PSTAT126 Final Project

March 19, 2025

[46]:
```
import pandas as pd
import zipfile
print ("Part 1")
# Define the path to the zip file in your Downloads folder
zip_path = '/Users/ruilanzeng/Downloads/Diamonds Prices2022.csv.zip'
extract_dir = '/Users/ruilanzeng/Downloads/'  # Directory where the CSV will be␣
↪extracted

# Extract the CSV file from the zip archive
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

# Construct the CSV file path (assuming the CSV file is named "Diamonds␣
↪Prices2022.csv")
csv_path = extract_dir + 'Diamonds Prices2022.csv'

# Load the CSV file into a DataFrame
df = pd.read_csv(csv_path)

# Display the column names to inspect the variables in the dataset
print("Dataset columns:")
print(df.columns.tolist())

# Select a random sample of 10 rows (using a fixed seed for reproducibility)
sample_df = df.sample(n=10, random_state=42)
print("\nRandom sample of 10 rows:")
print(sample_df)
```

```
Part 1
Dataset columns:
['Unnamed: 0', 'carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price',
'x', 'y', 'z']

Random sample of 10 rows:
       Unnamed: 0  carat        cut color clarity  depth  table  price     x  \
1388         1389   0.24      Ideal     G    VVS1   62.1   56.0    559  3.97
19841       19842   1.21  Very Good     F     VS2   62.9   54.0   8403  6.78
41647       41648   0.50       Fair     E     SI1   61.7   68.0   1238  5.09
```

```
41741        41742   0.50      Ideal     D      SI2   62.8    56.0    1243  5.06
