GoogleStockAnalysis

November 7, 2024

Lee Johnston 10/12/2024

Google Stock Prediction/Analysis

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import neighbors
import scipy.stats
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree
import plotly.express as px
```

[3]:
```
df = pd.read_csv("GOOG.csv")

df.head(5)
```

[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	2.490664	2.591785	2.390042	2.499133	2.499133	897427216
1	2004-08-20	2.515820	2.716817	2.503118	2.697639	2.697639	458857488
2	2004-08-23	2.758411	2.826406	2.716070	2.724787	2.724787	366857939
3	2004-08-24	2.770615	2.779581	2.579581	2.611960	2.611960	306396159
4	2004-08-25	2.614201	2.689918	2.587302	2.640104	2.640104	184645512

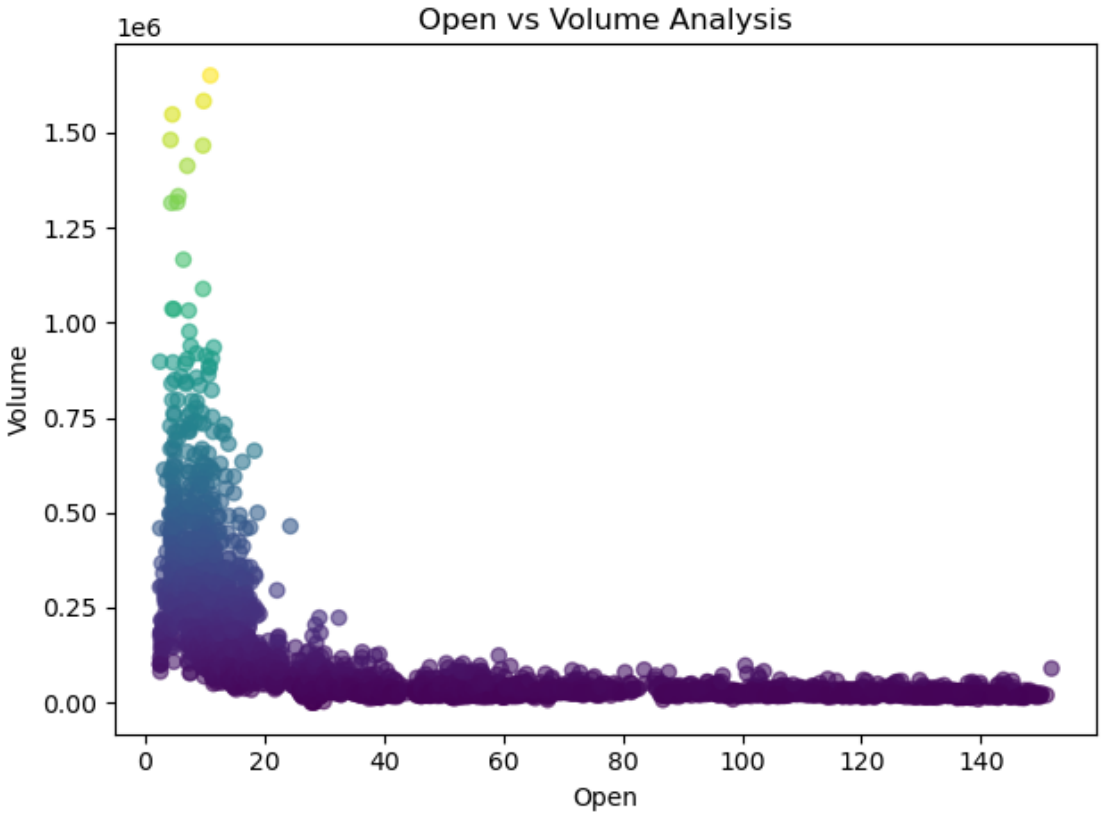
[4]:
```
df.columns
```

[4]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

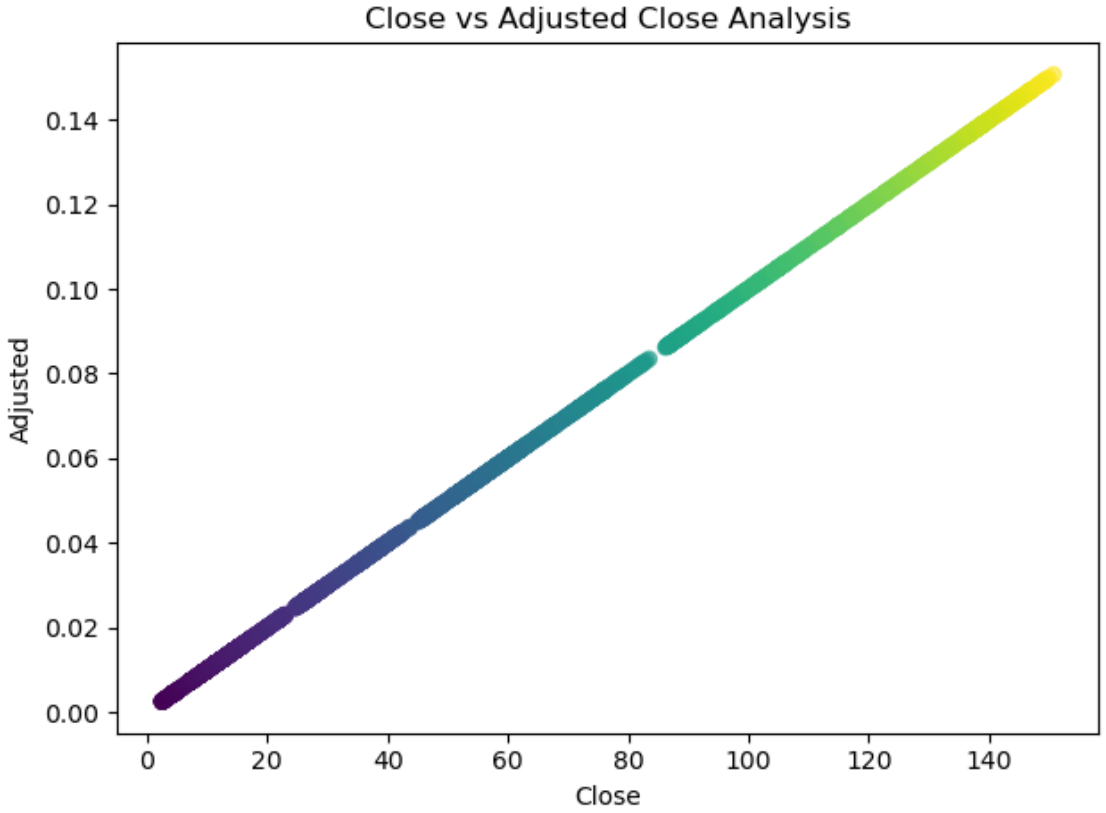
[5]:
```
df.isna().sum()
```

[5]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

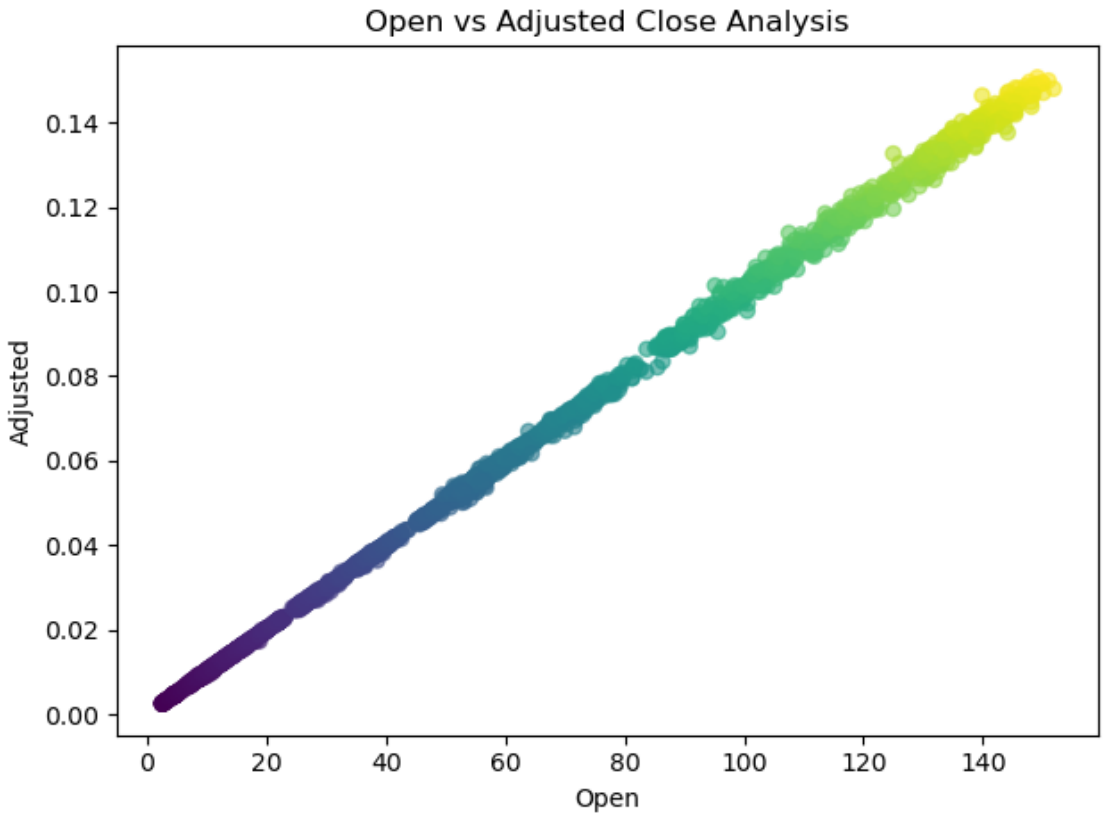
[6]:
```
plt.scatter(df["Open"], df["Volume"]/1000, s=35, alpha=0.6,
            c=df["Volume"]/1000)
plt.title("Open vs Volume Analysis")
plt.xlabel("Open")
plt.ylabel("Volume")
plt.tight_layout()
plt.show()
```



[7]:
```
plt.scatter(df["Close"], df["Adj Close"]/1000, s=35, alpha=0.6,
            c=df["Adj Close"]/1000)
plt.title("Close vs Adjusted Close Analysis")
plt.xlabel("Close")
plt.ylabel("Adjusted")
plt.tight_layout()
plt.show()
```



[8]:
```
plt.scatter(df["Open"], df["Adj Close"]/1000, s=35, alpha=0.6,
            c=df["Adj Close"]/1000)
plt.title("Open vs Adjusted Close Analysis")
plt.xlabel("Open")
plt.ylabel("Adjusted")
plt.tight_layout()
plt.show()
```



[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4858 entries, 0 to 4857
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       4858 non-null   object
 1   Open       4858 non-null   float64
 2   High       4858 non-null   float64
 3   Low        4858 non-null   float64
 4   Close      4858 non-null   float64
 5   Adj Close  4858 non-null   float64
 6   Volume     4858 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 265.8+ KB
```

