hw2

February 27, 2024

1 Annaly Rocha

Z23695925

CAP4773

HW2

[ ]:
```
# Section 1
```

[116]:
```
num_predictors = data.shape[1]
print(f'The number of predictors in the dataset is: {num_predictors}')
```

```
The number of predictors in the dataset is: 7
```

1.1 1.1 Dataset: New York Stock Exchange - 7 Predictors

1.2 Description of the dataset

This dataset is a playground for fundamental and technical analysis. It is said that 30% of traffic on stocks is already generated by machines, can trading be fully automated? If not, there is still a lot to learn from historical data.

using: prices.csv: raw, as-is daily prices. Most of data spans from 2010 to the end 2016, for companies new on stock market date range is shorter. There have been approx. 140 stock splits in that time, this set doesn't account for that.

https://www.kaggle.com/ Search -> New York Stock Exchange -> Download

https://www.kaggle.com/datasets/dgawlik/nyse?resource=download

[74]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.stats.outliers_influence as inf
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('prices.csv')
```

[76]:
```
data.columns
```

[76]:
```
Index(['date', 'symbol', 'open', 'close', 'low', 'high', 'volume'],
dtype='object')
```

[78]:
```
data = data. dropna(subset=['date', 'symbol', 'open', 'close', 'low', 'high',␣
↪'volume'])
```

[80]:
```
data.head
```

[80]:
```
<bound method NDFrame.head of                        date symbol        open
close         low  \
0       2016-01-05 00:00:00   WLTW  123.430000  125.839996  122.309998
1       2016-01-06 00:00:00   WLTW  125.239998  119.980003  119.940002
2       2016-01-07 00:00:00   WLTW  116.379997  114.949997  114.930000
3       2016-01-08 00:00:00   WLTW  115.480003  116.620003  113.500000
4       2016-01-11 00:00:00   WLTW  117.010002  114.970001  114.089996
…                         …    …         …           …           …
851259           2016-12-30    ZBH  103.309998  103.199997  102.849998
851260           2016-12-30   ZION   43.070000   43.040001   42.689999
851261           2016-12-30    ZTS   53.639999   53.529999   53.270000
851262  2016-12-30 00:00:00    AIV   44.730000   45.450001   44.410000
851263  2016-12-30 00:00:00    FTV   54.200001   53.630001   53.389999

              high     volume
0       126.250000  2163600.0
1       125.540001  2386400.0
2       119.739998  2489500.0
3       117.440002  2006300.0
4       117.330002  1408600.0
…              …          …
851259  103.930000   973800.0
851260   43.310001  1938100.0
851261   53.740002  1701200.0
851262   45.590000  1380900.0
851263   54.480000   705100.0

[851264 rows x 7 columns]>
```

[82]:
```
data.info
```

[82]:
```
<bound method DataFrame.info of                        date symbol        open
close         low  \
0       2016-01-05 00:00:00   WLTW  123.430000  125.839996  122.309998
1       2016-01-06 00:00:00   WLTW  125.239998  119.980003  119.940002
2       2016-01-07 00:00:00   WLTW  116.379997  114.949997  114.930000
3       2016-01-08 00:00:00   WLTW  115.480003  116.620003  113.500000
4       2016-01-11 00:00:00   WLTW  117.010002  114.970001  114.089996
…                         …    …         …           …           …
```

```
851259           2016-12-30    ZBH  103.309998  103.199997  102.849998
851260           2016-12-30   ZION   43.070000   43.040001   42.689999
851261           2016-12-30    ZTS   53.639999   53.529999   53.270000
851262  2016-12-30 00:00:00    AIV   44.730000   45.450001   44.410000
851263  2016-12-30 00:00:00    FTV   54.200001   53.630001   53.389999

              high     volume
0       126.250000  2163600.0
1       125.540001  2386400.0
2       119.739998  2489500.0
3       117.440002  2006300.0
4       117.330002  1408600.0
…              …         …
851259  103.930000   973800.0
851260   43.310001  1938100.0
851261   53.740002  1701200.0
851262   45.590000  1380900.0
851263   54.480000   705100.0