17244        17245   1.55      Ideal     E      SI2   62.3    55.0    6901  7.44
1608          1609   1.00       Fair     E      SI2   55.4    62.0    3011  6.63
46401        46402   0.51      Ideal     H     VVS1   62.2    56.0    1766  5.12
24625        24626   1.52    Premium     G      VS2   62.6    55.0   12958  7.39
49388        49389   0.57      Ideal     D      VS2   61.8    56.0    2103  5.34
10460        10461   1.14      Ideal     H      SI1   60.3    57.0    4789  6.79

          y     z
1388   4.00  2.47
19841  6.82  4.28
41647  5.03  3.12
41741  5.03  3.17
17244  7.37  4.61
1608   6.59  3.66
46401  5.14  3.19
24625  7.28  4.59
49388  5.31  3.29
10460  6.85  4.11
```

[2]:
```python
import pandas as pd

# Assuming `df` is your DataFrame of the Diamonds dataset
print("Dataset Information:")
df.info()
```

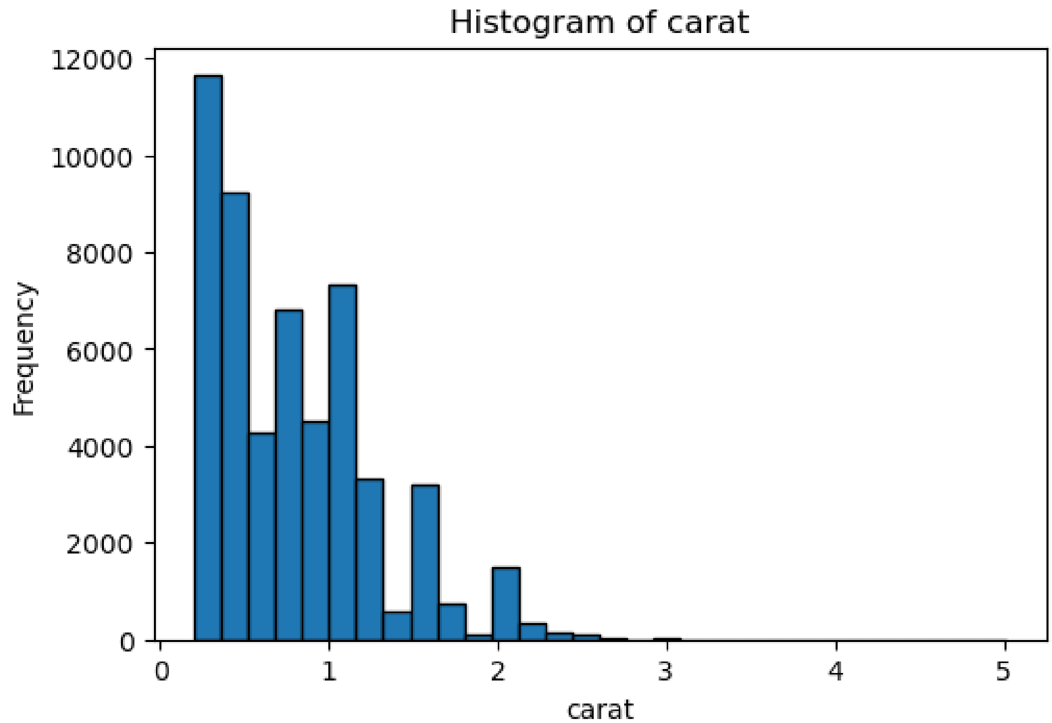
```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53943 entries, 0 to 53942
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53943 non-null  int64
 1   carat       53943 non-null  float64
 2   cut         53943 non-null  object
 3   color       53943 non-null  object
 4   clarity     53943 non-null  object
 5   depth       53943 non-null  float64
 6   table       53943 non-null  float64
 7   price       53943 non-null  int64
 8   x           53943 non-null  float64
 9   y           53943 non-null  float64
 10  z           53943 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

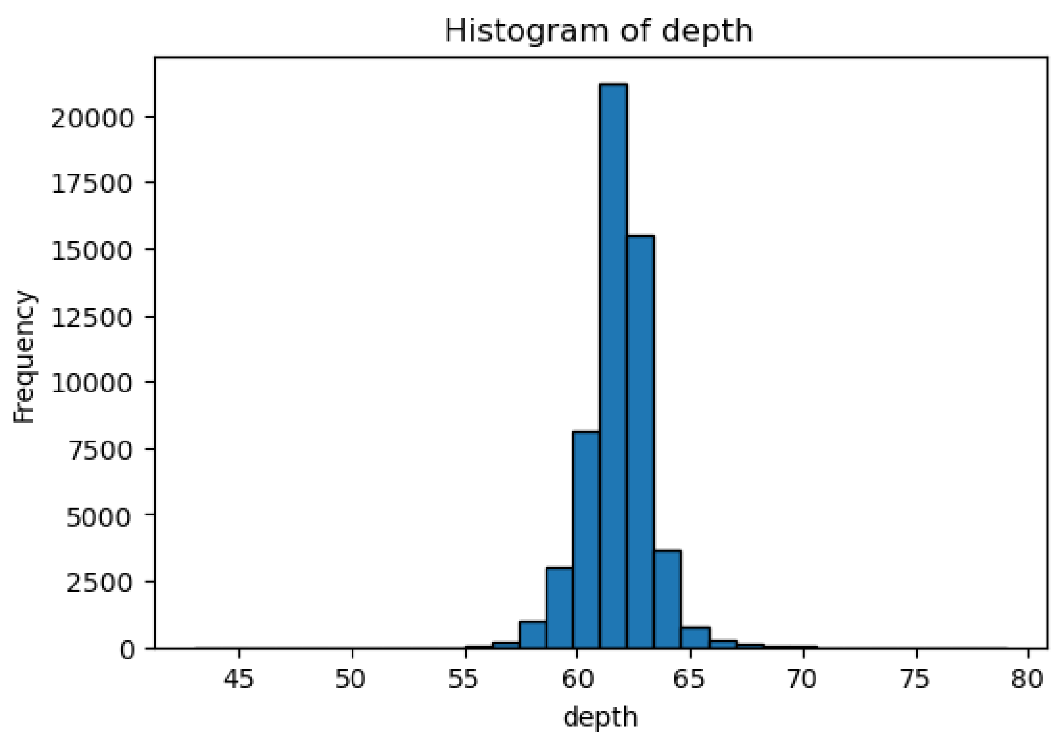
