...	...	...	...
180	Jul-20	25.60	28.30	11.10	11.95
181	Aug-20	12.00	17.16	11.85	14.37
182	Sep-20	14.30	15.34	12.75	13.15
183	Oct-20	13.30	14.01	12.11	12.42
184	Nov-20	12.41	14.90	12.21	14.67

185 rows × 5 columns

```
# Dataset First Look
df.head() # Changed 'data' to 'df' to match the DataFrame created in the previous cell
```

	Date	Open	High	Low	Close
0	Jul-05	13.00	14.00	11.25	12.46
1	Aug-05	12.58	14.88	12.55	13.42
2	Sep-05	13.48	14.87	12.27	13.30
3	Oct-05	13.20	14.47	12.40	12.99
4	Nov-05	13.35	13.88	12.88	13.41

```
# Dataset Rows & Columns count
df.shape # Changed 'data' to 'df' to refer to the DataFrame
```

```
(185, 5)
```

```
# Dataset Info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185 entries, 0 to 184
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    185 non-null    object 
 1   Open    185 non-null    float64
 2   High    185 non-null    float64
 3   Low     185 non-null    float64
 4   Close   185 non-null    float64
```

```
dtypes: float64(4), object(1)
memory usage: 7.4+ KB
```

```
# Dataset Duplicate Value Count
len(df[df.duplicated()]) # Changed 'data' to 'df'
```

```
0
```

```
# Missing Values/Null Values Count
df.isnull().sum()
```

	0
Date	0
Open	0
High	0
Low	0
Close	0

dtype: int64

```
# Dataset Columns
df.columns
```

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

```
# Dataset Describe
df.describe(include='all')
```

	Date	Open	High	Low	Close
count	185	185.000000	185.000000	185.000000	185.000000
unique	185	NaN	NaN	NaN	NaN
top	Jul-05	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN
mean	NaN	105.541405	116.104324	94.947838	105.204703
std	NaN	98.879850	106.333497	91.219415	98.583153
min	NaN	10.000000	11.240000	5.550000	9.980000
25%	NaN	33.800000	36.140000	28.510000	33.450000
50%	NaN	62.980000	72.550000	58.000000	62.540000
75%	NaN	153.000000	169.190000	138.350000	153.300000
max	NaN	369.950000	404.000000	345.500000	367.900000

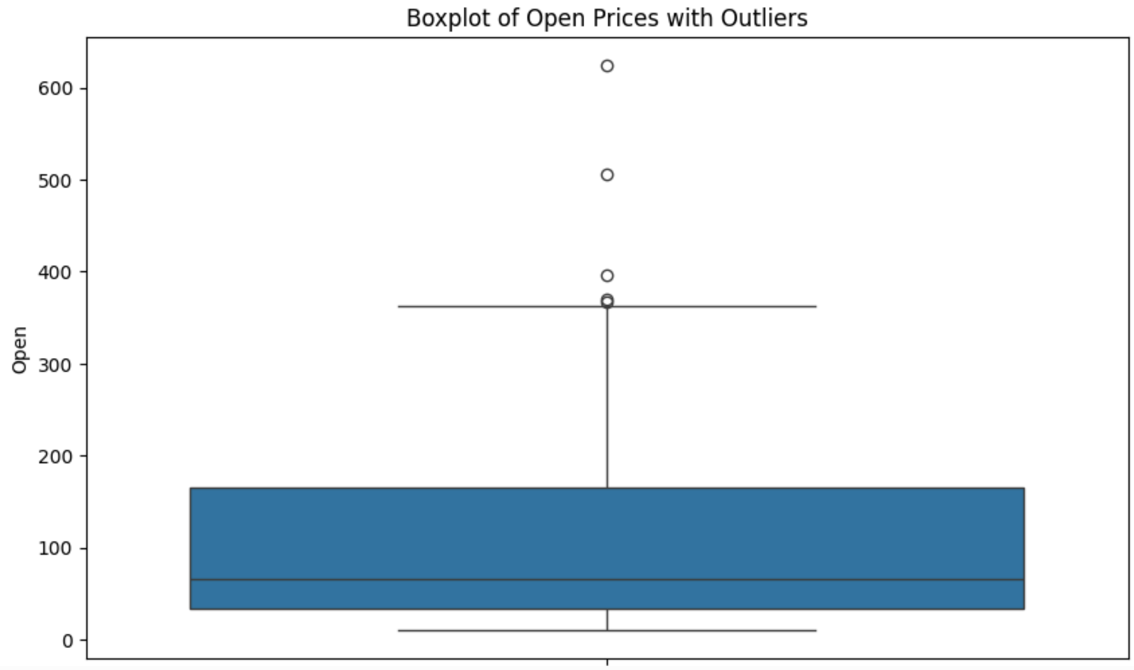
```
# Converting Date column from object format to Date
df["Date"] = pd.to_datetime(df["Date"], format='%b-%y')  # Changed 'data' to 'df'
df['Date']  # Changed 'data' to 'df'
```

