Multiple Linear Regression - Cumulative Lab

Introduction

In this cumulative lab you'll perform an end-to-end analysis of a dataset using multiple linear regression.

Objectives

You will be able to:

- Prepare data for regression analysis using pandas
- Build multiple linear regression models using StatsModels
- Measure regression model performance
- Interpret multiple linear regression coefficients

Your Task: Develop a Model of Diamond Prices

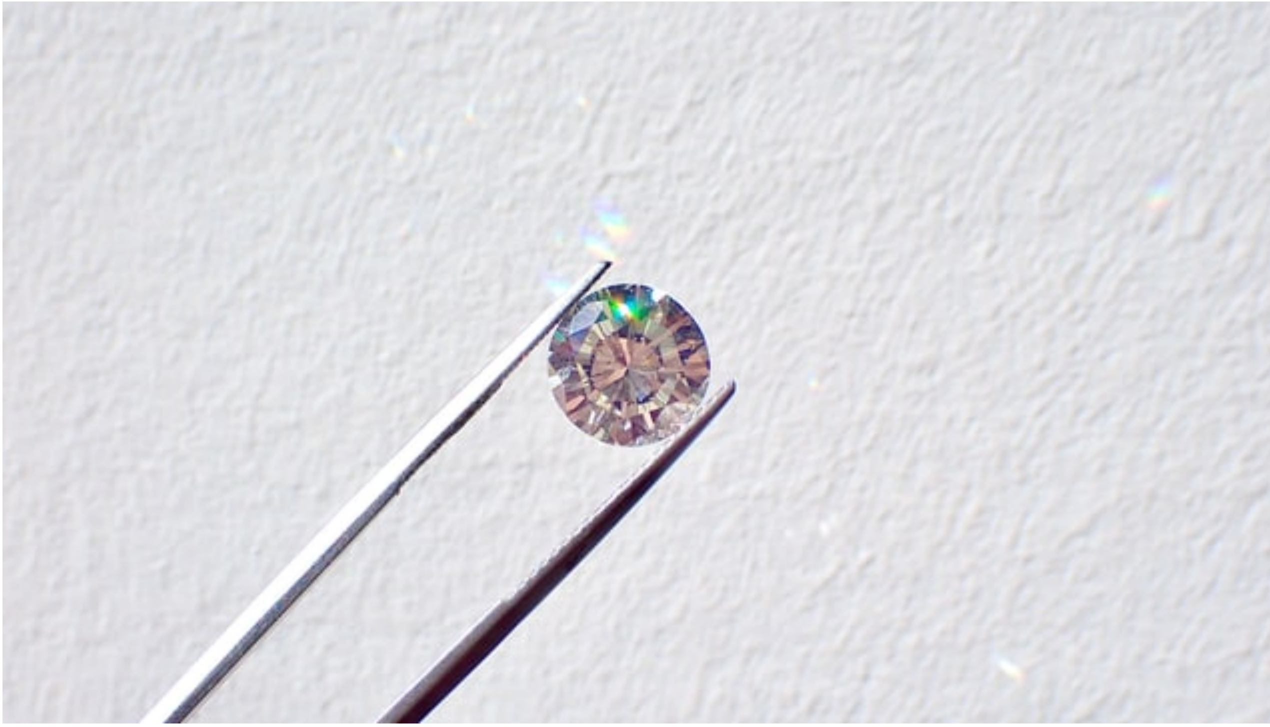


Photo by Tahlia Doyle on Unsplash

Business Understanding

You've been asked to perform an analysis to see how various factors impact the price of diamonds. There are various guides online that claim to tell consumers how to avoid getting "ripped off", but you've been asked to dig into the data to see whether these claims ring true.

Data Understanding

We have downloaded a diamonds dataset from Kaggle, which came with this description:

- **price** price in US dollars ($326--$18,823)
- **carat** weight of the diamond (0.2--5.01)
- **cut** quality of the cut (Fair, Good, Very Good, Premium, Ideal)

- **color** diamond colour, from J (worst) to D (best)
- **clarity** a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
- **x** length in mm (0--10.74)
- **y** width in mm (0--58.9)
- **z** depth in mm (0--31.8)
- **depth** total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)
- **table** width of top of diamond relative to widest point (43--95)

Requirements

1. Load the Data Using Pandas

Practice once again with loading CSV data into a `pandas` dataframe.