[3]:
```python
print("\nSummary Statistics (Numerical Variables):")
print(df.describe())
```

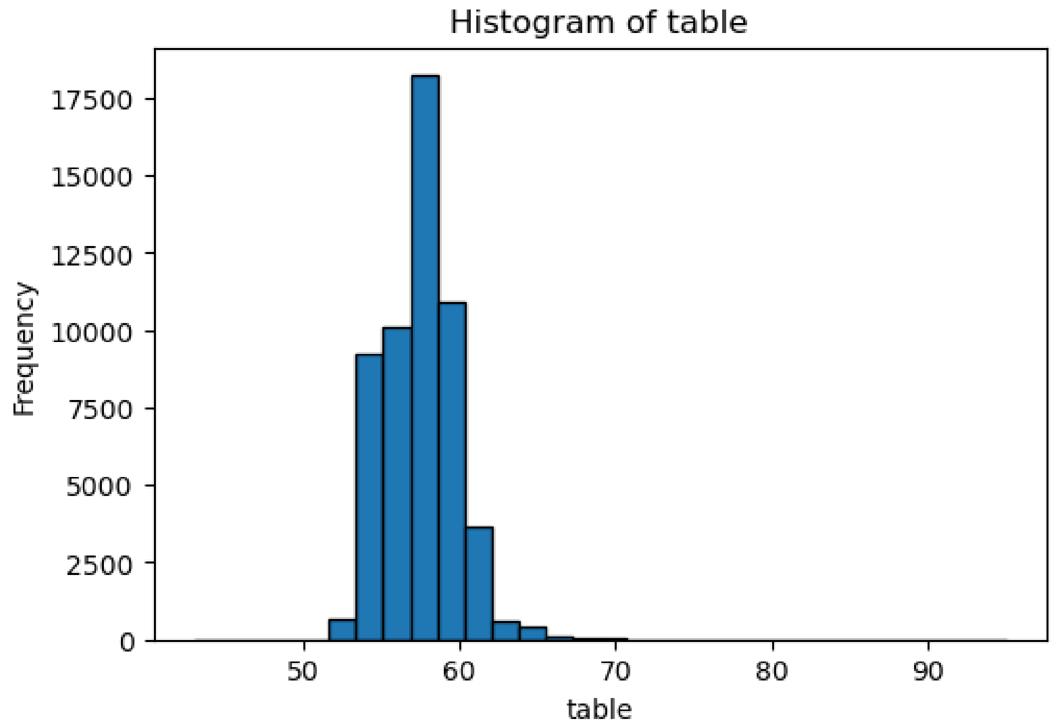
```
Summary Statistics (Numerical Variables):
         Unnamed: 0         carat         depth         table         price  \
count  53943.000000  53943.000000  53943.000000  53943.000000  53943.000000
mean   26972.000000      0.797935     61.749322     57.457251   3932.734294
std    15572.147122      0.473999      1.432626      2.234549   3989.338447
min        1.000000      0.200000     43.000000     43.000000    326.000000
25%    13486.500000      0.400000     61.000000     56.000000    950.000000
50%    26972.000000      0.700000     61.800000     57.000000   2401.000000
