In [1]:
```
#import libraries
import pandas as pd
import numpy as np
from sklearn import metrics
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv("GOOG.csv")
df.head()
```

Out[2]:

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	splitFactor
0	GOOG	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	1306065	718.27	722.47	713.1200	716.48	1306065	0.0	1.0
1	GOOG	2016-06-15 00:00:00+00:00	718.92	722.98	717.3100	719.00	1214517	718.92	722.98	717.3100	719.00	1214517	0.0	1.0
2	GOOG	2016-06-16 00:00:00+00:00	710.36	716.65	703.2600	714.91	1982471	710.36	716.65	703.2600	714.91	1982471	0.0	1.0
3	GOOG	2016-06-17 00:00:00+00:00	691.72	708.82	688.4515	708.65	3402357	691.72	708.82	688.4515	708.65	3402357	0.0	1.0
4	GOOG	2016-06-20 00:00:00+00:00	693.71	702.48	693.4100	698.77	2082538	693.71	702.48	693.4100	698.77	2082538	0.0	1.0

In [3]:
```
df.shape
```

Out[3]: (1258, 14)

In [4]:
```
df=df.drop(columns=[
    'symbol','adjClose','adjHigh','adjLow','adjOpen','adjVolume','divCash','splitFactor'
],axis=1)
df.head()
```

Out[4]:

	date	close	high	low	open	volume
0	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	1306065
1	2016-06-15 00:00:00+00:00	718.92	722.98	717.3100	719.00	1214517
2	2016-06-16 00:00:00+00:00	710.36	716.65	703.2600	714.91	1982471
3	2016-06-17 00:00:00+00:00	691.72	708.82	688.4515	708.65	3402357
4	2016-06-20 00:00:00+00:00	693.71	702.48	693.4100	698.77	2082538

In [5]:
```
#Are there any Duplicate values
df.duplicated().sum().any()
```

Out[5]: False

In [6]:
```
# Cheaking & reviewing DataFrame information
df.isnull().values.any()
```

Out[6]: False

In [7]:
```
df.describe()
```

Out[7]:

	close	high	low	open	volume
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03
mean	1216.317067	1227.430934	1204.176430	1215.260779	1.601590e+06
std	383.333358	387.570872	378.777094	382.446995	6.960172e+05
min	668.260000	672.300000	663.284000	671.000000	3.467530e+05
25%	960.802500	968.757500	952.182500	959.005000	1.173522e+06
50%	1132.460000	1143.935000	1117.915000	1131.150000	1.412588e+06
75%	1360.595000	1374.345000	1348.557500	1361.075000	1.812156e+06
max	2521.600000	2526.990000	2498.290000	2524.920000	6.207027e+06

In [10]:
```
df['date'] = pd.to_datetime(df['date'])
df.head()
```

Out[10]:

	date	close	high	low	open	volume
0	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	1306065
1	2016-06-15 00:00:00+00:00	718.92	722.98	717.3100	719.00	1214517
2	2016-06-16 00:00:00+00:00	710.36	716.65	703.2600	714.91	1982471
3	2016-06-17 00:00:00+00:00	691.72	708.82	688.4515	708.65	3402357
4	2016-06-20 00:00:00+00:00	693.71	702.48	693.4100	698.77	2082538

In [11]:

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

Out[11]:

	date	close	high	low	open	volume
0	2016-06-14 00:00:00+00:00	718.27	722.47	713.1200	716.48	1306065
1	2016-06-15 00:00:00+00:00	718.92	722.98	717.3100	719.00	1214517
2	2016-06-16 00:00:00+00:00	710.36	716.65	703.2600	714.91	1982471
3	2016-06-17 00:00:00+00:00	691.72	708.82	688.4515	708.65	3402357
4	2016-06-20 00:00:00+00:00	693.71	702.48	693.4100	698.77	2082538

In [12]:

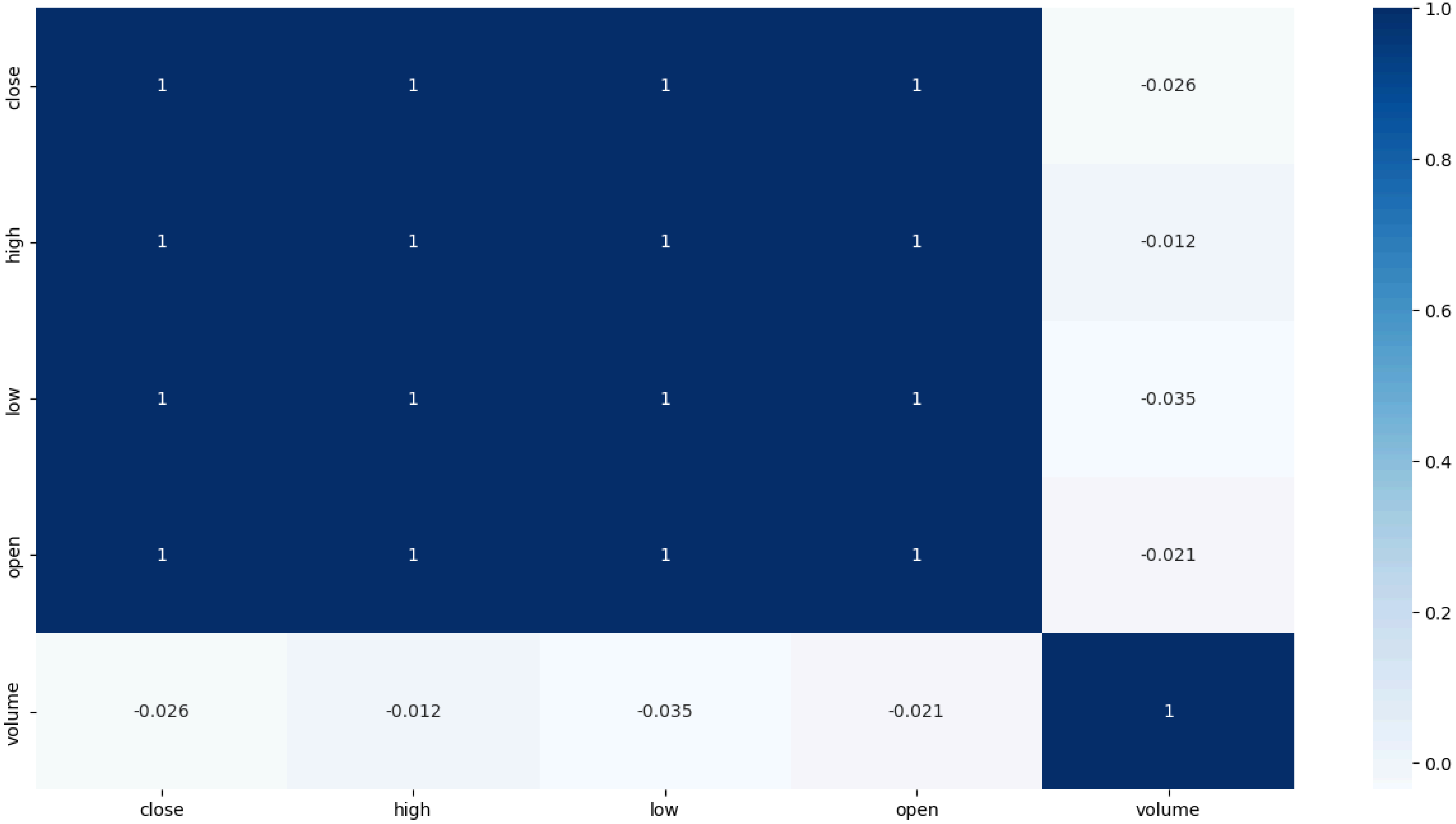
```
df['date'] = df['date'].dt.strftime('%Y-%m-%d')
df.head()
```