2. Build a Baseline Simple Linear Regression Model

Identify the feature that is most correlated with `price` and build a StatsModels linear regression model using just that feature.

3. Evaluate and Interpret Baseline Model Results

Explain the overall performance as well as parameter coefficients for the baseline simple linear regression model.

4. Prepare a Categorical Feature for Multiple Regression Modeling

Identify a promising categorical feature and use `pd.get_dummies()` to prepare it for modeling.

5. Build a Multiple Linear Regression Model

Using the data from Step 4, create a second StatsModels linear regression model using one numeric feature and one one-hot encoded categorical feature.

6. Evaluate and Interpret Multiple Linear Regression Model Results

Explain the performance of the new model in comparison with the baseline, and interpret the new parameter coefficients.

1. Load the Data Using Pandas

Import `pandas` (with the standard alias `pd`), and load the data from the file `diamonds.csv` into a DataFrame called `diamonds`.

Be sure to specify `index_col=0` to avoid creating an "Unnamed: 0" column.

In [11]:
```
# Your code here
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
diamonds = pd.read_csv('diamonds.csv', index_col=0)
```

The following code checks that you loaded the data correctly:

In [12]:
```
# Run this cell without changes

# diamonds should be a dataframe
assert type(diamonds) == pd.DataFrame

# Check that there are the correct number of rows
assert diamonds.shape[0] == 53940

# Check that there are the correct number of columns
# (if this crashes, make sure you specified `index_col=0`)
assert diamonds.shape[1] == 10
```

Inspect the distributions of the numeric features:

In [13]:
```
# Run this cell without changes
diamonds.describe()
```

Out[13]:

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

And inspect the value counts for the categorical features:

In [14]:

```
# Run this cell without changes
categoricals = diamonds.select_dtypes("object")

for col in categoricals:
    print(diamonds[col].value_counts(), "\n")
```

```
cut
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: count, dtype: int64

color
G    11292
E     9797
F     9542
H     8304
D     6775
I     5422
J     2808
Name: count, dtype: int64

clarity
SI1     13065
VS2     12258
SI2      9194
VS1      8171
VVS2     5066
VVS1     3655
IF       1790
I1        741
Name: count, dtype: int64
```

2. Build a Baseline Simple Linear Regression Model

Identifying a Highly Correlated Predictor

The target variable is `price` . Look at the correlation coefficients for all of the predictor variables to find the one with the highest correlation with `price` .

In [22]:
```
# Your code here - look at correlations
# Show correlations between numeric columns
```

```python
corr = diamonds.corr(numeric_only=True)

# Display correlation of all variables with price
corr['price'].sort_values(ascending=False)
```

Out[22]:

```
price    1.000000
carat    0.921591
x        0.884435
y        0.865421
z        0.861249
table    0.127134
depth   -0.010647
Name: price, dtype: float64
```

In [ ]:

```
Identify the name of the predictor column with the strongest correlation below.
```

In [25]:

```python
# Replace None with appropriate code
most_correlated ='carat'
```

The following code checks that you specified a column correctly:

In [26]:

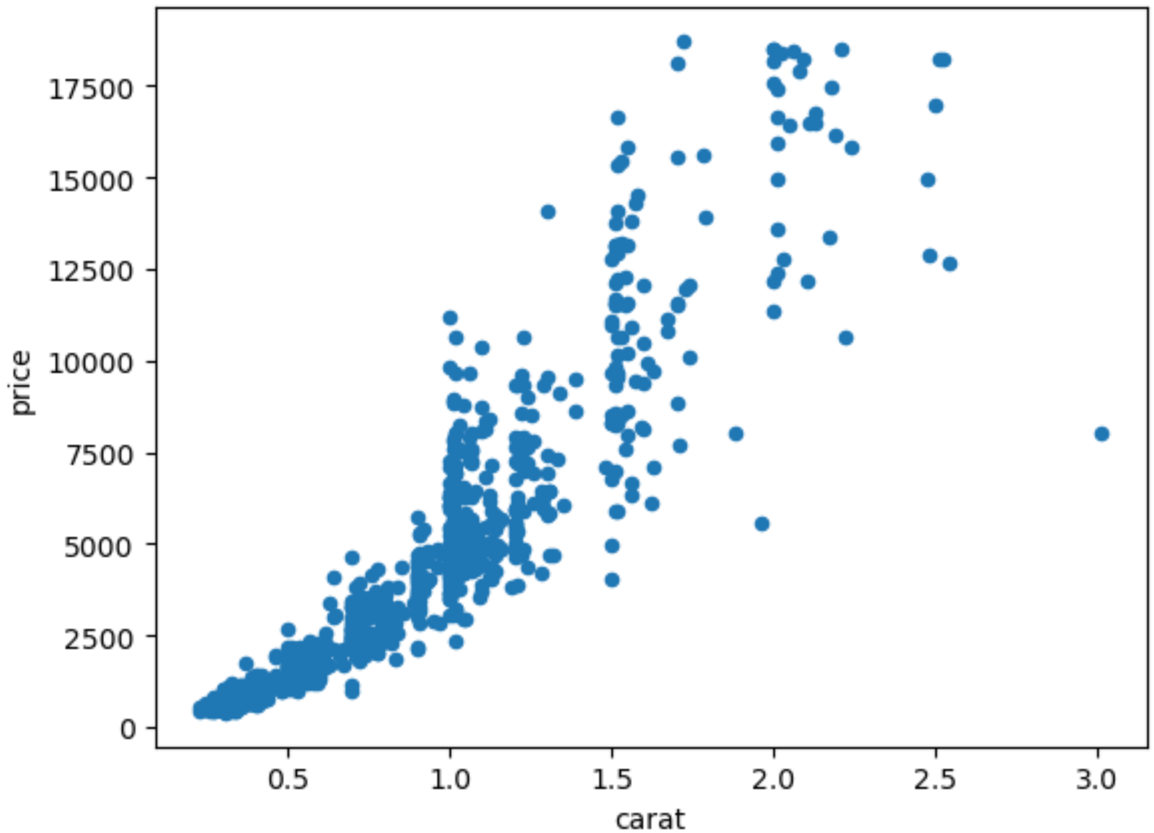
```python
# Run this cell without changes

# most_correlated should be a string
assert type(most_correlated) == str

# most_correlated should be one of the columns other than price
assert most_correlated in diamonds.drop("price", axis=1).columns
```

Plotting the Predictor vs. Price

We'll also create a scatter plot of that variable vs. `price` :

In [27]:

```python
# Run this cell without changes

# Plot a sample of 1000 data points, most_correlated vs. price
diamonds.sample(1000, random_state=1).plot.scatter(x=most_correlated, y="price");
```



Setting Up Variables for Regression

Declare `y` and `X_baseline` variables, where `y` is a Series containing `price` data and `X_baseline` is a DataFrame containing the column with the strongest correlation.

In [29]:
```
# Replace None with appropriate code

y = diamonds['price']


X_baseline = diamonds[['carat']]
```

The following code checks that you created valid `y` and `X_baseline` variables:

In [30]:
```
# Run this code without changes

# y should be a series
assert type(y) == pd.Series

# y should contain about 54k rows
assert y.shape == (53940,)

# X_baseline should be a DataFrame
assert type(X_baseline) == pd.DataFrame

# X_baseline should contain the same number of rows as y
assert X_baseline.shape[0] == y.shape[0]

# X_baseline should have 1 column
assert X_baseline.shape[1] == 1
```

Creating and Fitting Simple Linear Regression

The following code uses your variables to build and fit a simple linear regression.

In [31]:
```
# Run this cell without changes
import statsmodels.api as sm

baseline_model = sm.OLS(y, sm.add_constant(X_baseline))
baseline_results = baseline_model.fit()
```

3. Evaluate and Interpret Baseline Model Results

Write any necessary code to evaluate the model performance overall and interpret its coefficients.