75%    40457.500000      1.040000     62.500000     59.000000   5324.000000
max    53943.000000      5.010000     79.000000     95.000000  18823.000000

                  x             y             z
count  53943.000000  53943.000000  53943.000000
mean       5.731158      5.734526      3.538730
std        1.121730      1.142103      0.705679
min        0.000000      0.000000      0.000000
25%        4.710000      4.720000      2.910000
50%        5.700000      5.710000      3.530000
75%        6.540000      6.540000      4.040000
max       10.740000     58.900000     31.800000
```

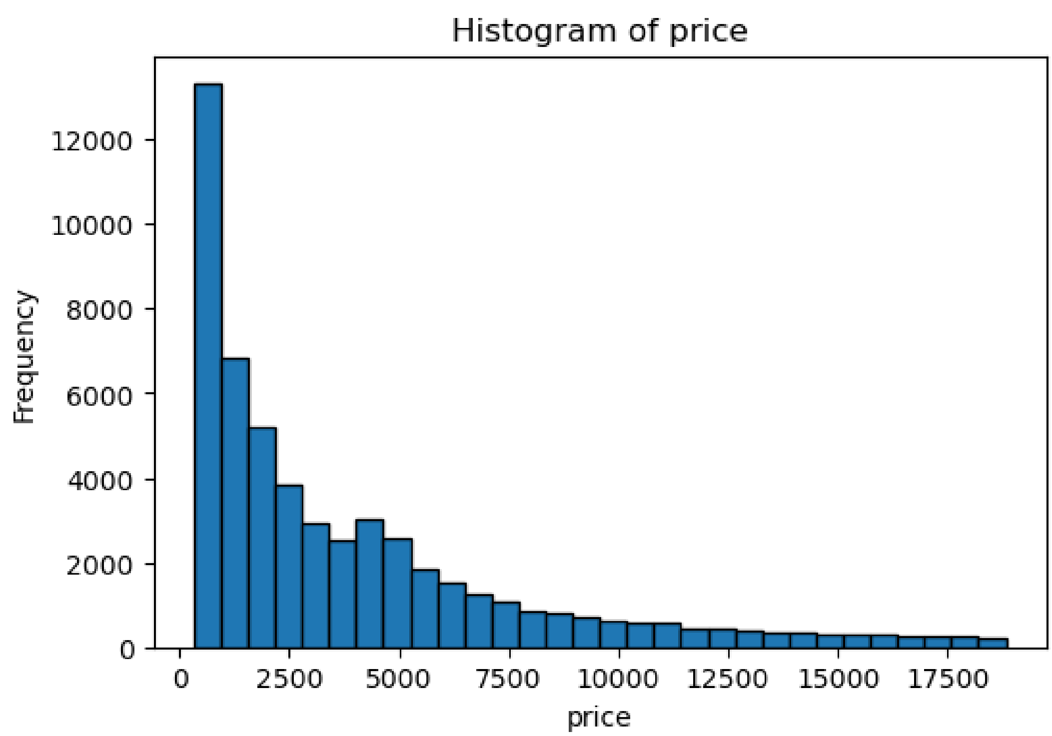
[4]:

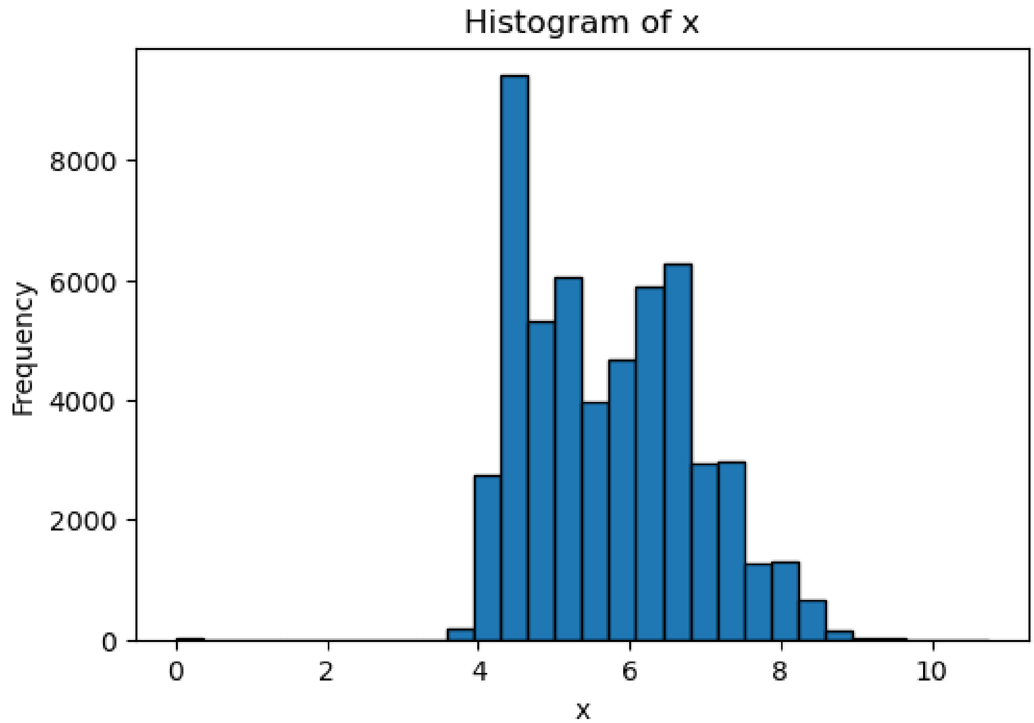
```python
import matplotlib.pyplot as plt

continuous_vars = ['carat', 'depth', 'table', 'price', 'x', 'y', 'z']