	Date
0	2005-07-01
1	2005-08-01
2	2005-09-01
3	2005-10-01
4	2005-11-01
...	...
180	2020-07-01
181	2020-08-01
182	2020-09-01
183	2020-10-01
184	2020-11-01

185 rows × 1 columns

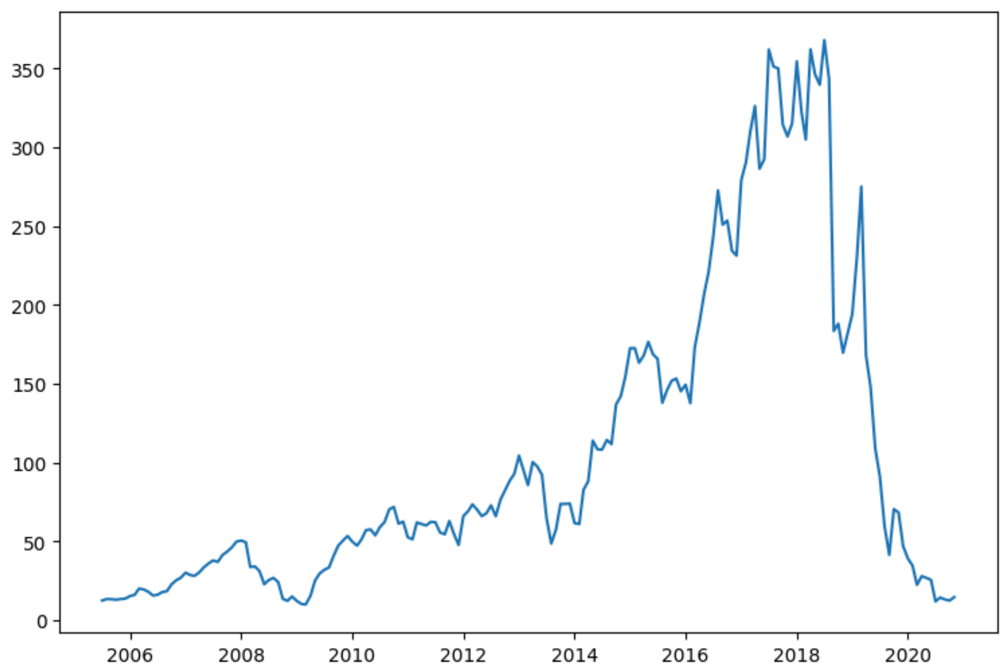
dtype: datetime64[ns]

```
# Example: Adding extreme outliers to 'Open' price
num_outliers = 5  # Number of outliers to add
outlier_factor = 3  # Factor to multiply standard deviation for outlier generation

# Calculate mean and standard deviation of 'Open'
# Changed 'data' to 'df' to match the DataFrame used earlier
open_mean = df['Open'].mean()
open_std = df['Open'].std()

# Generate random outliers for 'Open'
outliers = np.random.normal(open_mean + outlier_factor * open_std, open_std, num_outliers)

# Instead of concatenating, assign outliers to random existing rows
# Changed 'data' to 'df' to match the DataFrame used earlier
random_indices = np.random.choice(df.index, num_outliers, replace=False)  # Get random indices
df.loc[random_indices, 'Open'] = outliers  # Assign outliers to those indices

# Visualization: (Your existing visualization code, such as boxplots or scatter plots)
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
# Changed 'data' to 'df' to match the DataFrame used earlier
sns.boxplot(df['Open'])
plt.title('Boxplot of Open Prices with Outliers')
plt.show()
```




```
plt.figure(figsize=(9,6))
plt.plot(df['Date'],df['Close']) # Changed 'data' to 'df' to match the DataFrame created earlier
```