[10]:
```
df.describe()
```

[10]:
```
              Open         High          Low        Close    Adj Close  \
count  4858.000000  4858.000000  4858.000000  4858.000000  4858.000000
```

```
mean     41.477174    41.917059    41.055491    41.494404    41.494404
std      38.590695    39.031758    38.193016    38.618107    38.618107
min       2.470490     2.534002     2.390042     2.490913     2.490913
25%      12.846597    12.954195    12.712414    12.834642    12.834642
50%      26.499958    26.728268    26.289323    26.537501    26.537501
75%      57.367250    58.028500    56.962251    57.611249    57.611249
max     151.863495   152.100006   149.887497   150.709000   150.709000

              Volume
count   4.858000e+03
mean    1.189152e+08
std     1.512424e+08
min     1.584340e+05
25%     2.854912e+07
50%     6.168836e+07
75%     1.467329e+08
max     1.650833e+09
```

[11]:
```
subset = df[["Open", "High", "Low", "Close", "Adj Close", "Volume"]]

subset.head(1)
```

[11]:
```
       Open      High       Low     Close  Adj Close     Volume
0  2.490664  2.591785  2.390042  2.499133   2.499133  897427216
```

[12]:
```
print(subset.corr())
```

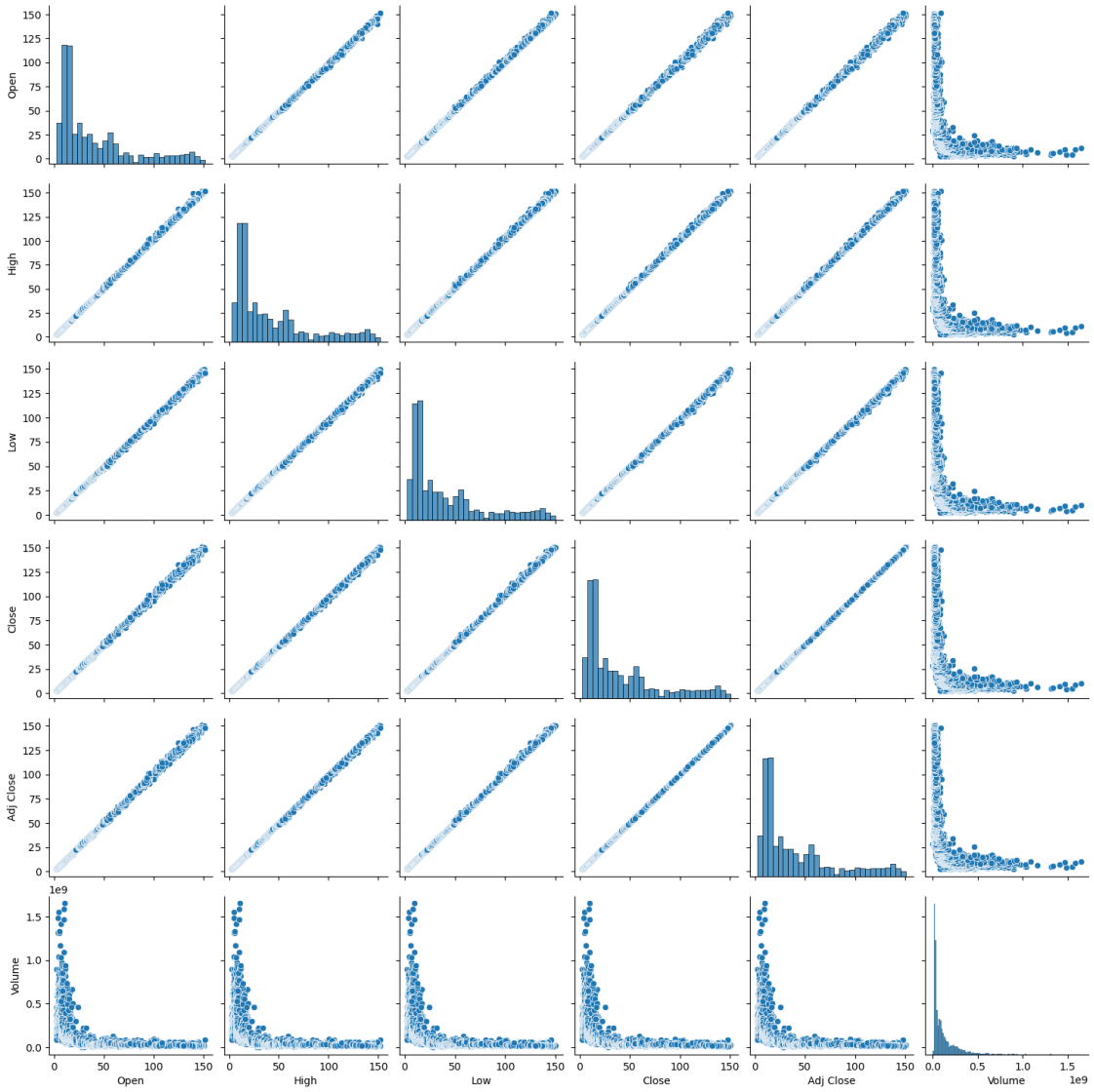
```
                Open      High       Low     Close  Adj Close    Volume
Open        1.000000  0.999903  0.999896  0.999774   0.999774 -0.471746
High        0.999903  1.000000  0.999875  0.999894   0.999894 -0.470766
Low         0.999896  0.999875  1.000000  0.999904   0.999904 -0.473027
Close       0.999774  0.999894  0.999904  1.000000   1.000000 -0.472017
Adj Close   0.999774  0.999894  0.999904  1.000000   1.000000 -0.472017
Volume     -0.471746 -0.470766 -0.473027 -0.472017  -0.472017  1.000000
```

[13]:
```
sns.pairplot(subset)
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
```