for var in continuous_vars:
    plt.figure(figsize=(6,4))
    plt.hist(df[var], bins=30, edgecolor='black')
    plt.title(f'Histogram of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.show()
```

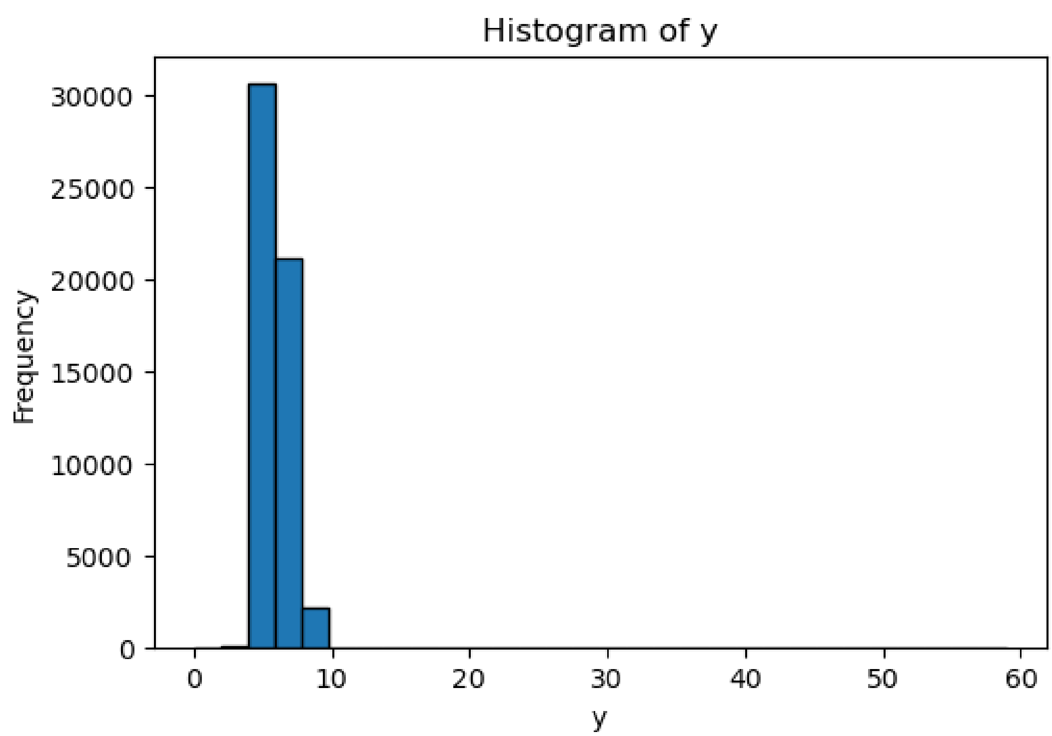

Histogram of carat
Frequency
12000
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
carat

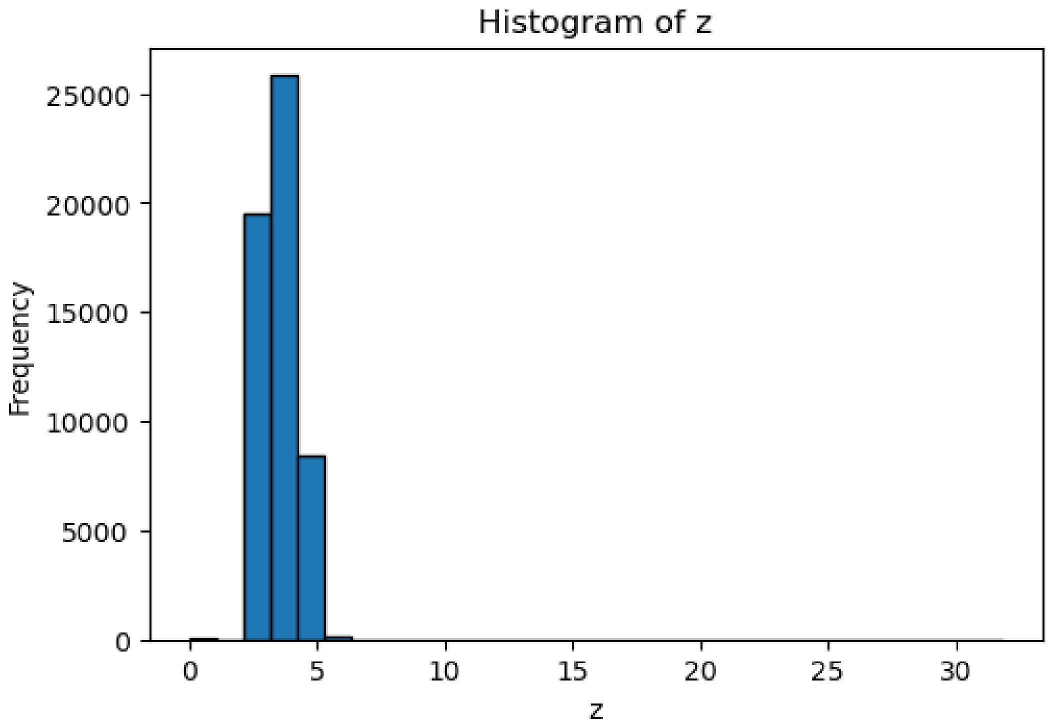

Histogram of depth
Frequency
20000
17500
15000
12500
10000
7500
5000
2500
0
45
50
55
60
65
70
75
80
depth


Histogram of table
Frequency
17500
15000
12500
10000
7500
5000
2500
0
50
60
70
80
90
table

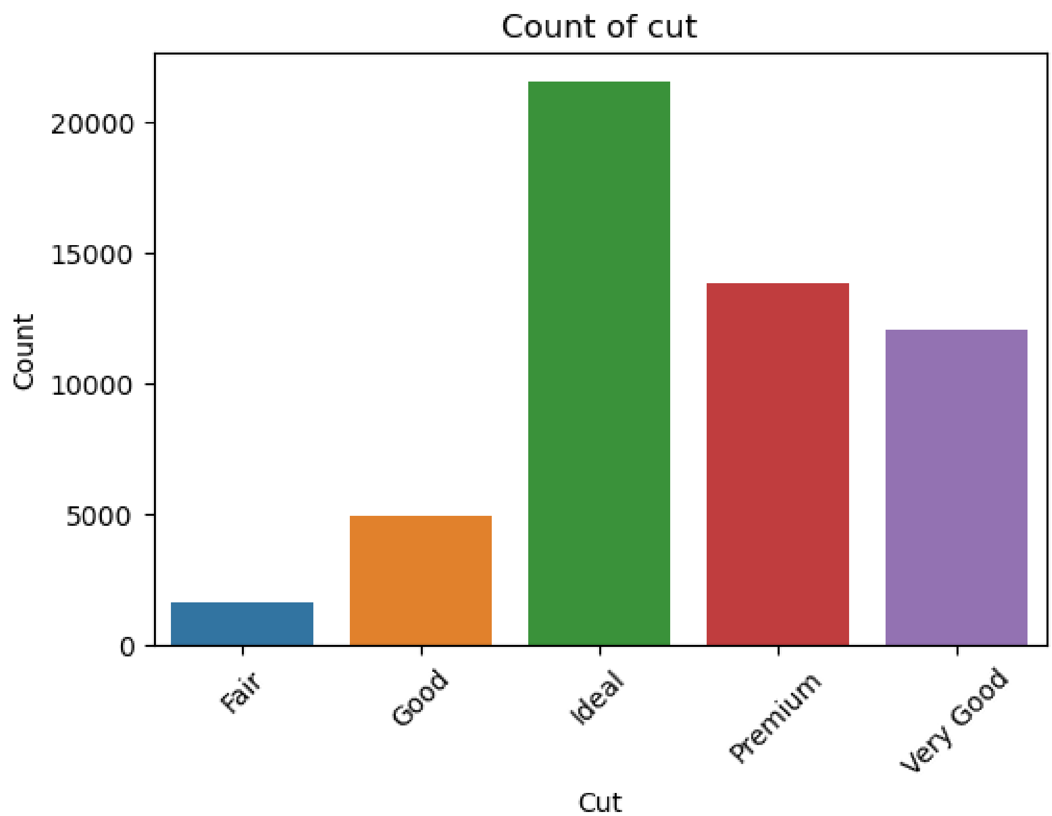

Histogram of price
Frequency
12000
10000
8000
6000
4000
2000
0
0
2500
5000
7500
10000
12500
15000