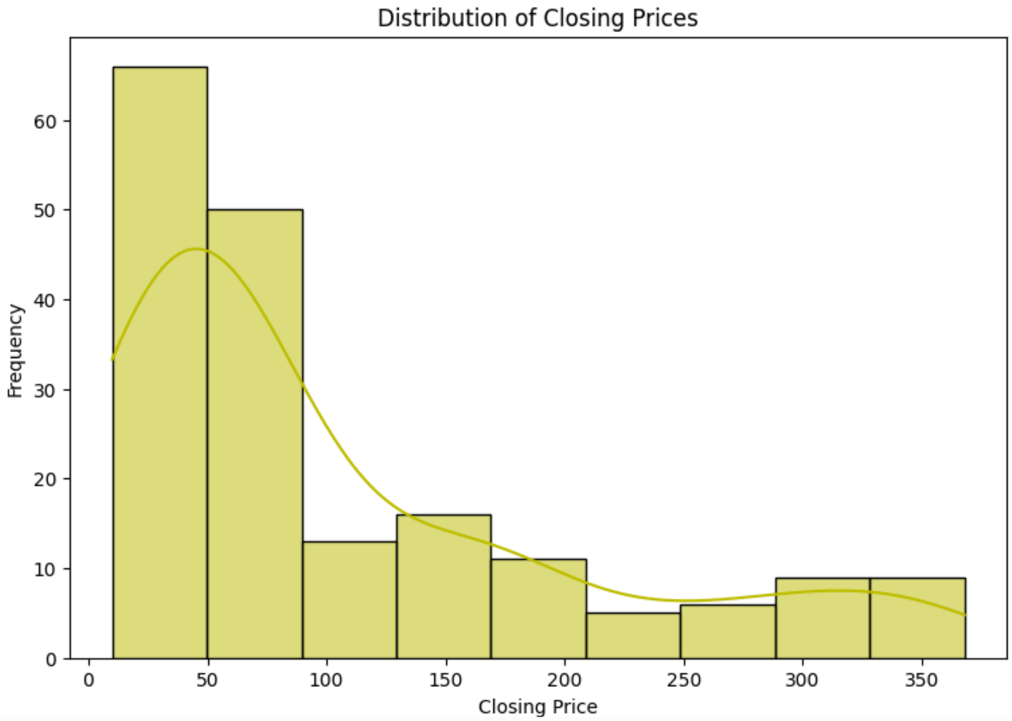
```
[<matplotlib.lines.Line2D at 0x7e0a4ae66a90>]
```



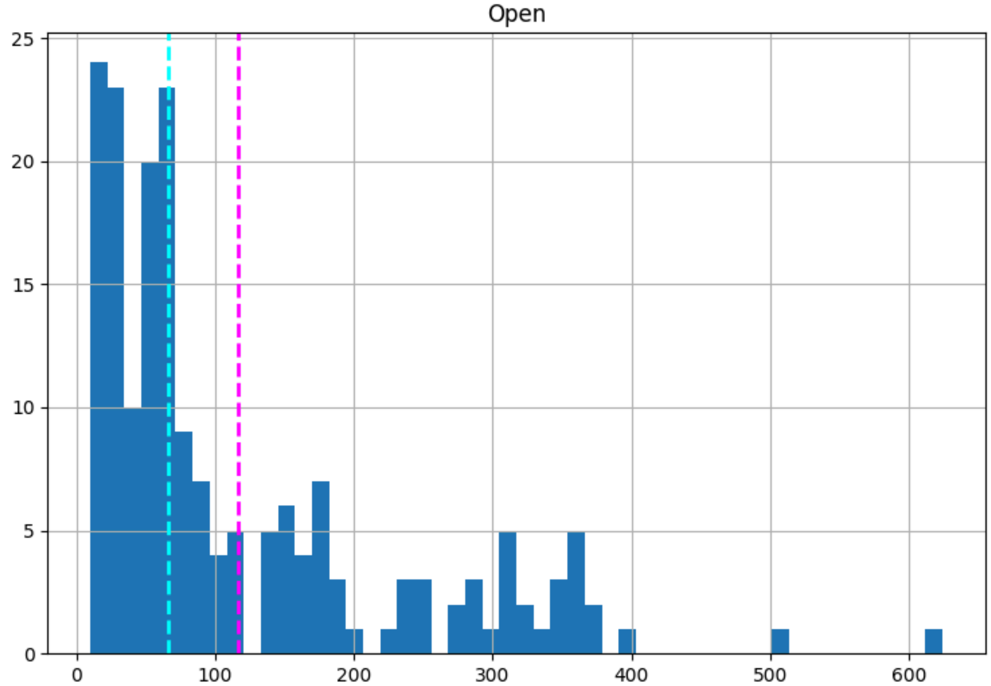

```
# Taking a Numerical Feature from Data
numeric_fea=df.describe().columns # Changed 'data' to 'df' to match the DataFrame created earlier
numeric_fea
```