[851264 rows x 7 columns]>
```

[84]:

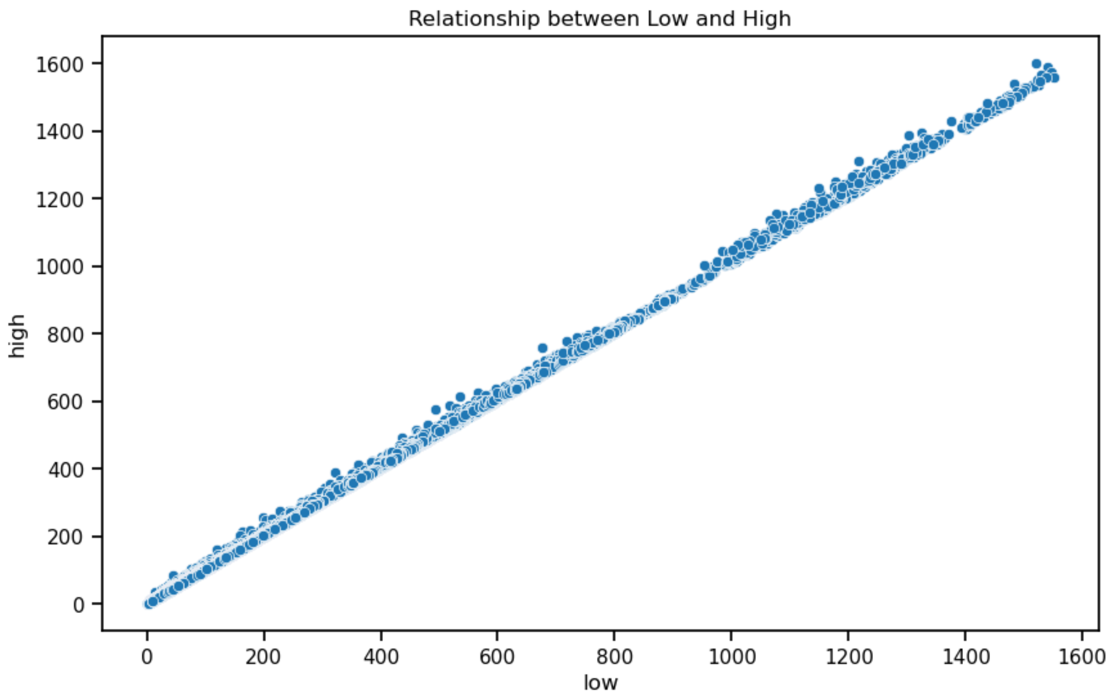
```python
import pandas as pd

# Assuming you have a DataFrame named 'df' with the given data
df_data = {
    'date': ['2016-01-05', '2016-01-06', '2016-01-07', '2016-01-08',
 ↪'2016-01-11'],
    'symbol': ['WLTW', 'WLTW', 'WLTW', 'WLTW', 'WLTW'],
    'open': [123.43, 125.24, 116.38, 115.48, 117.01],
    'close': [125.84, 119.98, 114.95, 116.62, 114.97],
    'low': [122.31, 119.94, 114.93, 113.5, 114.09],
    'high': [126.25, 125.54, 119.74, 117.44, 117.33],
    'volume': [2163600.0, 2386400.0, 2489500.0, 2006300.0, 1408600.0]
}

df = pd.DataFrame(df_data)
data_points = df.to_dict(orient='records')
print(data_points)
```