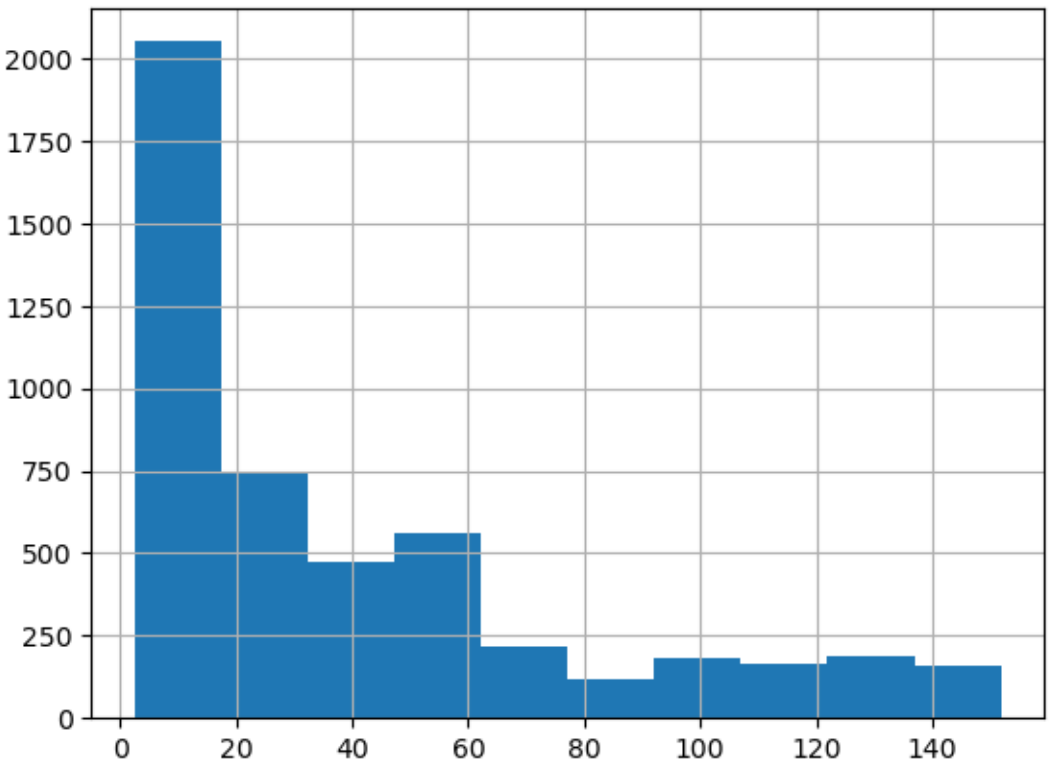
```
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
[13]: <seaborn.axisgrid.PairGrid at 0x739c052b25f0>
```