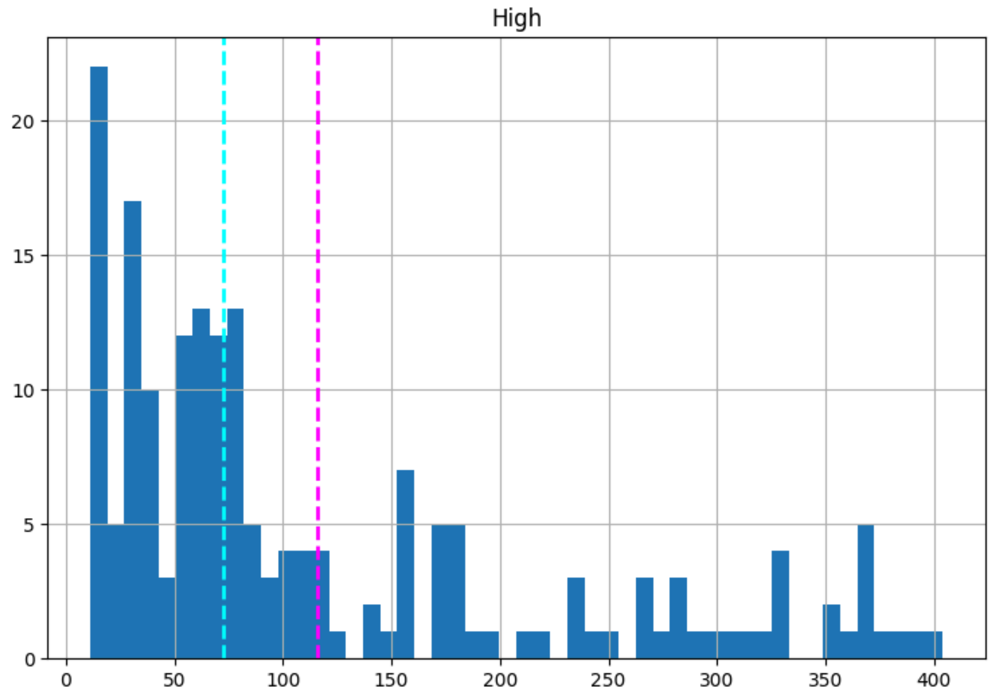
```
Index(['Date', 'Open', 'High', 'Low', 'Close'], dtype='object')
```

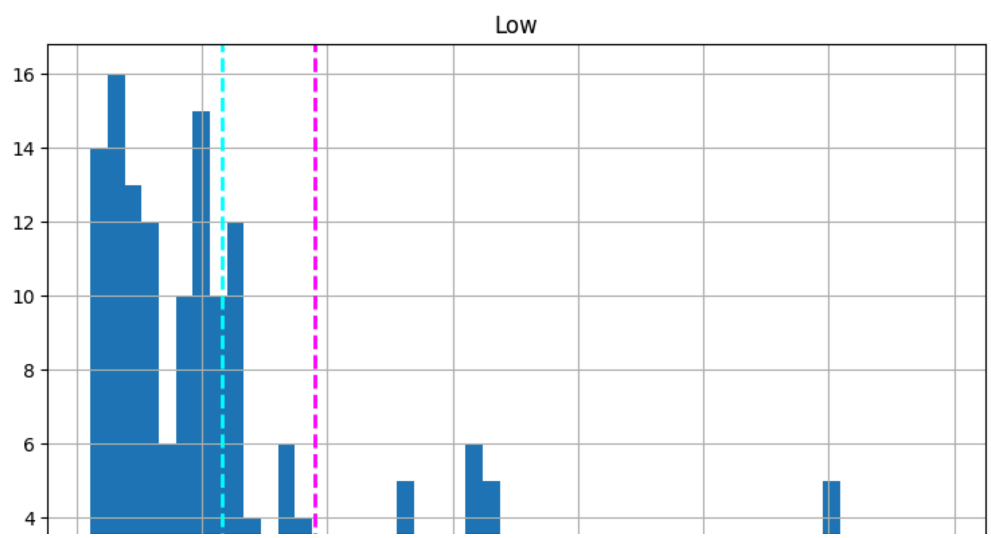
```
# Chart - 1 visualization code
# Doing Visualisation of Distributed Data for Close column.
plt.figure(figsize=(9,6))
sns.histplot(df['Close'], color='y', kde=True)  # kde=True adds the kernel density estimate
plt.title('Distribution of Closing Prices')
plt.xlabel('Closing Price')
plt.ylabel('Frequency')
plt.show()
```