```
[{'date': '2016-01-05', 'symbol': 'WLTW', 'open': 123.43, 'close': 125.84,
'low': 122.31, 'high': 126.25, 'volume': 2163600.0}, {'date': '2016-01-06',
'symbol': 'WLTW', 'open': 125.24, 'close': 119.98, 'low': 119.94, 'high':
125.54, 'volume': 2386400.0}, {'date': '2016-01-07', 'symbol': 'WLTW', 'open':
116.38, 'close': 114.95, 'low': 114.93, 'high': 119.74, 'volume': 2489500.0},
{'date': '2016-01-08', 'symbol': 'WLTW', 'open': 115.48, 'close': 116.62, 'low':
113.5, 'high': 117.44, 'volume': 2006300.0}, {'date': '2016-01-11', 'symbol':
'WLTW', 'open': 117.01, 'close': 114.97, 'low': 114.09, 'high': 117.33,
```

```
'volume': 1408600.0}]
```

[86]:
```
# 2.1 Select a numerical column as your target (Y).
# 2.2 Choose one predictor (X).
X = data['low']
y = data['high']
```

[88]:
```
# 2.3 Visualize the relationship with a scatter plot.
# Plotting the data
plt.figure(figsize=(10,6))
sns.scatterplot(x='low', y='high', data=data)
plt.title('Relationship between Low and High')
plt.show()
```



[90]:
```
# 2.4 # Fiting a Linear Model
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