Out[12]:

	date	close	high	low	open	volume
0	2016-06-14	718.27	722.47	713.1200	716.48	1306065
1	2016-06-15	718.92	722.98	717.3100	719.00	1214517
2	2016-06-16	710.36	716.65	703.2600	714.91	1982471
3	2016-06-17	691.72	708.82	688.4515	708.65	3402357
4	2016-06-20	693.71	702.48	693.4100	698.77	2082538

Visulation Correlations

In [18]:
```
# Assuming df is your DataFrame and you want to drop non-numeric columns before plotting
numeric_df = df.select_dtypes(include=['number'])

# Plot the correlation heatmap
plt.figure(figsize=(16, 8))
sns.heatmap(numeric_df.corr(), cmap="Blues", annot=True)
plt.show()
```

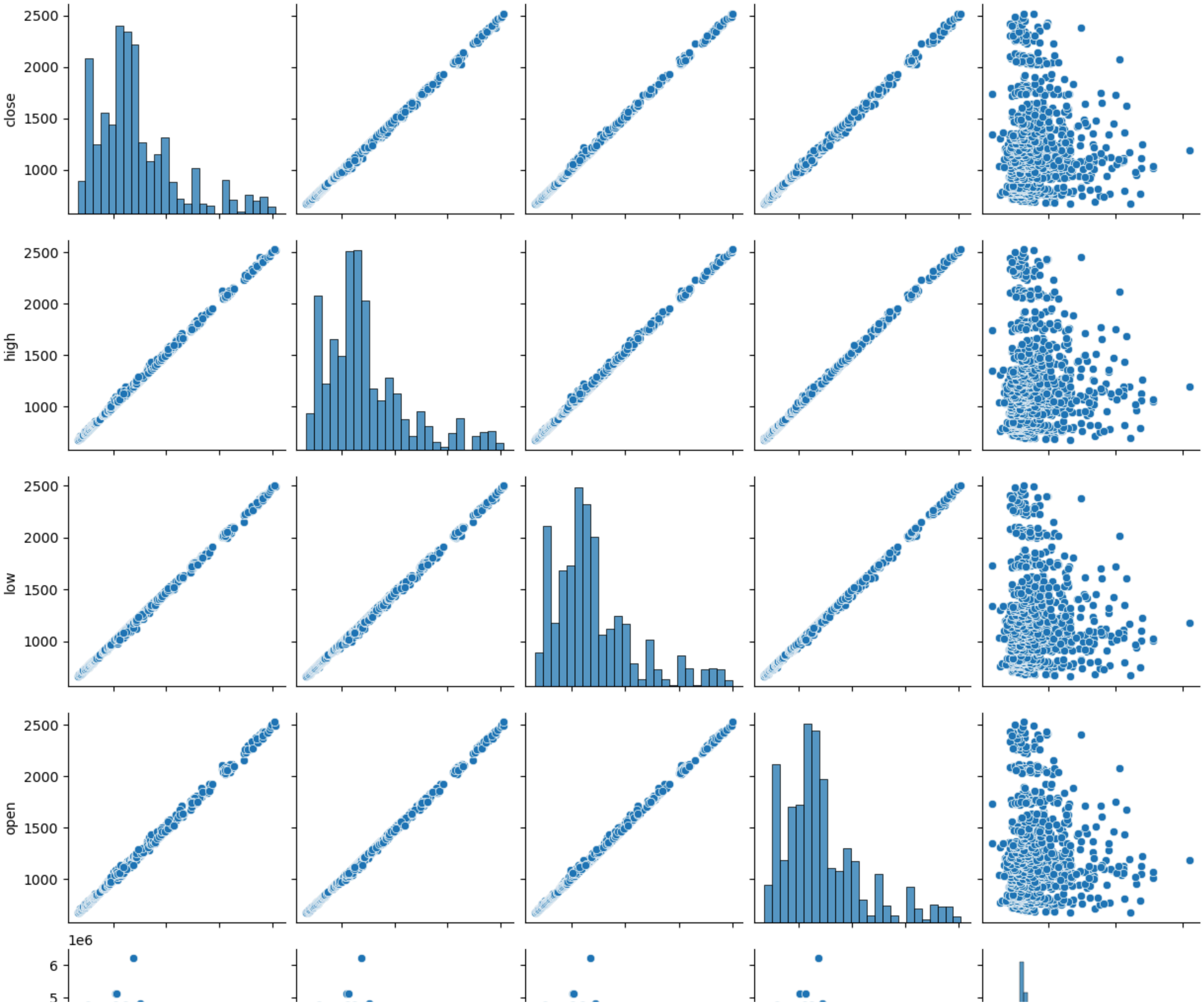


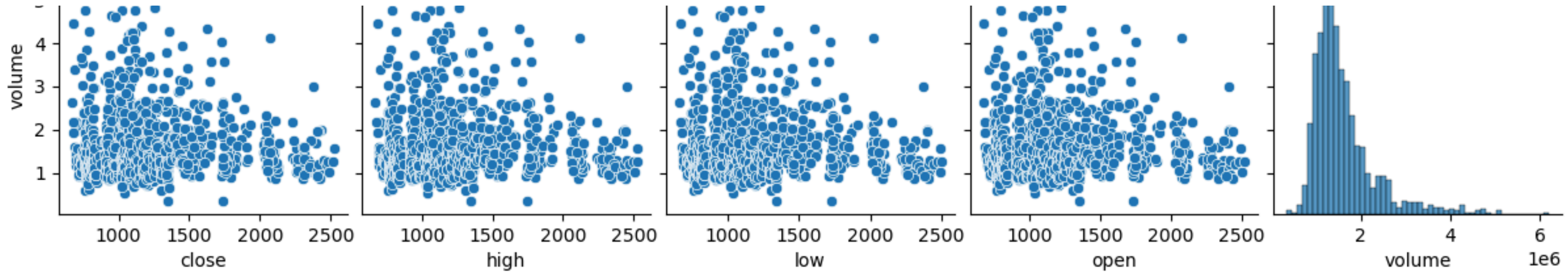
Visualization overview of relationships in dataset