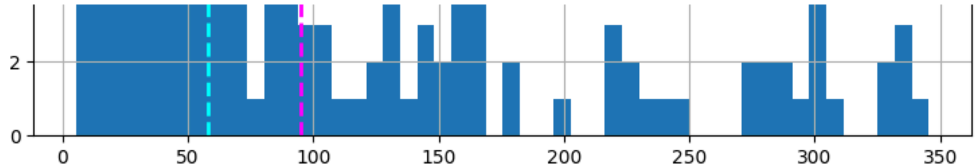



```
# Chart - 2 visualization code
# Plotting Histogram for each independent column in Data.
for col in numeric_fea[:-1]:
  if df[col].dtype in [np.int64, np.float64]:  # Check if column is numeric
    fig = plt.figure(figsize=(9, 6))
    ax = fig.gca()
    feature = df[col]
    feature.hist(bins=50, ax=ax)
    ax.axvline(feature.mean(), color='magenta', linestyle='dashed', linewidth=2)
    ax.axvline(feature.median(), color='cyan', linestyle='dashed', linewidth=2)
    ax.set_title(col)
plt.show()
```

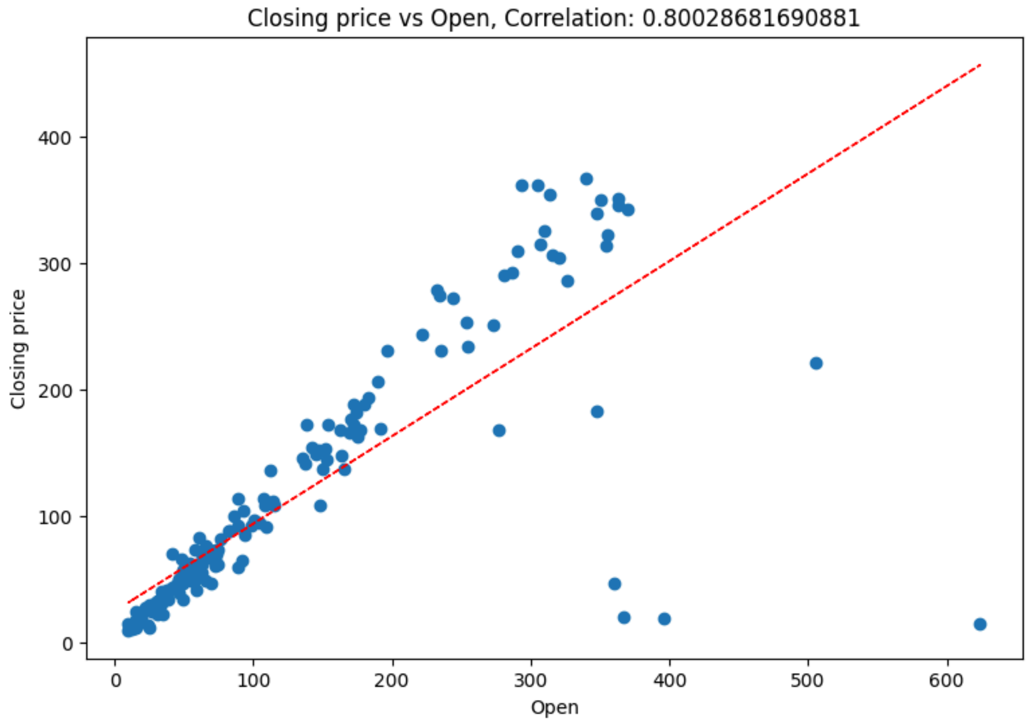

Open
0
5
10
15
20
25
0
100
200
300
400
500
600

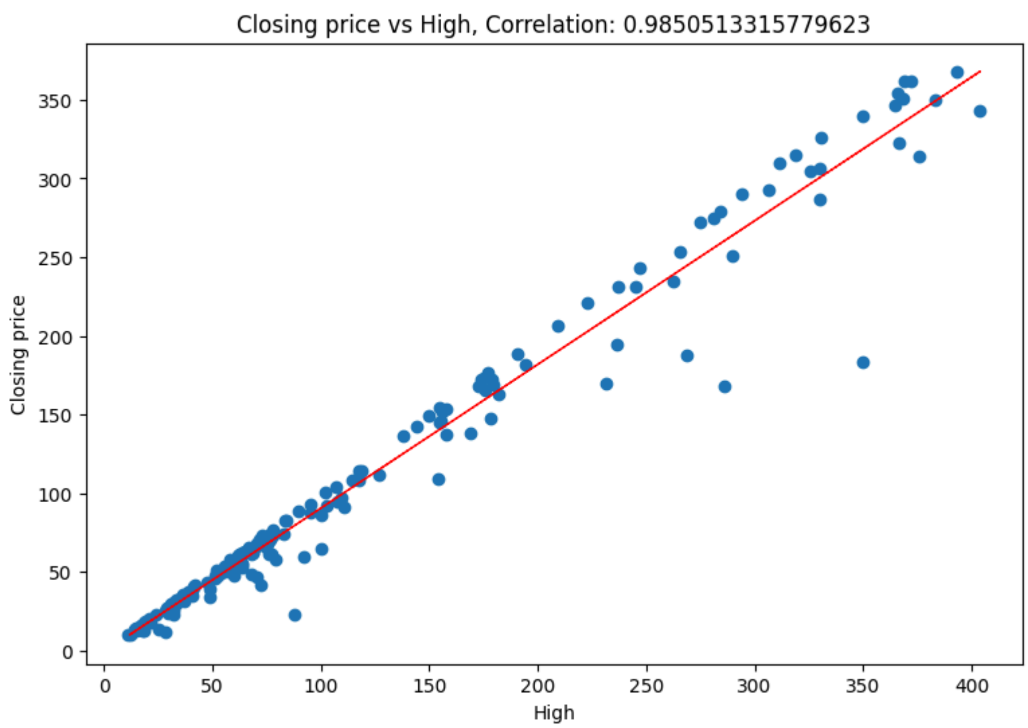

High
0
5
10
15
20
0
50
100
150