# Printing a summary
model.summary()
```

[90]:

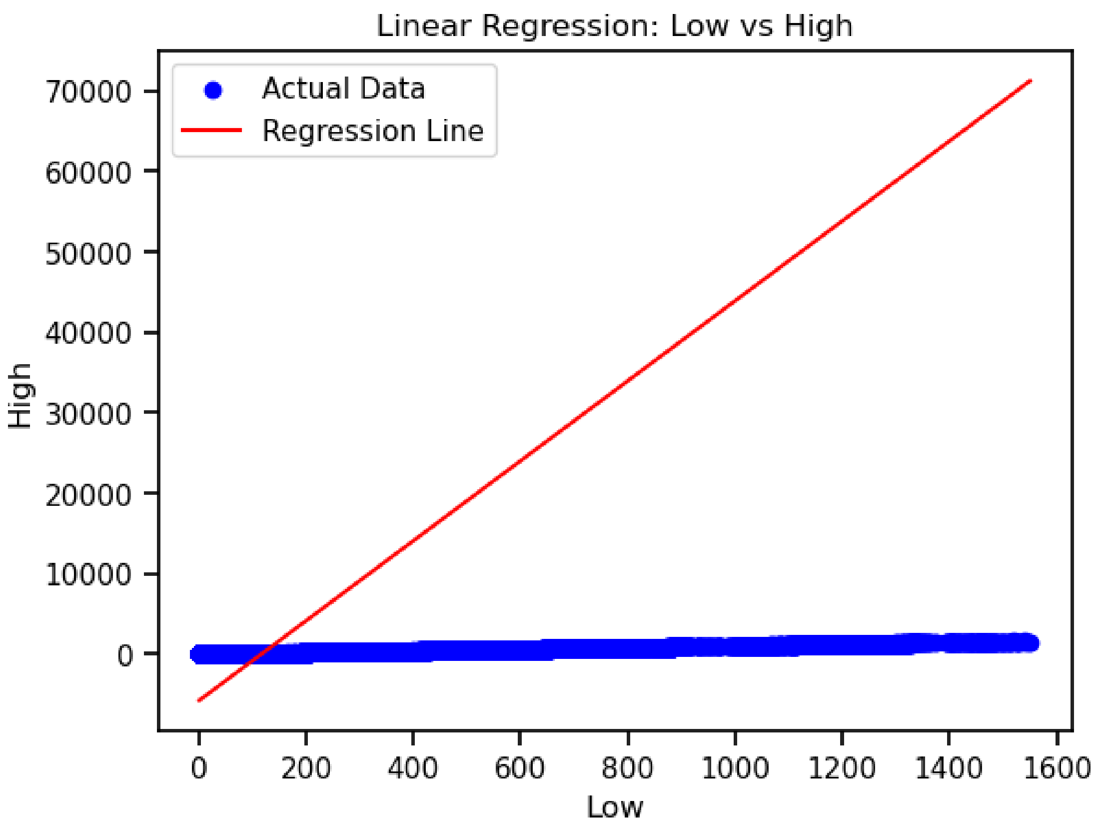
Dep. Variable:	high	**R-squared:**	1.000
Model:	OLS	**Adj. R-squared:**	1.000
Method:	Least Squares	**F-statistic:**	3.958e+09
Date:	Tue, 27 Feb 2024	**Prob (F-statistic):**	0.00
Time:	03:52:40	**Log-Likelihood:**	-1.3900e+06
No. Observations:	851264	**AIC:**	2.780e+06
Df Residuals:	851262	**BIC:**	2.780e+06
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	0.0890	0.002	50.635	0.000	0.086	0.092
low	1.0191	1.62e-05	6.29e+04	0.000	1.019	1.019

Omnibus:	1242450.041	**Durbin-Watson:**	1.671
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2149484359.660
Skew:	8.256	**Prob(JB):**	0.00
Kurtosis:	248.619	**Cond. No.**	142.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

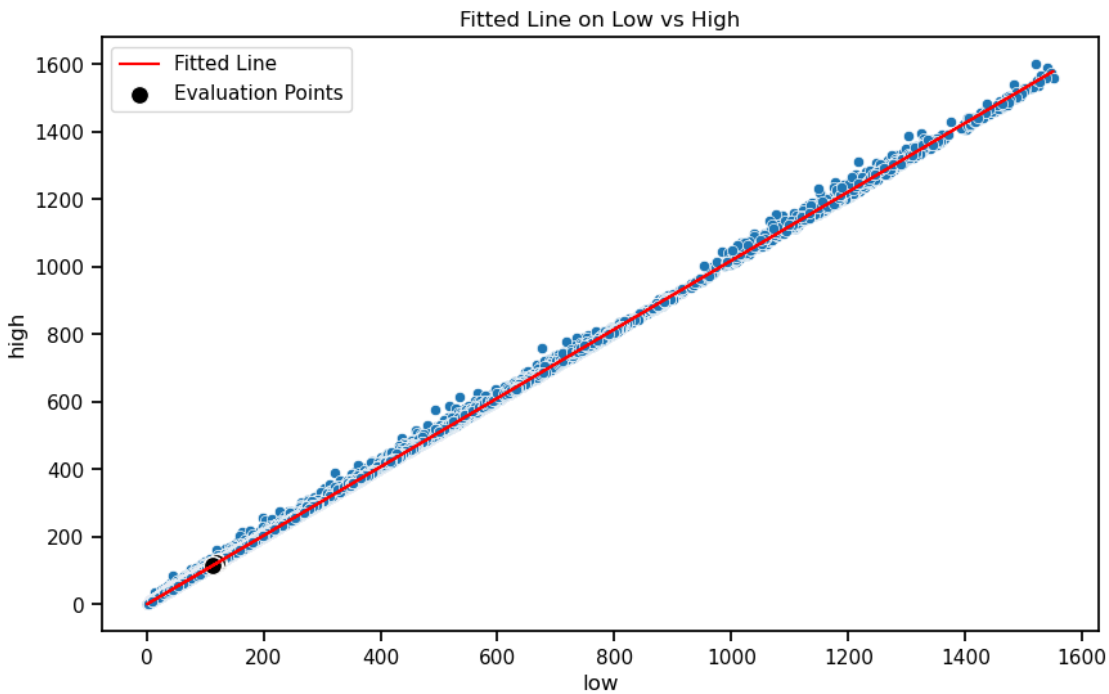
[127]:
```
# 2.6 - Equation of the model
# high = -5780.8314 + 49.6856 x

intercept = -5780.8314
slope = 49.6856

# Generate some example data
low_values = np.linspace(min(data['low']), max(data['low']), 100)
predicted_high_values = intercept + slope * low_values