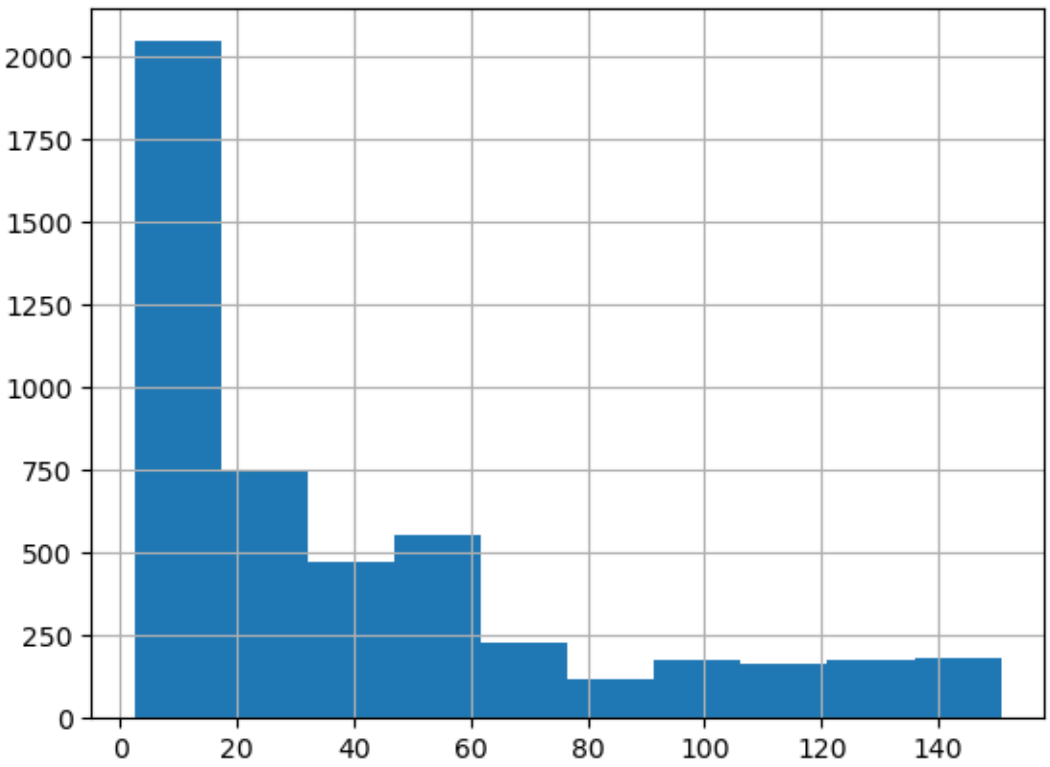
[14]:
```
df["Open"].hist()
```

[14]: <Axes: >



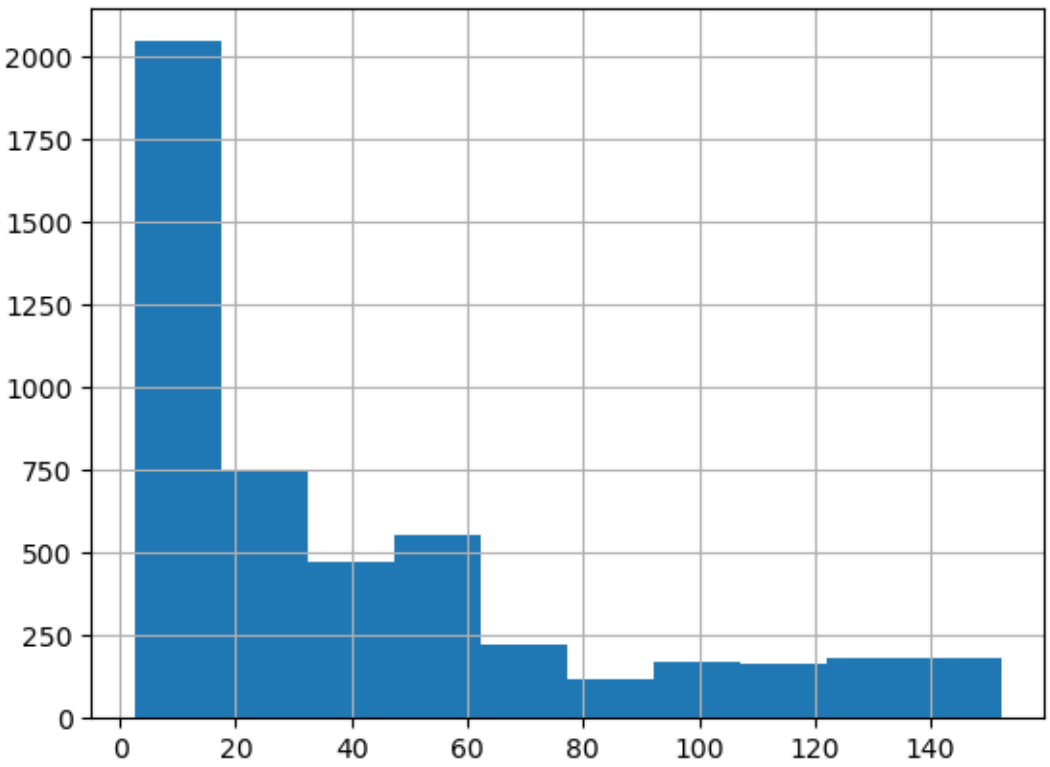
[15]:
```
df["Adj Close"].hist()
```

[15]: <Axes: >



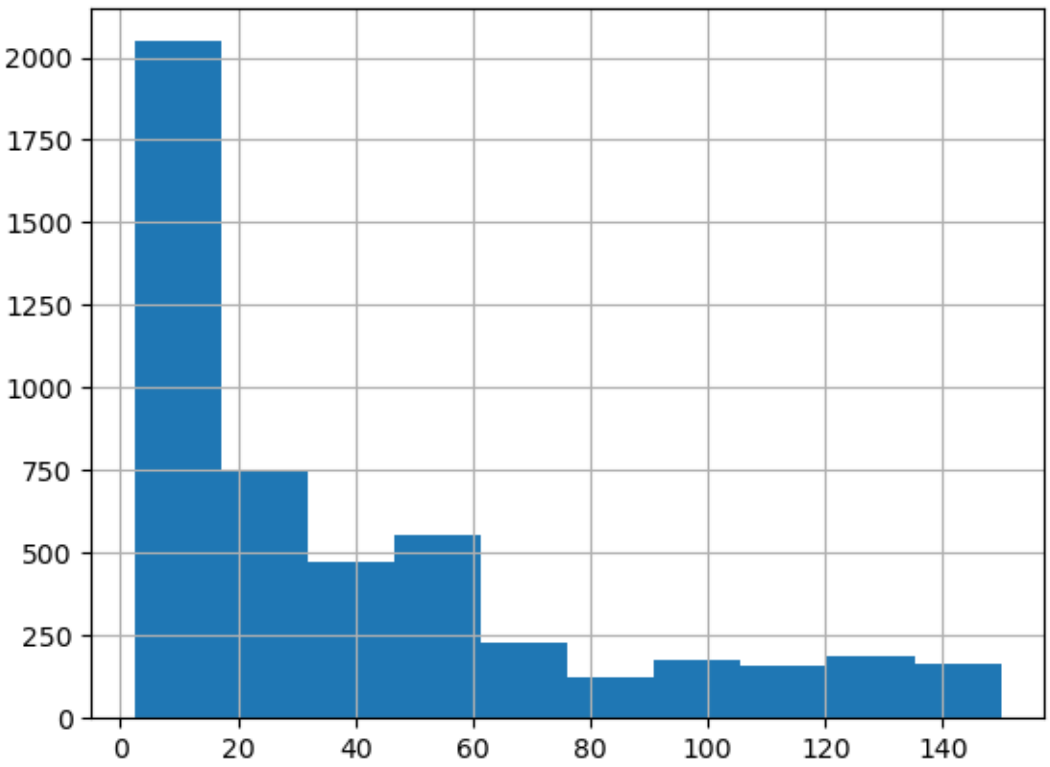
[16]:
```
df["High"].hist()
```

[16]: <Axes: >



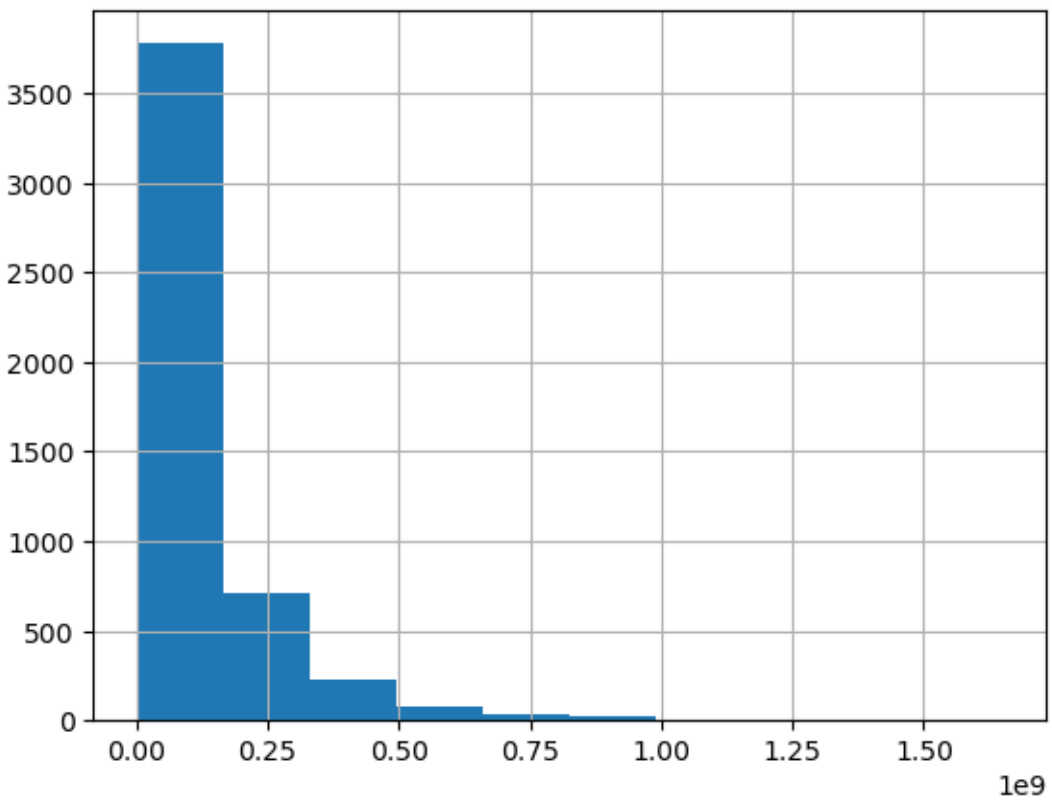
[17]:
```
df["Low"].hist()
```

[17]: <Axes: >

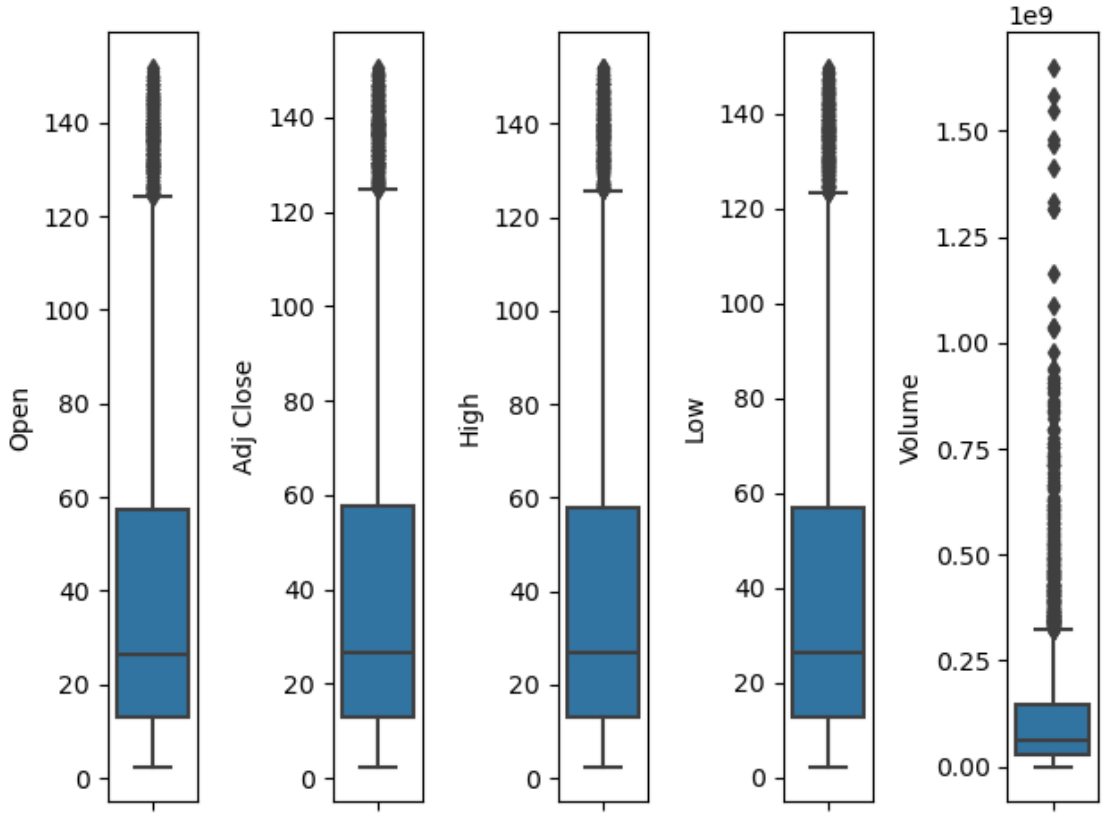


[18]:
```
df["Volume"].hist()
```

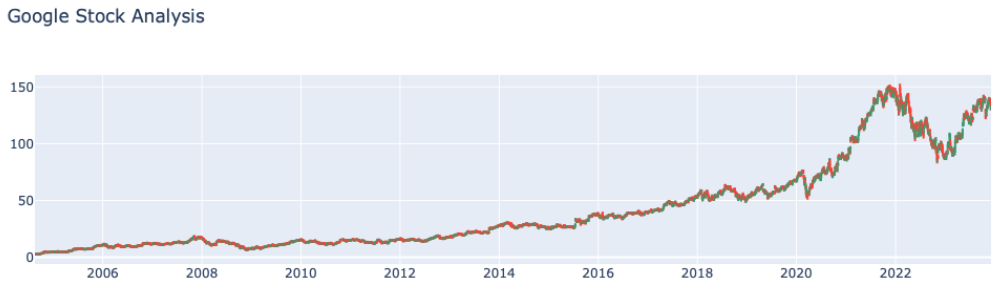
[18]: <Axes: >



[19]:
```
f, axes = plt.subplots(1,5)
sns.boxplot(y='Open', data=df, ax = axes[0])
sns.boxplot(y='Adj Close', data=df, ax = axes[1])
sns.boxplot(y='High', data=df, ax = axes[2])
sns.boxplot(y='Low', data=df, ax = axes[3])
sns.boxplot(y='Volume', data=df, ax = axes[4])
plt.tight_layout()
```



[20]:
```
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 Analysis", xaxis_rangeslider_visible␣
↪= False)
figure.show()
```



[21]:
```
X = df[['Open', 'High', 'Low', 'Volume']].values
y = df['Close'].values
```

[22]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 0)
```

[23]:
```
print('Train Shape:', X_train.shape)
print('Test Shape:', X_test.shape)
```