In [19]:
```
#showing visualization on all variables in data
sns.pairplot(df)
```

```
C:\Users\KIIT\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

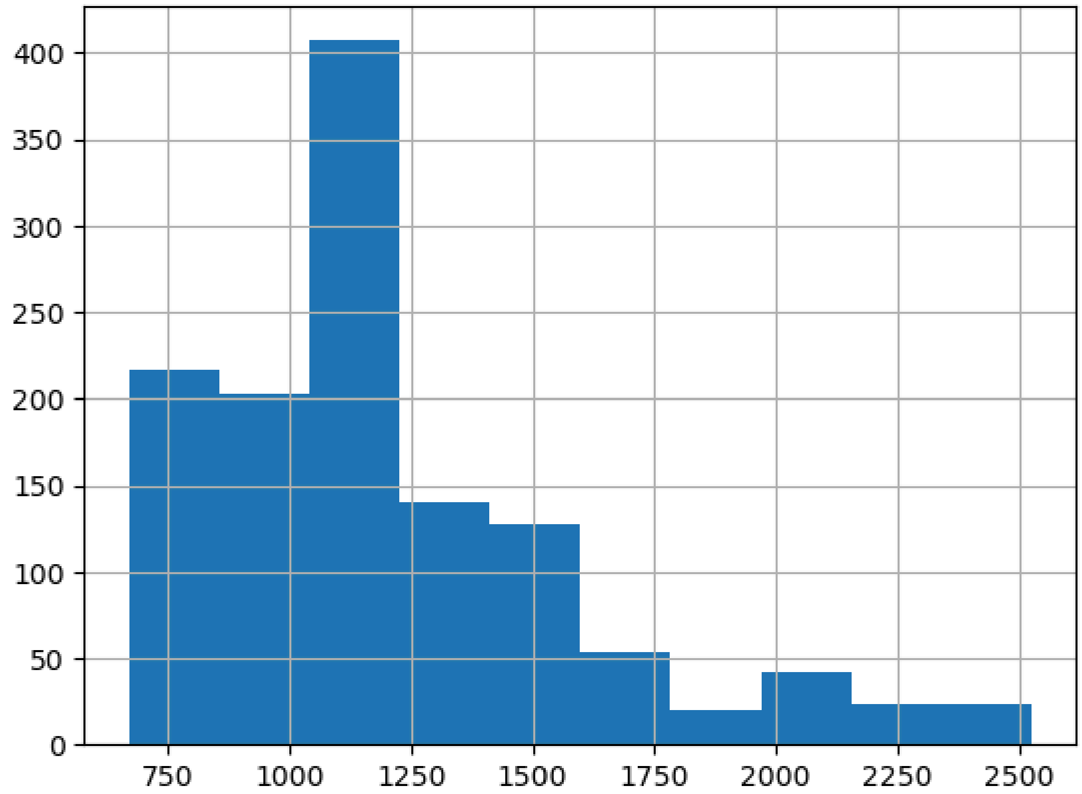
Out[19]:
```
<seaborn.axisgrid.PairGrid at 0x1f790a98210>
```


2500
2000
1500
1000
close
high
low
open
1e6
6
5



In [20]:
```python
df['open'].hist()
```

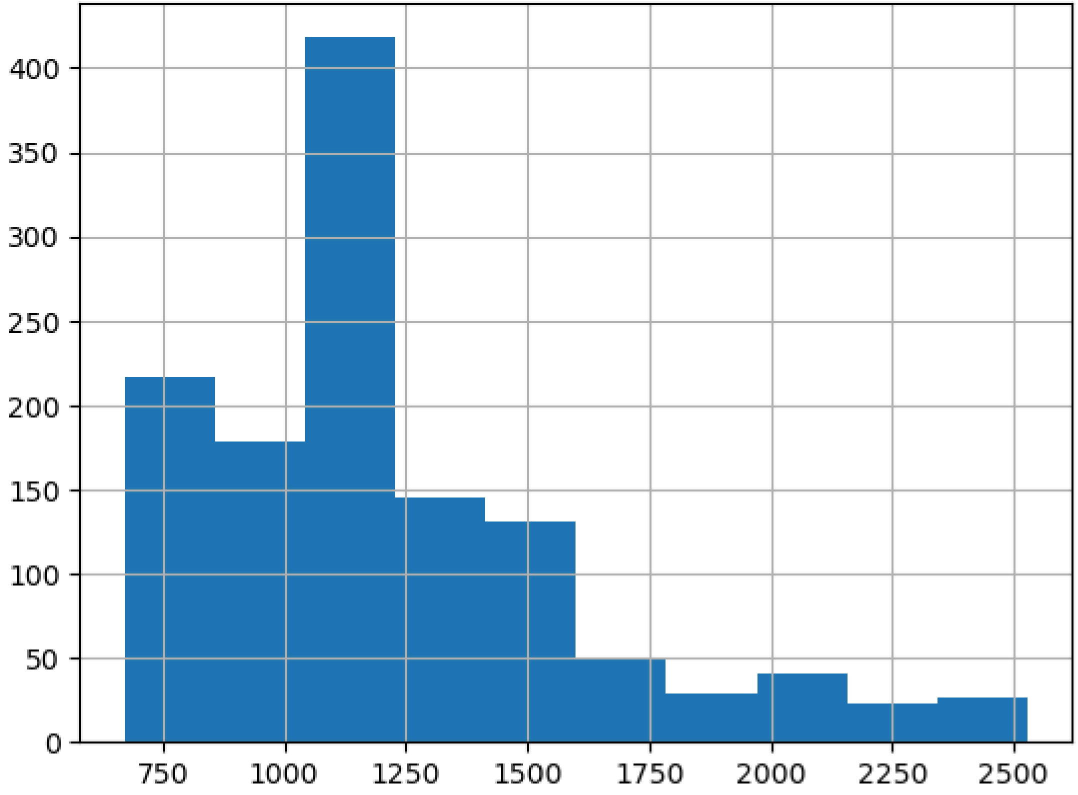
Out[20]: <Axes: >



In [21]:

```
df['high'].hist()
```

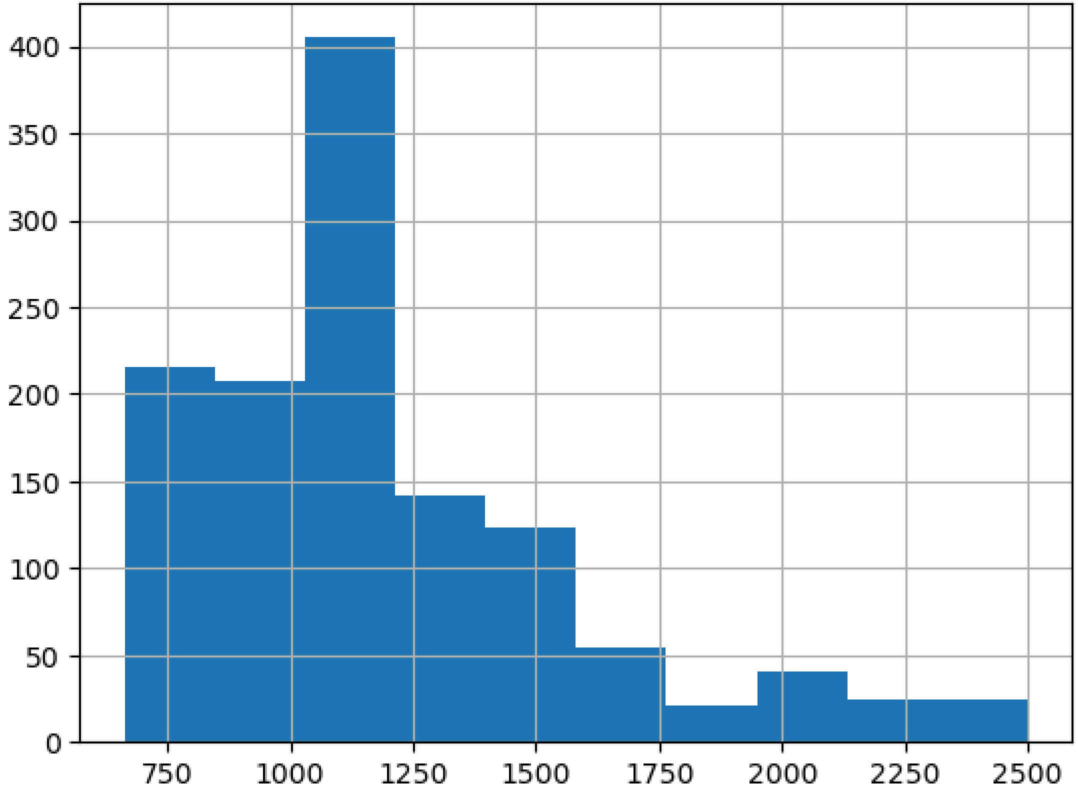
Out[21]: <Axes: >



In [22]:

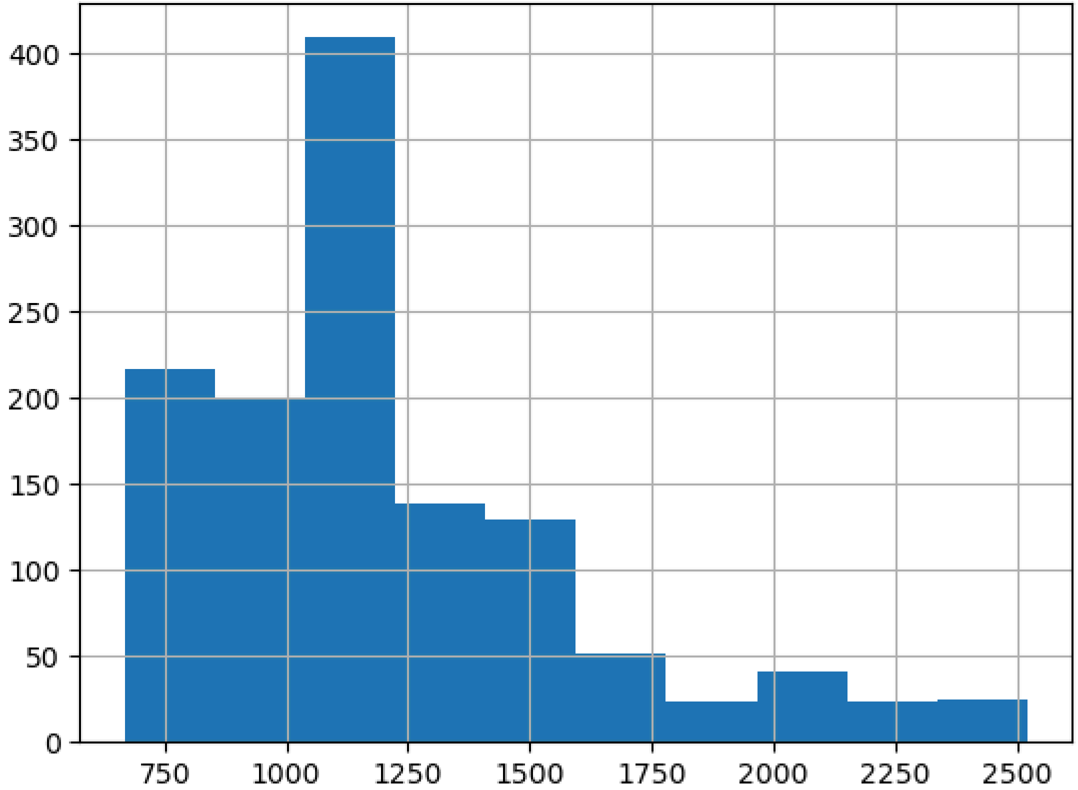
```
df['low'].hist()
```

Out[22]: <Axes: >



In [23]:
```
df['close'].hist()