# Plotting the data points
plt.scatter(data['low'], data['high'], label='Actual Data', color='blue')

# Plotting the regression line
plt.plot(low_values, predicted_high_values, label='Regression Line',␣
 ↪color='red')

# Adding labels and legend
plt.xlabel('Low')
plt.ylabel('High')
plt.title('Linear Regression: Low vs High')
plt.legend()

# Show the plot
plt.show()
```



[92]:

```
# 2.7 Predict the target value for the evaluation set and overlay these on the␣
↪scatter plot with a different color
#or marker.
# 2.8 Add the regression line to the scatter plot.
# Ploting the regression line and evaluation points

# Get a list of the evaluation points for lstat
low_values = []
for point in data_points:
    low_values.append(point['low'])

# Least squares coefficients
beta_1 = model.params['low']
beta_0 = model.params['const']

# Extracting high values using the regression equation for these low_values
high_values = beta_0 + beta_1 * np.array(low_values)

# Original plot with scatter points and regression line
plt.figure(figsize=(10, 6))
```

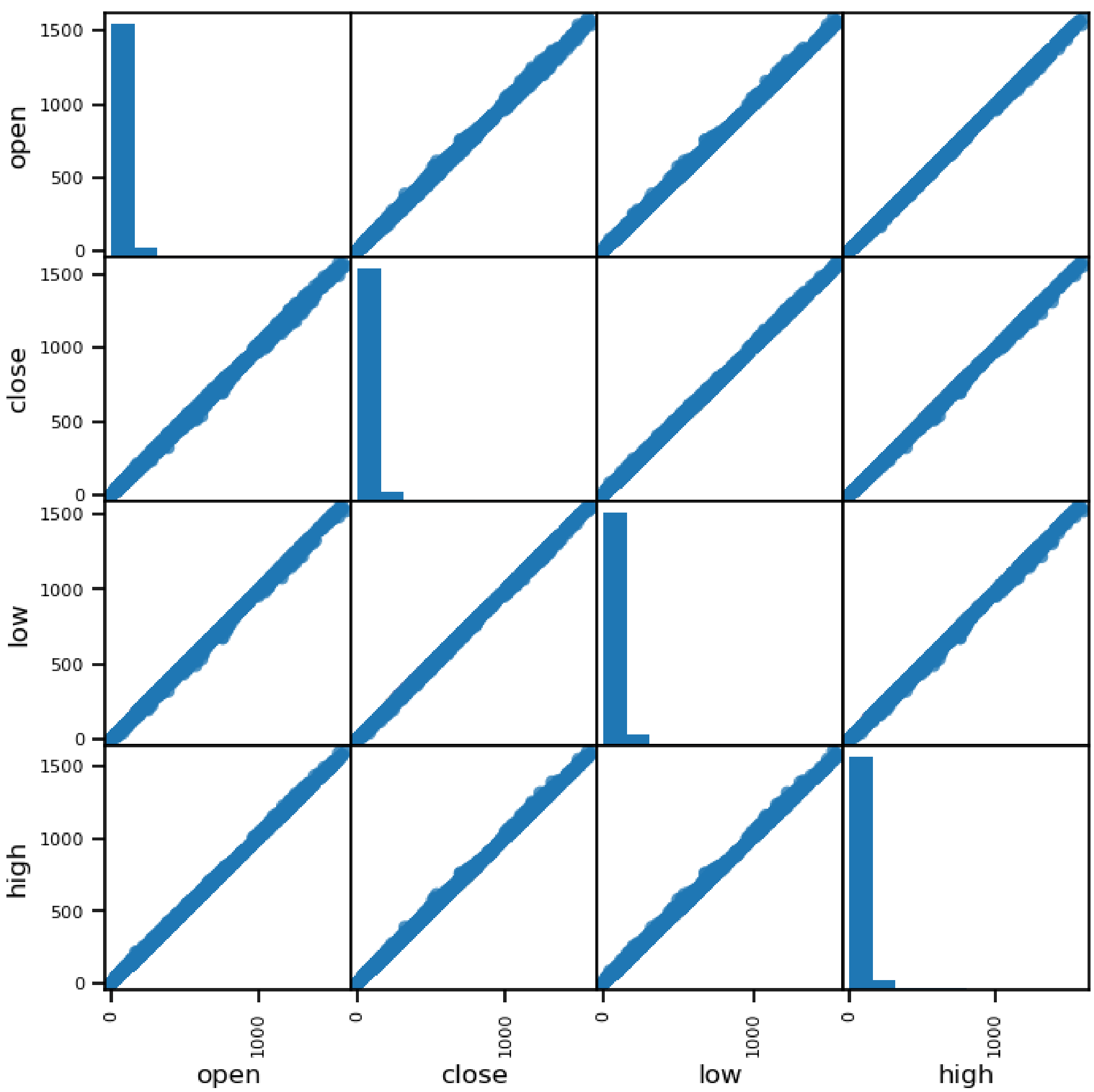
```
sns.scatterplot(x=data['low'], y=data['high'])
plt.plot(data['low'], beta_0 + beta_1 * data['low'], color='red', label="Fitted␣
↪Line")