In [34]:
```
# Your code here
# View summary statistics for the regression
print(baseline_results.summary())

# Predict the fitted values
y_pred = baseline_results.predict(sm.add_constant(X_baseline))
```

```python
# Calculate residuals (errors)
residuals = y - y_pred

# Compute evaluation metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

mae = mean_absolute_error(y, y_pred)
rmse = np.sqrt(mean_squared_error(y, y_pred))
r2 = r2_score(y, y_pred)

print(f"\nModel Performance Metrics:")
print(f"R-squared: {r2:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.849
Model:                            OLS   Adj. R-squared:                  0.849
Method:                 Least Squares   F-statistic:                 3.041e+05
Date:                Sat, 04 Oct 2025   Prob (F-statistic):               0.00
Time:                        14:32:24   Log-Likelihood:            -4.7273e+05
No. Observations:               53940   AIC:                         9.455e+05
Df Residuals:                   53938   BIC:                         9.455e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2256.3606     13.055   -172.830      0.000   -2281.949   -2230.772
carat       7756.4256     14.067    551.408      0.000    7728.855    7783.996
==============================================================================
Omnibus:                    14025.341   Durbin-Watson:                   0.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           153030.525
Skew:                           0.939   Prob(JB):                         0.00
Kurtosis:                      11.035   Cond. No.                         3.65
==============================================================================

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

Model Performance Metrics:
R-squared: 0.8493
Mean Absolute Error (MAE): 1007.46
Root Mean Squared Error (RMSE): 1548.53
```

Then summarize your findings below:

Your written answer here

The baseline simple linear regression using carat as the sole predictor shows a very strong positive relationship with diamond price, explaining about 85% ($R^2 = 0.849$) of the variation in prices. The model estimates that each additional 1 carat increases the price by approximately USD 7,756, confirming carat weight as the primary driver of diamond value. The relatively low MAE (USD 1,007) and RMSE (USD1,549) indicate good predictive accuracy, though residual patterns suggest some non-linearity and heteroscedasticity for larger stones. Overall, the model provides a strong baseline for understanding diamond pricing and sets a solid foundation for further improvement by adding other characteristics such as cut, color, and clarity.

▸ **Solution (click to expand)**

`carat` was the attribute most strongly correlated with `price` , therefore our model is describing this relationship.

Overall this model is statistically significant and explains about 85% of the variance in price. In a typical prediction, the model is off by about $1k.

- The intercept is at about -$2.3k. This means that a zero-carat diamond would sell for -$2.3k.
- The coefficient for `carat` is about $7.8k. This means for each additional carat, the diamond costs about $7.8k more.

4. Prepare a Categorical Feature for Multiple Regression Modeling

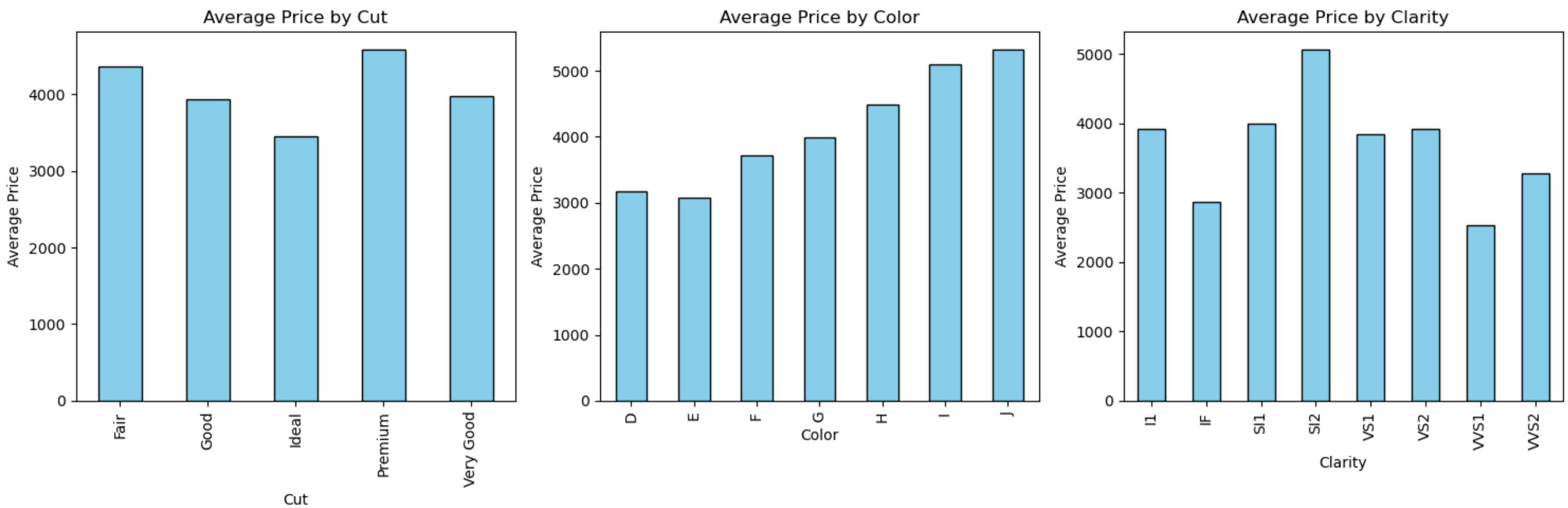
Now let's go beyond our simple linear regression and add a categorical feature.