```

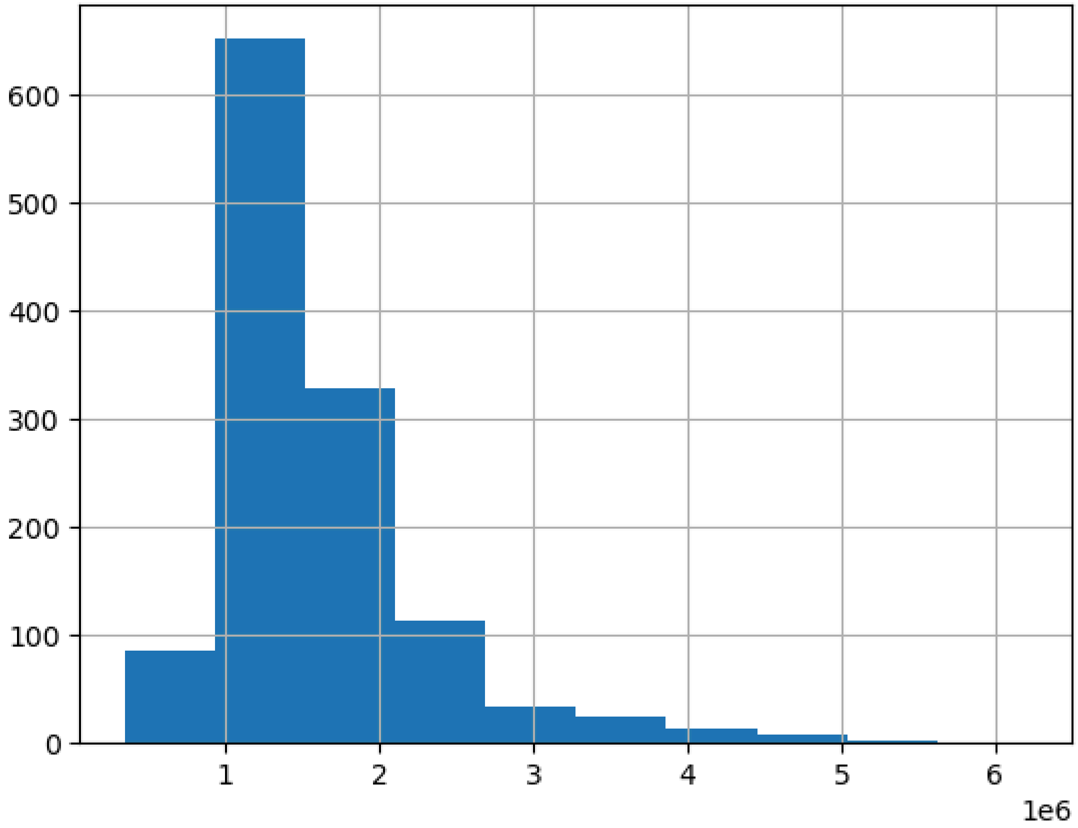
Out[23]: <Axes: >



In [24]:

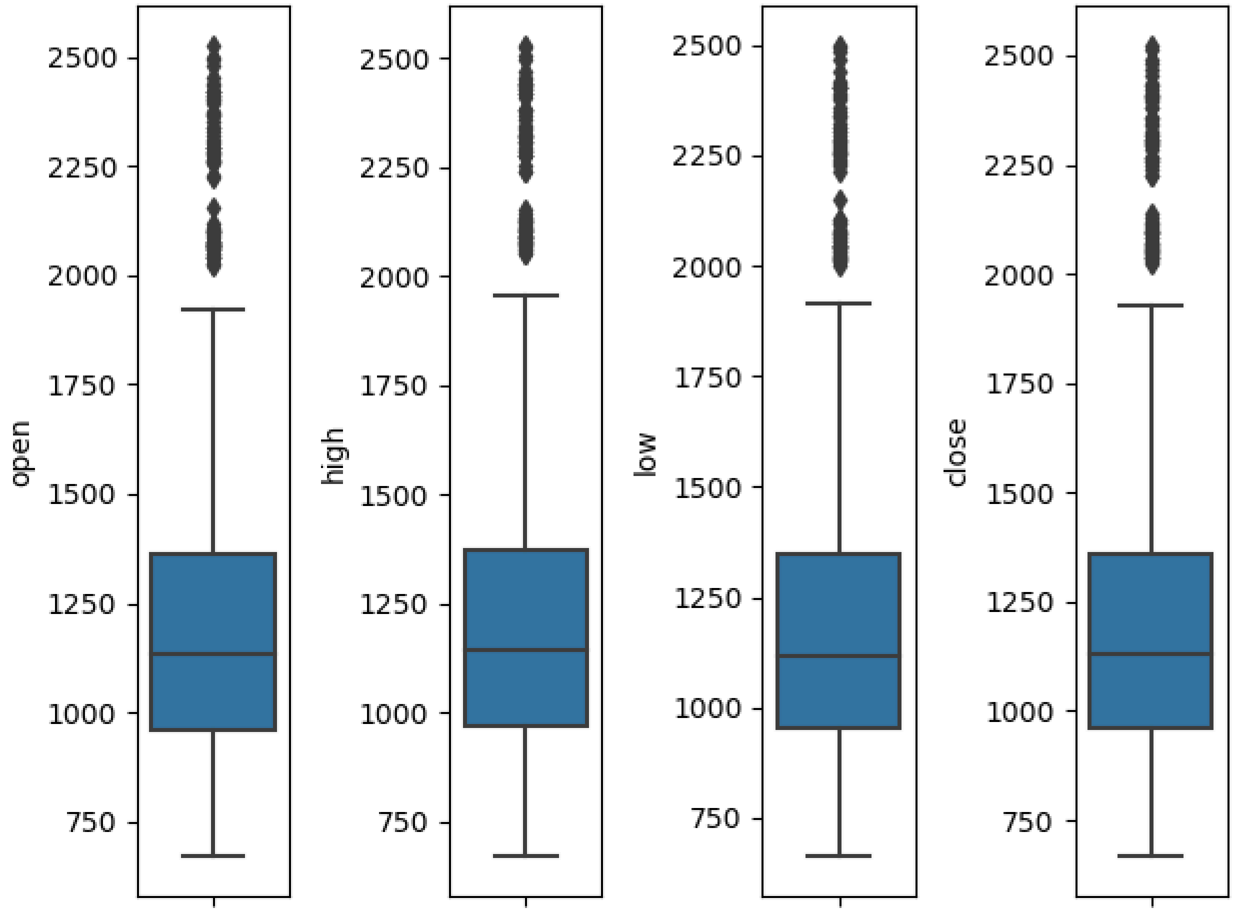
```
df['volume'].hist()
```

Out[24]: <Axes: >

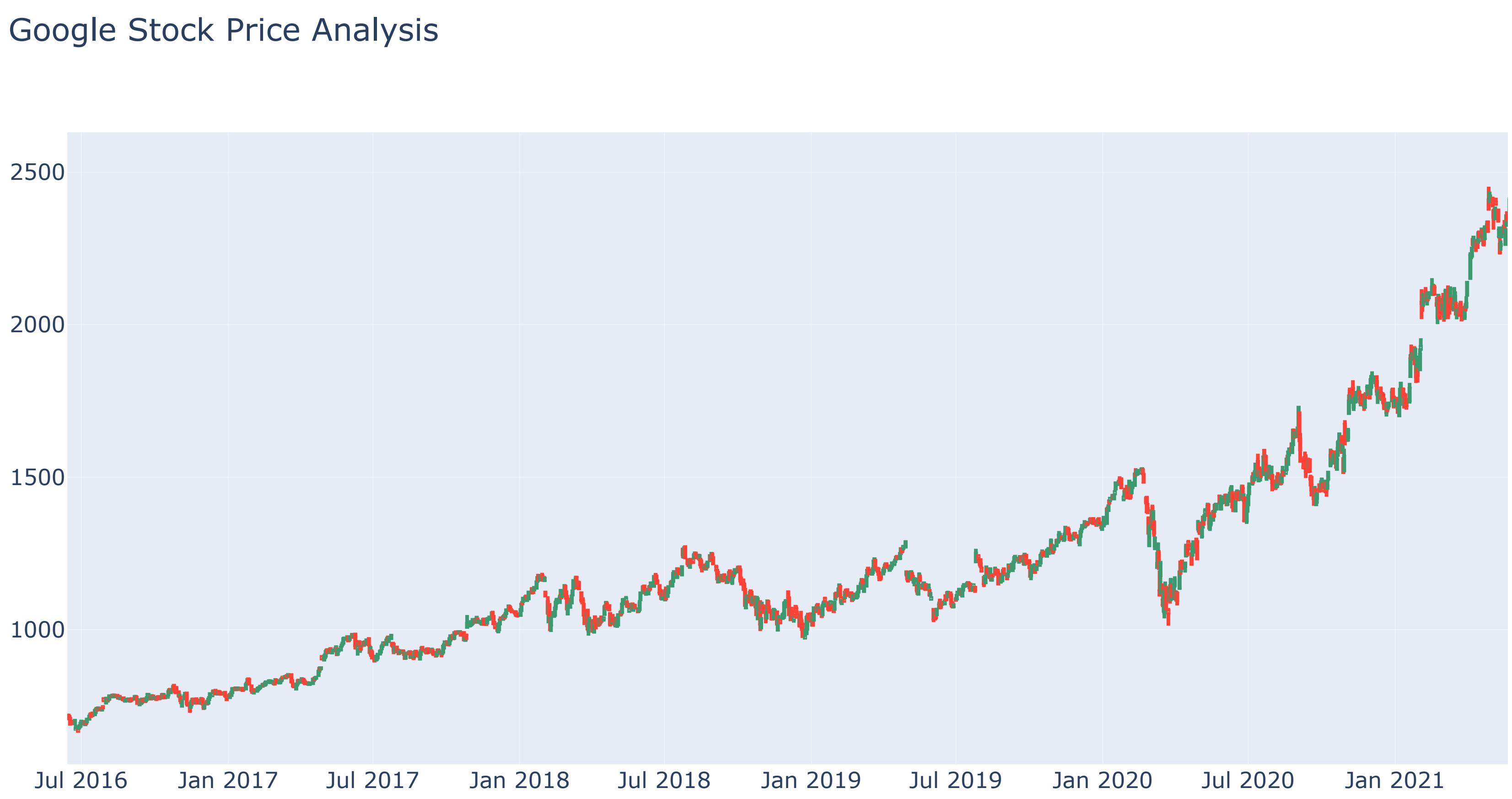


In [25]:

```
#Review box plots
f, axes = plt.subplots(1,4)
sns.boxplot(y='open', data=df, ax=axes[0])
sns.boxplot(y='high', data=df, ax=axes[1])
sns.boxplot(y='low', data=df, ax=axes[2])
sns.boxplot(y='close', data=df, ax=axes[3])
plt.tight_layout()
```



In [26]:
```
import plotly.graph_objects as go
figure = go.Figure(data=[go.Candlestick(x=df["date"],open=df["open"],high=df["high"],low=df["low"],close=df["close
figure.update_layout(title= "Google Stock Price Analysis", xaxis_rangeslider_visible=False)
figure.show()
```



Split the dataset

In [27]:
```
x=df[['open','high','low','volume']].values # independent variables
y=df['close'].values # dependent variable
```

Split the data 80% train and 20% testing

In [28]:
```
from sklearn.model_selection import train_test_split
# Splitting the data 80% train and 20% testing
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2,random_state=0)
```

In [29]:
```
# Checking the shape for train data
print('Train:', x_train.shape)
print('test:', x_test.shape)
```

```
Train: (1006, 4)
test: (252, 4)
```

Training the model Linear Regression

In [30]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix, accuracy_score
import statsmodels.api as sm

# Creating Regression Model
regressor=LinearRegression()

# fit linear regression model
model=regressor.fit(x_train,y_train)

# Use model to make predictions
y_pred=regressor.predict(x_test)
```

Prediction

In [31]:
```
# With the test predictions complete, the next step will better compare them
# with the actual output values for x_test by organizing them in a DataFrame format:

predicted=regressor.predict(x_test)
```

In [32]:
```
# x_test shape
predicted.shape
```

Out[32]:
```
(252,)
```

Validating the Fit

In [34]:
```
# Printout relevant metrics
print("Model Coefficients:",regressor.coef_)
#Looking at the intercept
print("Model Intercept:",regressor.intercept_)
```

```
Model Coefficients: [-5.54784375e-01  7.77461854e-01  7.76833889e-01 -4.55059829e-07]
Model Intercept: 1.4776059638254537
```