# Adding the points from the dictionary with a different color and size
sns.scatterplot(x=low_values, y=high_values, color='black', s=100,␣
↪label="Evaluation Points")

# Title and legend
plt.title('Fitted Line on Low vs High')
plt.legend()
plt.show()
```



[94]:
```
# Section 3
#3.1 Choose 3 predictors and create a subset of your dataset.
selected_predictors =['open','close','low','high']

#3.2 Visualize the relationships using a scatter plot matrix.
X = data[selected_predictors]
pd.plotting.scatter_matrix(X, figsize=(8, 8), alpha=0.8, marker='o',␣
↪diagonal='hist')
```

[94]:
```
array([[<Axes: xlabel='open', ylabel='open'>,
        <Axes: xlabel='close', ylabel='open'>,
```

```
 <Axes: xlabel='low', ylabel='open'>,
 <Axes: xlabel='high', ylabel='open'>],
[<Axes: xlabel='open', ylabel='close'>,
 <Axes: xlabel='close', ylabel='close'>,
 <Axes: xlabel='low', ylabel='close'>,
 <Axes: xlabel='high', ylabel='close'>],
[<Axes: xlabel='open', ylabel='low'>,
 <Axes: xlabel='close', ylabel='low'>,
 <Axes: xlabel='low', ylabel='low'>,
 <Axes: xlabel='high', ylabel='low'>],
[<Axes: xlabel='open', ylabel='high'>,
 <Axes: xlabel='close', ylabel='high'>,
 <Axes: xlabel='low', ylabel='high'>,
 <Axes: xlabel='high', ylabel='high'>]], dtype=object)
```



[129]:
```
#3.3 Based on visual inspection, note any predictors that appear to be␣
↪correlated.
# <there seems to be some relationship.
#3.4 Calculate the VIF (Variance Inflation Factor) for these predictors.
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Subset of the dataset with selected predictors
X = data[selected_predictors]

# Calculate VIF for each predictor
for i in range(X.shape[1]):
    vif = variance_inflation_factor(X.values, i)
    print(f"VIF for {X.columns[i]}: \t{vif:10.3f}")
```