200
250
300
350
400


Low
4
6
8
10
12
14
16

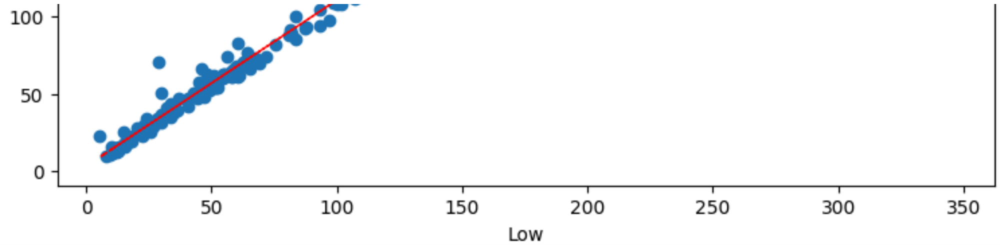



```
# Chart - 3 visualization code
# Plotting graph Independent variable vs Dependent variable to check Multicollinearity.
for col in numeric_fea[:-1]:
  if col != 'Date':  # Skip the 'Date' column
    fig=plt.figure(figsize=(9,6))
    ax=fig.gca()
    feature=df[col]
    label=df["Close"]
    correlation=feature.corr(label)
    plt.scatter(x=feature,y=label)
    plt.ylabel("Closing price")
    plt.xlabel(col)
    ax.set_title('Closing price vs '+col+', Correlation: '+str(correlation))
    z=np.polyfit(df[col],df['Close'],1)
    y_hat=np.poly1d(z)(df[col])
    plt.plot(df[col],y_hat,"r--",lw=1)
    plt.show()
```