Prediction Table of Actual Prices vs Predicted values

In [35]:
```
dframe=pd.DataFrame(y_test,predicted)
dfr=pd.DataFrame({'Actual_Price':y_test,'Predicted_Price':predicted})
print(dfr)
```

```
     Actual_Price  Predicted_Price
0          695.94       697.302933
1         1084.99      1090.146792
2          769.54       772.628263
3         1349.33      1345.790934
4          843.25       841.900950
..            ...              ...
247       1567.24      1577.560900
248        745.91       741.785159
249       1175.84      1162.560630
250        762.49       766.104077
251       1036.23      1032.660476

[252 rows x 2 columns]
```

In [36]:
```
# Stats on Actual price & Predicted price
dfr.describe()
```

Out[36]:

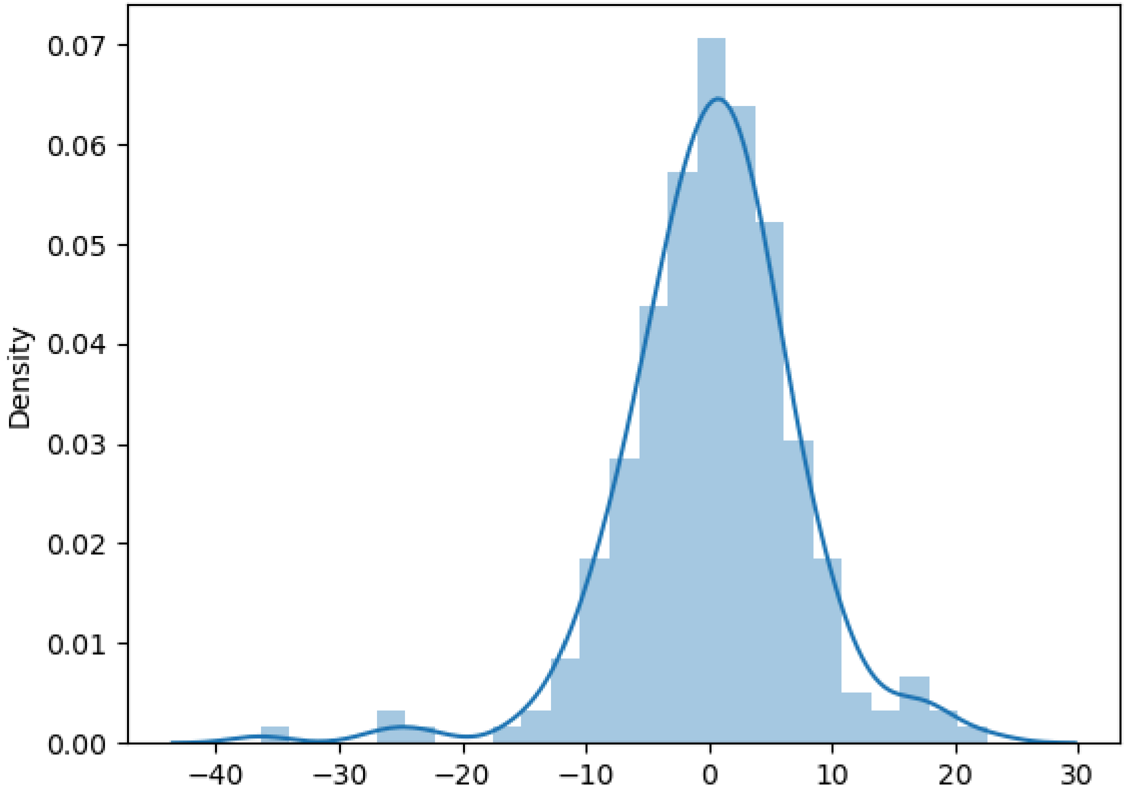
	Actual_Price	Predicted_Price
count	252.00000	252.000000
mean	1239.92381	1239.673289
std	378.69218	379.364626
min	675.22000	675.546297
25%	1028.45250	1021.774677
50%	1163.42500	1158.541487
75%	1428.65000	1429.116931
max	2411.56000	2419.701570

Normality of Residual

In [37]:
```
# This is the difference of y_test values subtracting the prediction values
residual = y_test - predicted
sns.distplot(residual)
```

```
C:\Users\KIIT\AppData\Local\Temp\ipykernel_24084\785408112.py:3: UserWarning:



`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed29744
57ad6372750bbe5751)
```

Out[37]:
```
<Axes: ylabel='Density'>
```



In [38]:

```
# Checking p-value with right tailed or upper tailed test

#importing scipy library
import scipy.stats

# finding p-value
p_value=scipy.stats.norm.sf(abs(1.67))
print('p value is : ' + str(p_value))
```

```
p value is : 0.04745968180294733
```

In [39]:

```
# Printing the Ordinary least squares (OLS) Regression Results model
results3=sm.OLS(y_test,x_test).fit()
results3.summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	1.000
Model:	OLS	**Adj. R-squared (uncentered):**	1.000
Method:	Least Squares	**F-statistic:**	2.090e+06
Date:	Fri, 15 Mar 2024	**Prob (F-statistic):**	0.00
Time:	12:47:28	**Log-Likelihood:**	-850.12
No. Observations:	252	**AIC:**	1708.
Df Residuals:	248	**BIC:**	1722.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.5442	0.046	-11.952	0.000	-0.634	-0.455
x2	0.6887	0.042	16.487	0.000	0.606	0.771
x3	0.8558	0.043	19.965	0.000	0.771	0.940
x4	1.274e-06	5.72e-07	2.226	0.027	1.47e-07	2.4e-06

Omnibus:	20.727	**Durbin-Watson:**	1.992
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	67.797
Skew:	-0.184	**Prob(JB):**	1.90e-15
Kurtosis:	5.514	**Cond. No.**	2.15e+05

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 2.15e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [40]:

```
# Checking the regression score
from sklearn.metrics import confusion_matrix, accuracy_score

regression_confidence=regressor.score(x_test,y_test)
print("Linear regression confidence: ",regression_confidence)
```

```
Linear regression confidence:  0.9996411863249148
```

Evaluating the Model

In [41]:

```
# Evaluating the Model- the closer to zero for all these metrics the better.
import math

print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,predicted))
print('Mean Squared Error:',metrics.mean_squared_error(y_test,predicted))
print('Root Mean Squared Error:',math.sqrt(metrics.mean_squared_error(y_test,predicted)))
```