```
VIF for open:    26369.991
VIF for close:   29024.136
VIF for low:     27338.362
VIF for high:    32794.691
```

[104]:
```
#3.5 Discuss which predictors show high multicollinearity based on the VIF␣
↪values.
#All of these VIF values are exceptionally high, well above the common␣
↪threshold of 10. This suggests severe multicollinearity among the predictors.
↪
```

[106]:
```
# Section 4 - Multiple Regresion Model
# 4.1 Choose at least additional 3 predictors which seem relevant to the target.
X = pd.DataFrame(data[['open','close','low']])
y = data['high']
# 4.2 Fit a linear model with the new predictors using statsmodels.api.
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
# 4.3 Print the model summary and analyze it. Compare the R^2 value with the␣
↪simple linear regression.
model.summary()
```

[106]:

Dep. Variable:	high	**R-squared:**	1.000
Model:	OLS	**Adj. R-squared:**	1.000
Method:	Least Squares	**F-statistic:**	5.423e+09
Date:	Tue, 27 Feb 2024	**Prob (F-statistic):**	0.00
Time:	04:02:54	**Log-Likelihood:**	-7.8843e+05
No. Observations:	851264	**AIC:**	1.577e+06
Df Residuals:	851260	**BIC:**	1.577e+06
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
const	0.0253	0.001	29.125	0.000	0.024	0.027
open	0.7092	0.001	1164.575	0.000	0.708	0.710
close	0.7274	0.001	1100.483	0.000	0.726	0.729
low	-0.4316	0.001	-482.892	0.000	-0.433	-0.430

Omnibus:	929334.332	**Durbin-Watson:**	1.711
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1023876685.133
Skew:	4.663	**Prob(JB):**	0.00
Kurtosis:	172.645	**Cond. No.**	313.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[140]:

```
# 4.4 Use the coefficients to write the equation for the model.
# high = -6424.7 + 4.16 x open + 20.04 x close + 50.26 x low
# Assuming these are the coefficients from your model
intercept = -6424.7
coef_open = 4.16
coef_close = 20.04
coef_low = 50.26

# Sample values for predictors
open_value = 10.0
close_value = 25.0
low_value = 30.0

# Calculate the predicted high value using the equation
predicted_high = intercept + coef_open * open_value + coef_close * close_value␣
↪+ coef_low * low_value

# Display the equation
equation = f"high = {intercept:.2f} + {coef_open:.2f} * open + {coef_close:.2f}␣
↪* close + {coef_low:.2f} * low"
print(equation)

# Display the calculated result
print(f"For open={open_value}, close={close_value}, low={low_value}, the␣
↪predicted high is: {predicted_high:.2f}")
```

```
high = -6424.70 + 4.16 * open + 20.04 * close + 50.26 * low
For open=10.0, close=25.0, low=30.0, the predicted high is: -4374.30
```

[138]:

```
# 4.5 Predict the target values for the evaluation set based on this new model
# Get lists of the evaluation points for all predictors
open_values = []
close_values = []
low_values = []
```

```python
for point in data_points:
    open_values.append(point['open'])
    close_values.append(point['close'])
    low_values.append(point['low'])
# Extracting coefficients from the model
beta_0 = model.params['const']
beta_1 = model.params['open']
beta_2 = model.params['close']
beta_3 = model.params['low']
# Calculating medv values using the multiple regression equation
mass_values = (beta_0 +
beta_1 * np.array(open_values) +
beta_2 * np.array(close_values) +
beta_3 * np.array(low_values))

# Printing the medv values along with the predictor values

for i, (open, close, low, high) in enumerate(zip(open_values, close_values,␣
 ↪low_values, high_values)):
    print(f"Data Point {i+1} - open: {open:.2f}, close: {close:.2f}, low: {low:.
 ↪2f}, Estimated high: {high:.2f}")
```

```
Data Point 1 - open: 123.43, close: 125.84, low: 122.31, Estimated high: 124.73
Data Point 2 - open: 125.24, close: 119.98, low: 119.94, Estimated high: 122.31
Data Point 3 - open: 116.38, close: 114.95, low: 114.93, Estimated high: 117.21
Data Point 4 - open: 115.48, close: 116.62, low: 113.50, Estimated high: 115.75
Data Point 5 - open: 117.01, close: 114.97, low: 114.09, Estimated high: 116.35
```