```
Train Shape: (3886, 4)
Test Shape: (972, 4)
```

[24]:
```
regressor = LinearRegression()

model = regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

prediction = regressor.predict(X_test)

prediction.shape
```

[24]:
```
(972,)
```

[25]:
```
print('Model Coefficients :', regressor.coef_)

print('Model Intercept :', regressor.intercept_)
```

```
Model Coefficients : [-6.10253628e-01  8.08129217e-01  8.01937121e-01
-1.28333383e-11]
Model Intercept : 0.01228491397410636
```

[26]:
```
data_frame = pd.DataFrame(y_test, prediction)

frame = pd.DataFrame({'Actual_Price': y_test, 'Predicted_Price': prediction})

print(frame)
```

```
     Actual_Price  Predicted_Price
0       10.185572        10.164893
1      119.306000       118.732552
2       13.331779        13.292533
3       51.078499        51.978363
4       15.568645        15.534941
..            …               …
```

```
967     58.705002        58.648101
968     75.899002        75.904647
969     21.578867        21.696195
970     14.583338        14.599901
971     11.883458        11.925920

[972 rows x 2 columns]
```

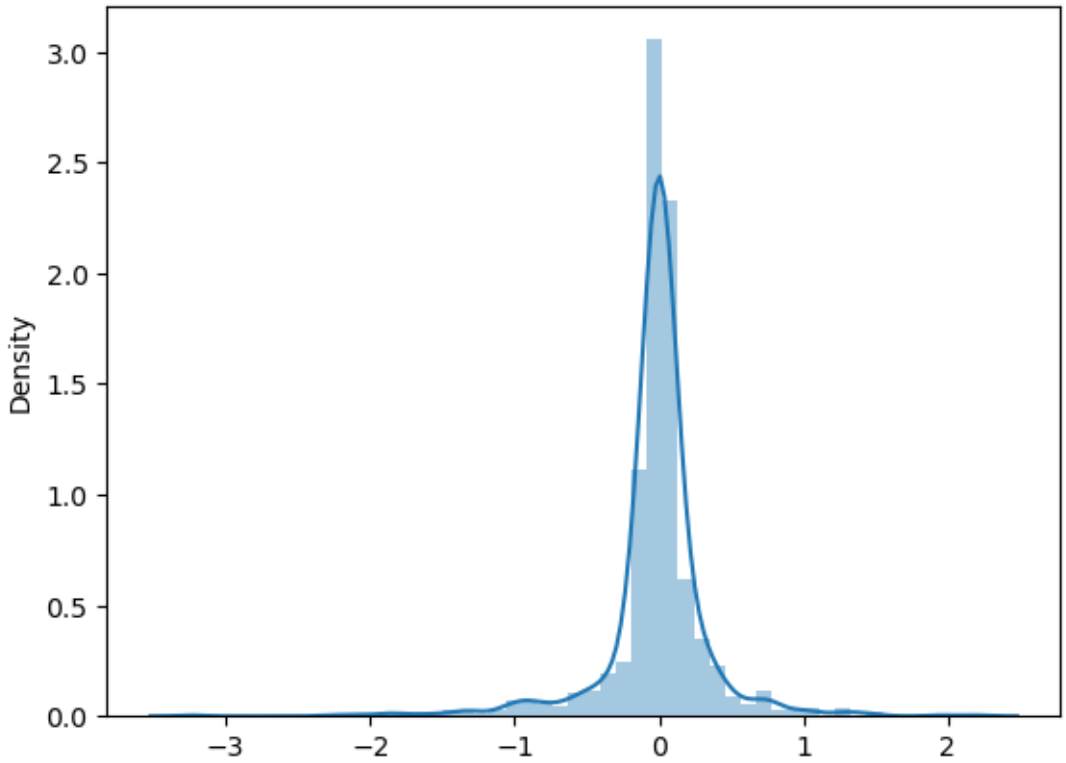
[27]:
```
frame.describe()
```

[27]:
```
       Actual_Price  Predicted_Price
count    972.000000       972.000000
mean      41.437752        41.453152
std       39.220179        39.248666
min        2.496891         2.512276
25%       12.662538        12.685805
50%       26.190500        26.206564
75%       56.235126        56.139368
max      150.709000       151.002961
```

[28]:
```
residual = y_test - prediction

sns.distplot(residual)
```

```
/tmp/ipykernel_495/2079149211.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


/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
```

[28]:
```
<Axes: ylabel='Density'>
```



[29]:
```
p_value = scipy.stats.norm.sf(abs(1.67))

print('The P_Value is:' + str(p_value))
```

```
The P_Value is:0.04745968180294733
```

[30]:
```
test_results = sm.OLS(y_test, X_test).fit()