Closing price vs Open, Correlation: 0.80028681690881
Closing price
Open
0
100
200
300
400
500
600

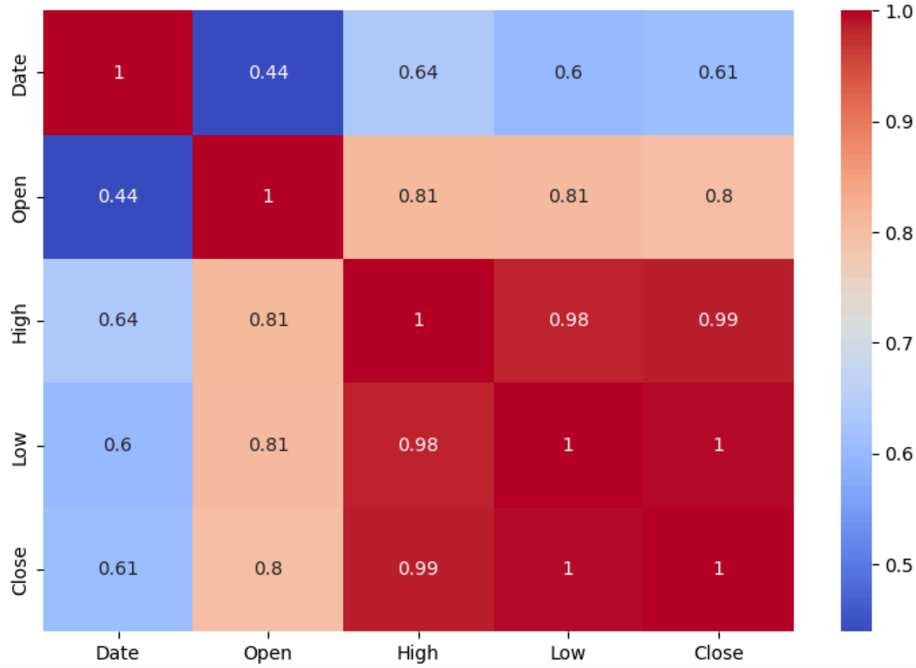

Closing price vs High, Correlation: 0.9850513315779623
Closing price
High
0
50
100
150
200
250
300
350
400


Closing price vs Low, Correlation: 0.9953579476474373
Closing price
150
200
250
300
350




```
# Chart - 4 Correlation Heatmap visualization code
# Heatmap to see collinearity between columns
plt.figure(figsize=(9,6))
cor=df.corr()
sns.heatmap(abs(cor),annot=True,cmap='coolwarm')
```

<Axes: >




```
# Scaling your data
data_pr=df.copy() # Making copy of our original data
# Separate Dependent and Independent variable
X=np.log10(data_pr.iloc[:,1:-1]) # Normalizing the data using log transformation
y=np.log10(data_pr['Close']) # Normalizing the data using log transformation
# Split your data to train and test. Choose Splitting ratio wisely.
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
print(X_train.shape)
print(X_test.shape)
```

```
(148, 3)
(37, 3)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# 1. Load the Dataset
data = pd.read_csv("/content/YesBank_StockPrices (2).csv")
target = 'Close'

# 2. Define features  (This line is added)
features = ['Open', 'High', 'Low']

X = data[features]
y = data[target]

# 3. Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# 4. Train the Model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Predict for New Data
new_data = pd.DataFrame({
    'Open': [150],
    'High': [155],
    'Low': [145]
})

predicted_close = model.predict(new_data)

# 6. Print Only the Predicted Value
print(predicted_close[0])
```

```
153.9692776654747
```

```
# Visualization: Bar Graph
plt.figure(figsize=(6, 4))
plt.bar(['Predicted Close'], [predicted_close[0]], color='skyblue', label='Predicted')
plt.axhline(y=predicted_close[0], color='red', linestyle='--', label=f'Predicted Value ({predicted_close[0]:.2f})')
plt.ylabel('Stock Price')
plt.title('Predicted Closing Price for Yes Bank Stock')
plt.legend()
plt.show()
```