Identifying a Promising Predictor

Below we create bar graphs for the categories present in each categorical feature:

In [36]:
```
# Run this code without changes
import matplotlib.pyplot as plt

categorical_features = diamonds.select_dtypes("object").columns


fig, axes = plt.subplots(ncols=len(categorical_features), figsize=(15, 5))


for index, feature in enumerate(categorical_features):
    (
        diamonds.groupby(feature)['price']  # only compute mean for numeric 'price'
        .mean()
        .plot(kind='bar', ax=axes[index], color='skyblue', edgecolor='black')
    )
    axes[index].set_title(f"Average Price by {feature.capitalize()}")
```

```
    axes[index].set_ylabel("Average Price")
    axes[index].set_xlabel(feature.capitalize())

plt.tight_layout()
plt.show()
```



Identify the name of the categorical predictor column you want to use in your model below. The choice here is more open-ended than choosing the numeric predictor above -- choose something that will be interpretable in a final model, and where the different categories seem to have an impact on the price.

In [37]:
```
# Replace None with appropriate code
cat_col = 'cut'
```

The following code checks that you specified a column correctly:

In [39]:
```
# Run this cell without changes

# cat_col should be a string
assert type(cat_col) == str

# cat_col should be one of the categorical columns
assert cat_col in diamonds.select_dtypes("object").columns
```

Setting Up Variables for Regression

The code below creates a variable `X_iterated` : a DataFrame containing the column with the strongest correlation **and** your selected categorical feature.

In [40]:
```
# Run this cell without changes
X_iterated = diamonds[[most_correlated, cat_col]]
X_iterated
```

Out[40]:

	carat	cut
1	0.23	Ideal
2	0.21	Premium
3	0.23	Good
4	0.29	Premium
5	0.31	Good
...	...	...
53936	0.72	Ideal
53937	0.72	Good
53938	0.70	Very Good
53939	0.86	Premium
53940	0.75	Ideal

53940 rows × 2 columns

Preprocessing Categorical Variable

If we tried to pass `X_iterated` as-is into `sm.OLS` , we would get an error. We need to use `pd.get_dummies` to create dummy variables for `cat_col` .

DO NOT use `drop_first=True` , so that you can intentionally set a meaningful reference category instead.

In [45]:
```
# Replace None with appropriate code

# Use pd.get_dummies to one-hot encode the categorical column in X_iterated
X_iterated = pd.concat([diamonds[['carat']], pd.get_dummies(diamonds[cat_col], prefix=cat_col)], axis=1)

X_iterated
```

Out[45]:

	carat	cut_Fair	cut_Good	cut_Ideal	cut_Premium	cut_Very Good
1	0.23	False	False	True	False	False
2	0.21	False	False	False	True	False
3	0.23	False	True	False	False	False
4	0.29	False	False	False	True	False
5	0.31	False	True	False	False	False
...	...	...	...	...	...	...
53936	0.72	False	False	True	False	False
53937	0.72	False	True	False	False	False
53938	0.70	False	False	False	False	True
53939	0.86	False	False	False	True	False
53940	0.75	False	False	True	False	False

53940 rows × 6 columns

The following code checks that you have the right number of columns:

In [46]:
```
# Run this cell without changes

# X_iterated should be a dataframe
assert type(X_iterated) == pd.DataFrame
```

```python
# You should have the number of unique values in one of the
# categorical columns + 1 (representing the numeric predictor)
valid_col_nums = diamonds.select_dtypes("object").nunique() + 1

# Check that there are the correct number of columns
# (if this crashes, make sure you did not use `drop_first=True`)
assert X_iterated.shape[1] in valid_col_nums.values
```

Now, applying your domain understanding, **choose a column to drop and drop it**. This category should make sense as a "baseline" or "reference". For the "cut_Very Good" column that was generated when `pd.get_dummies` was used, we need to remove the space in the column name.