```
Mean Absolute Error: 5.17001896276311
Mean Squared Error: 51.25247480680476
Root Mean Squared Error: 7.159083377556429
```

Model Accuracy

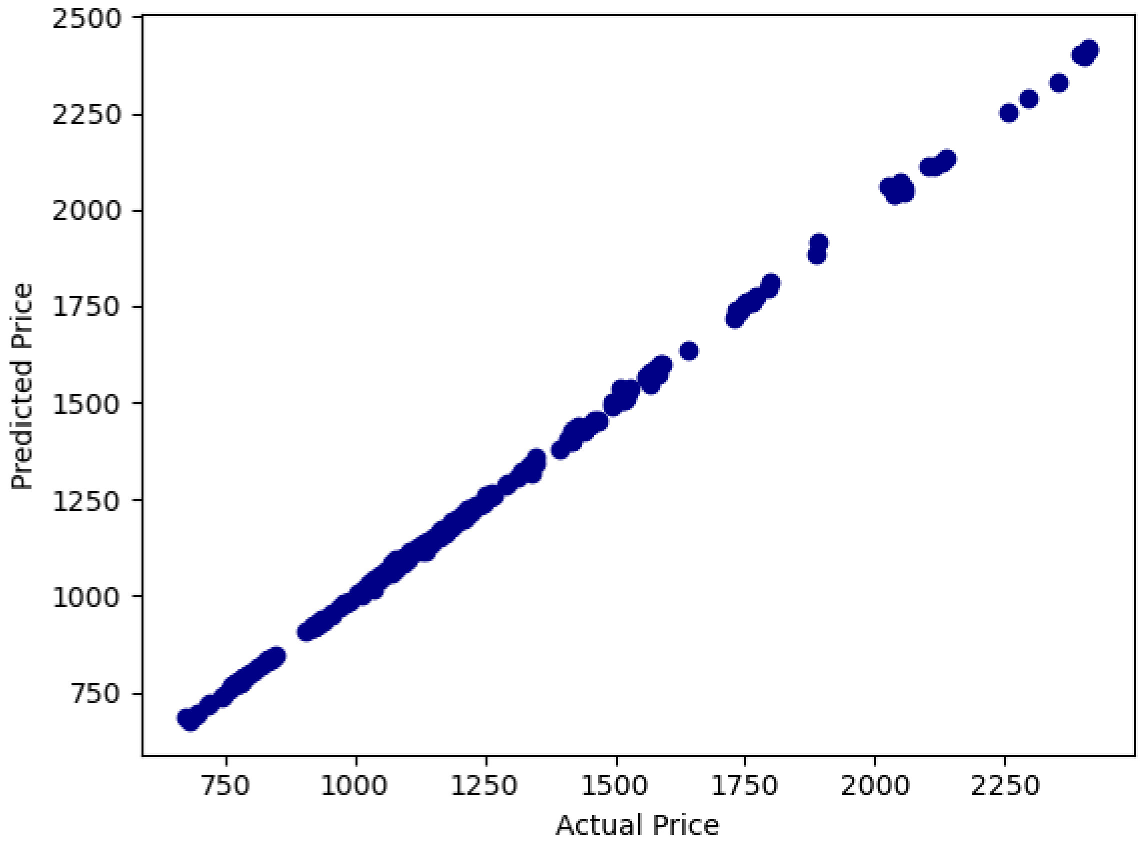
In [42]:

```
x2=abs(predicted-y_test)
y2=100*(x2/y_test)
accuracy=100-np.mean(y2)
print('Accuracy:',round(accuracy,2),'%.')
```

```
Accuracy: 99.59 %.
```

In [43]:

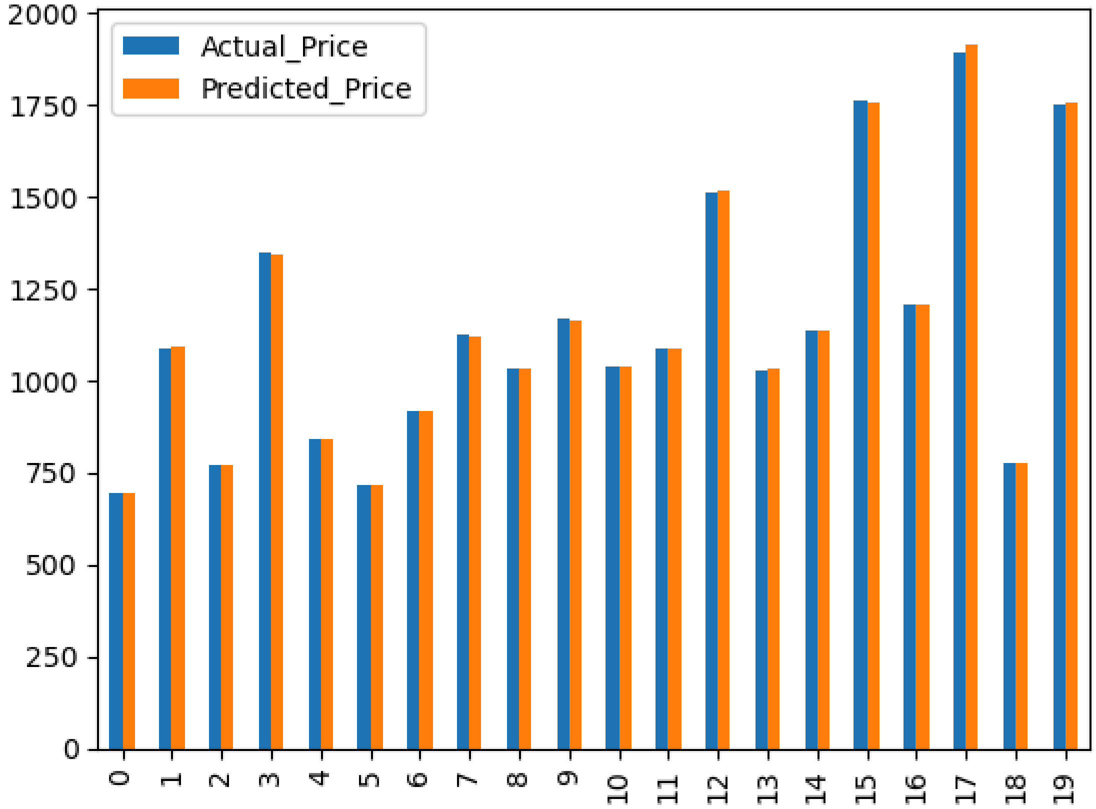
```
plt.scatter(dfr.Actual_Price, dfr.Predicted_Price, color='Darkblue')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.show()
```



Graphing the first 20 values

In [45]:
```
graph=dfr.head(20)
graph.plot(kind='bar')
```

Out[45]: <Axes: >



Stock Price Prediction using Regression Model

The End