test_results.summary()
```

[30]:

Dep. Variable:	y	**R-squared (uncentered):**	1.000
Model:	OLS	**Adj. R-squared (uncentered):**	1.000
Method:	Least Squares	**F-statistic:**	6.172e+06
Date:	Sun, 20 Oct 2024	**Prob (F-statistic):**	0.00
Time:	00:19:42	**Log-Likelihood:**	-378.47
No. Observations:	972	**AIC:**	764.9
Df Residuals:	968	**BIC:**	784.5
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
x1	-0.6295	0.026	-23.780	0.000	-0.681	-0.578
x2	0.6513	0.027	24.558	0.000	0.599	0.703
x3	0.9811	0.021	45.669	0.000	0.939	1.023
x4	5.977e-11	6e-11	0.996	0.320	-5.8e-11	1.78e-10

Omnibus:	217.571	**Durbin-Watson:**	2.057
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6671.725
Skew:	0.254	**Prob(JB):**	0.00
Kurtosis:	15.825	**Cond. No.**	5.85e+08

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, 5.85e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

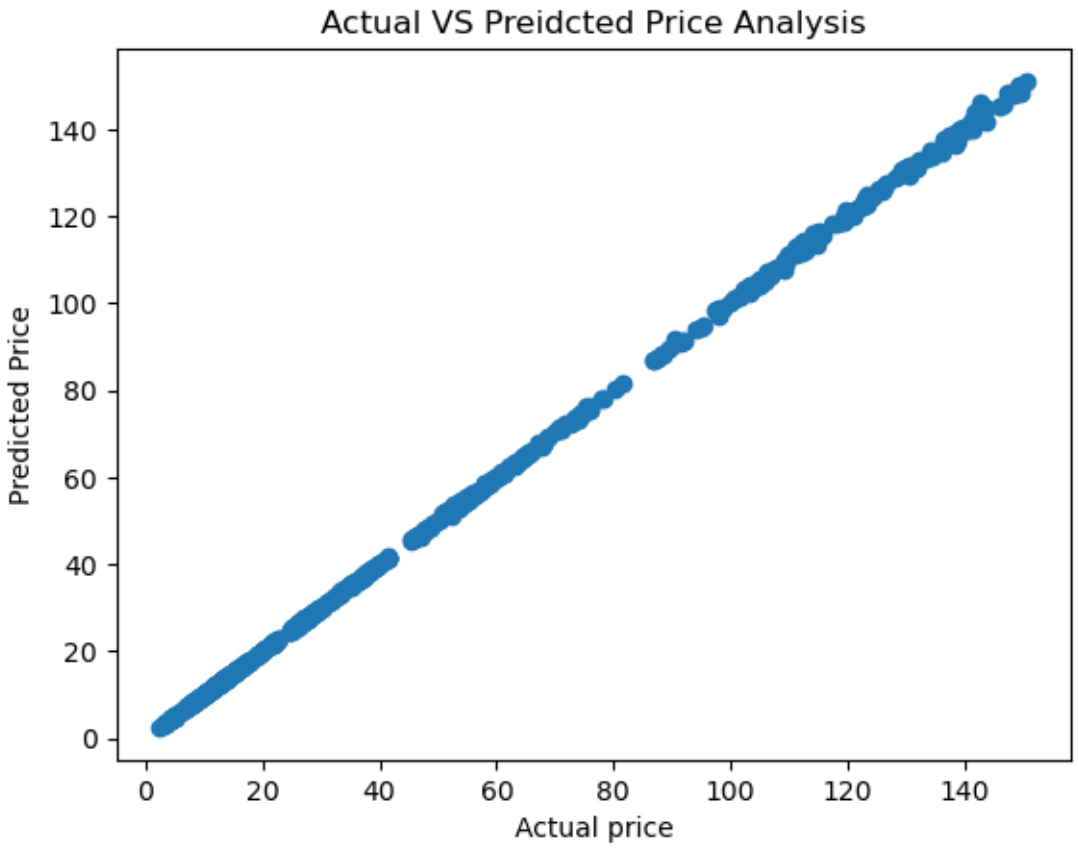
[31]:
```
regression_confidence = regressor.score(X_test, y_test)

print('Linear Regression Confidence Score: ', regression_confidence)
```

```
Linear Regression Confidence Score:  0.9999098437237599
```

[32]:
```
x2 = abs(prediction - y_test)
y2 = 100*(x2/y_test)
accuracy = 100 - np.mean(y2)

print('Accuracy:', round(accuracy, 2), '%.')
```

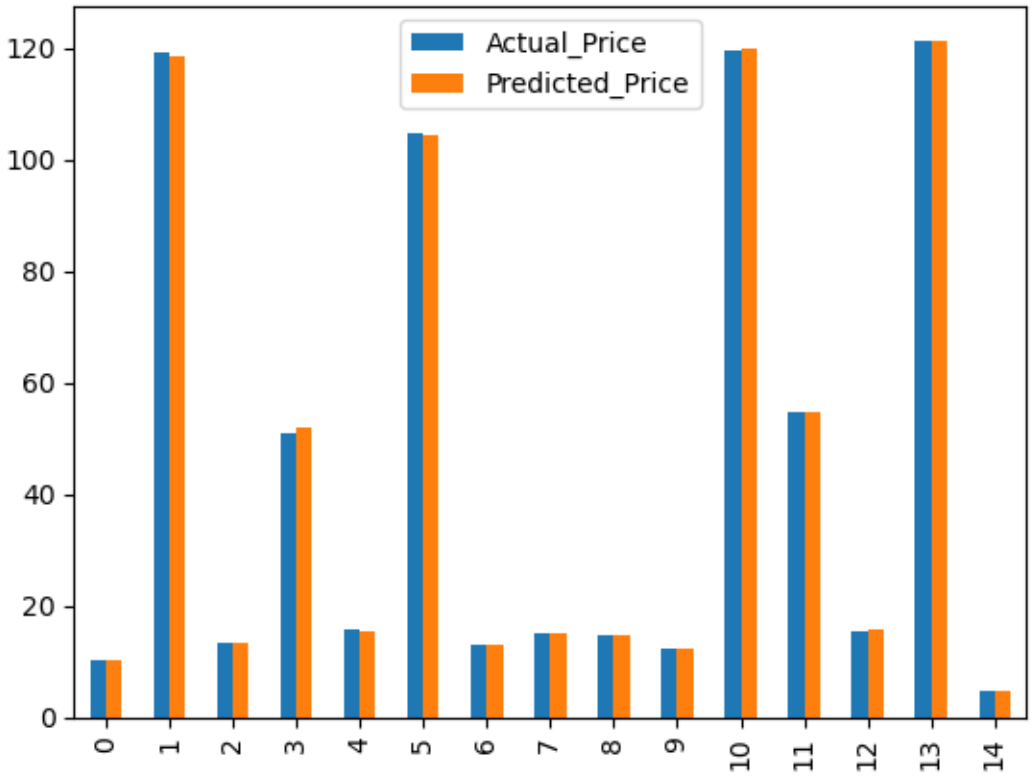
```
Accuracy: 99.55 %.
```

[71]:
```
plt.scatter(frame.Actual_Price, frame.Predicted_Price)
plt.title('Actual VS Preidcted Price Analysis')
plt.xlabel('Actual price')
plt.ylabel('Predicted Price')
plt.show()
```



[73]:
```
graph = frame.head(15)
graph.plot(kind = 'bar')
```

[73]: <Axes: >


Actual_Price
Predicted_Price
120
100
80
60
40
20
0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14