In [47]:

```python
# Your code here
# Remove space in column names for consistency
X_iterated.columns = X_iterated.columns.str.replace(' ', '_')

# Drop one category to serve as reference (e.g., 'cut_Ideal')
X_iterated = X_iterated.drop(columns=['cut_Ideal'])

# Check updated columns
X_iterated.head()
```

Out[47]:

	carat	cut_Fair	cut_Good	cut_Premium	cut_Very_Good
1	0.23	False	False	False	False
2	0.21	False	False	True	False
3	0.23	False	True	False	False
4	0.29	False	False	True	False
5	0.31	False	True	False	False

We now need to change the boolean values for the four "cut" column to 1s and 0s in order for the regression to run.

In [48]:

```python
# Your code here
# Convert boolean dummy columns (True/False) to numeric (1/0)
X_iterated = X_iterated.astype(int)
```

```
# Check result
X_iterated.head()
```

Out[48]:

	carat	cut_Fair	cut_Good	cut_Premium	cut_Very_Good
1	0	0	0	0	0
2	0	0	0	1	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	1	0	0

Now you should have 1 fewer column than before:

In [49]:
```
# Run this cell without changes

# Check that there are the correct number of columns
assert X_iterated.shape[1] in (valid_col_nums - 1).values
```

5. Build a Multiple Linear Regression Model

Using the `y` variable from our previous model and `X_iterated`, build a model called `iterated_model` and a regression results object called `iterated_results`.

In [51]:
```
# Your code here
import statsmodels.api as sm

# Build and fit multiple linear regression: price ~ carat + cut dummies
iterated_model = sm.OLS(y, sm.add_constant(X_iterated))
iterated_results = iterated_model.fit()
```

6. Evaluate and Interpret Multiple Linear Regression Model Results

If the model was set up correctly, the following code will print the results summary.

In [52]:
```
# Run this cell without changes
print(iterated_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.730
Model:                            OLS   Adj. R-squared:                  0.730
Method:                 Least Squares   F-statistic:                 2.919e+04
Date:                Sat, 04 Oct 2025   Prob (F-statistic):               0.00
Time:                        14:56:59   Log-Likelihood:            -4.8844e+05
No. Observations:               53940   AIC:                         9.769e+05
Df Residuals:                   53934   BIC:                         9.770e+05
Df Model:                           5
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          1669.6722     14.885    112.171      0.000    1640.497    1698.847
carat          6033.5706     15.933    378.673      0.000    6002.341    6064.800
cut_Fair      -1020.9982     53.782    -18.984      0.000   -1126.411    -915.586
cut_Good       -364.0458     32.856    -11.080      0.000    -428.444    -299.648
cut_Premium    -160.1686     22.852     -7.009      0.000    -204.959    -115.378
cut_Very_Good   -47.0074     23.601     -1.992      0.046     -93.265      -0.750
==============================================================================
Omnibus:                    11688.261   Durbin-Watson:                   0.796
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            37224.268
Skew:                           1.105   Prob(JB):                         0.00
Kurtosis:                       6.417   Cond. No.                         7.11
==============================================================================

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

Summarize your findings below. How did the iterated model perform overall? How does this compare to the baseline model? What do the coefficients mean?

Create as many additional cells as needed.

Your written answer here The multiple regression model including both carat and cut categories explains about 73% of the variation in diamond prices ($R^2 = 0.73$), which is lower than the baseline model's $R^2 = 0.85$ using carat alone. This indicates that while cut has a statistically significant effect, it does not add much explanatory power beyond carat. The coefficient for carat (6033.57) remains positive and highly significant, meaning that for every one-carat increase, price rises by roughly USD 6,034, holding cut constant. The negative coefficients for cut levels (Fair, Good, Premium, Very Good) indicate that these cuts are priced below the reference category (Ideal), consistent with the notion that Ideal cuts command higher prices. For example, a Fair cut diamond is priced about USD 1,021 less than an Ideal cut, on average. Overall, while cut quality does

influence prices, carat weight remains the dominant determinant of diamond value, and including categorical cut variables refines but does not drastically improve model performance.

Summary

Congratulations, you completed an iterative linear regression process! You practiced developing a baseline and an iterated model, as well as identifying promising predictors from both numeric and categorical features.