17500
price

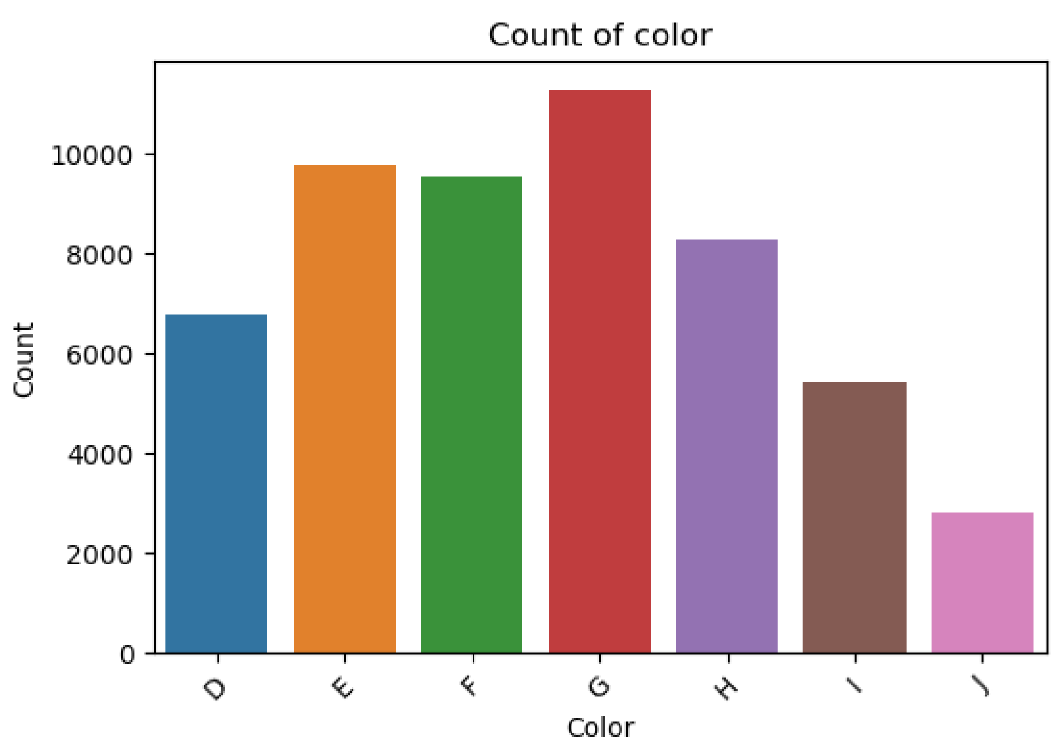

Histogram of x
Frequency
8000
6000
4000
2000
0
0
2
4
6
8
10
x

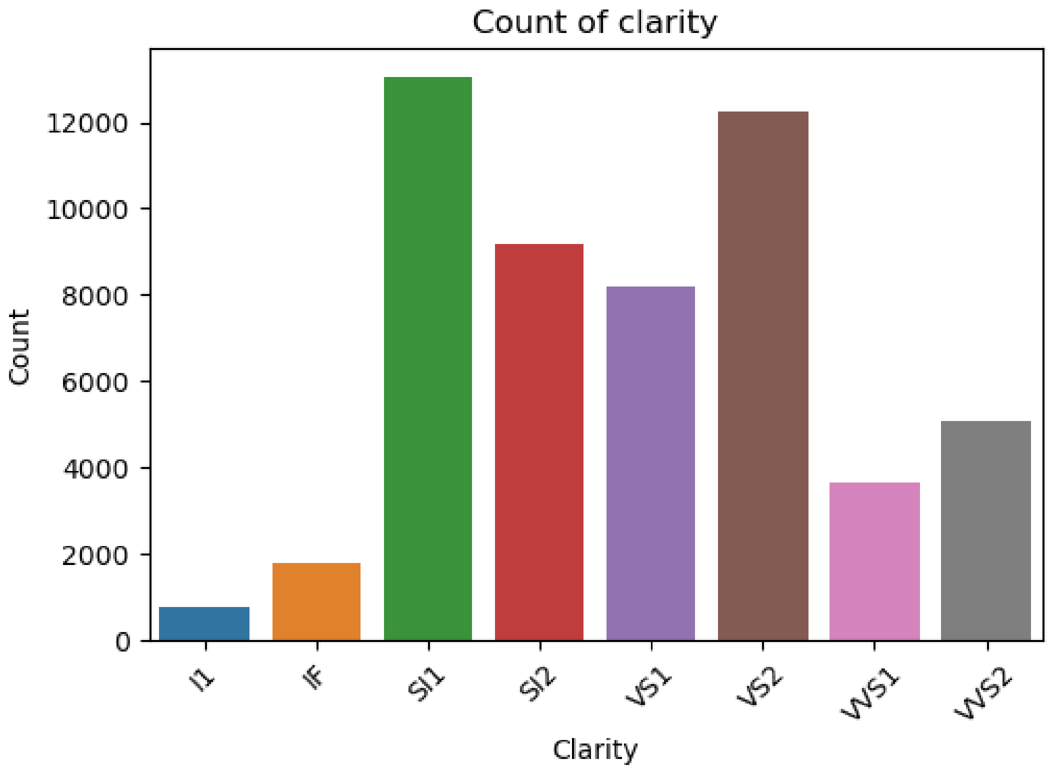

Histogram of y
Frequency
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50
60
y



[5]:
```
import seaborn as sns

categorical_vars = ['cut', 'color', 'clarity']

for var in categorical_vars:
    plt.figure(figsize=(6,4))
    sns.countplot(data=df, x=var, order=sorted(df[var].unique()))
    plt.title(f'Count of {var}')
    plt.xlabel(var.capitalize())
    plt.ylabel('Count')
    plt.xticks(rotation=45)  # rotate labels if needed
    plt.show()
```

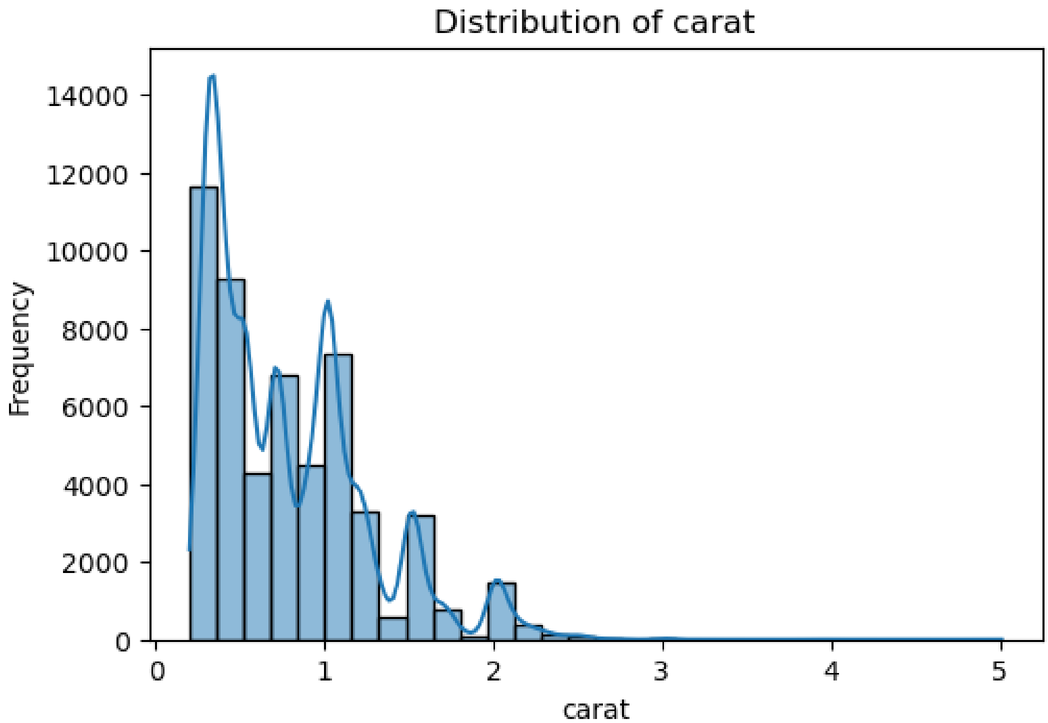

Count of cut
Count
20000
15000
10000
5000
0
Fair
Good
Ideal
Premium
Very Good
Cut

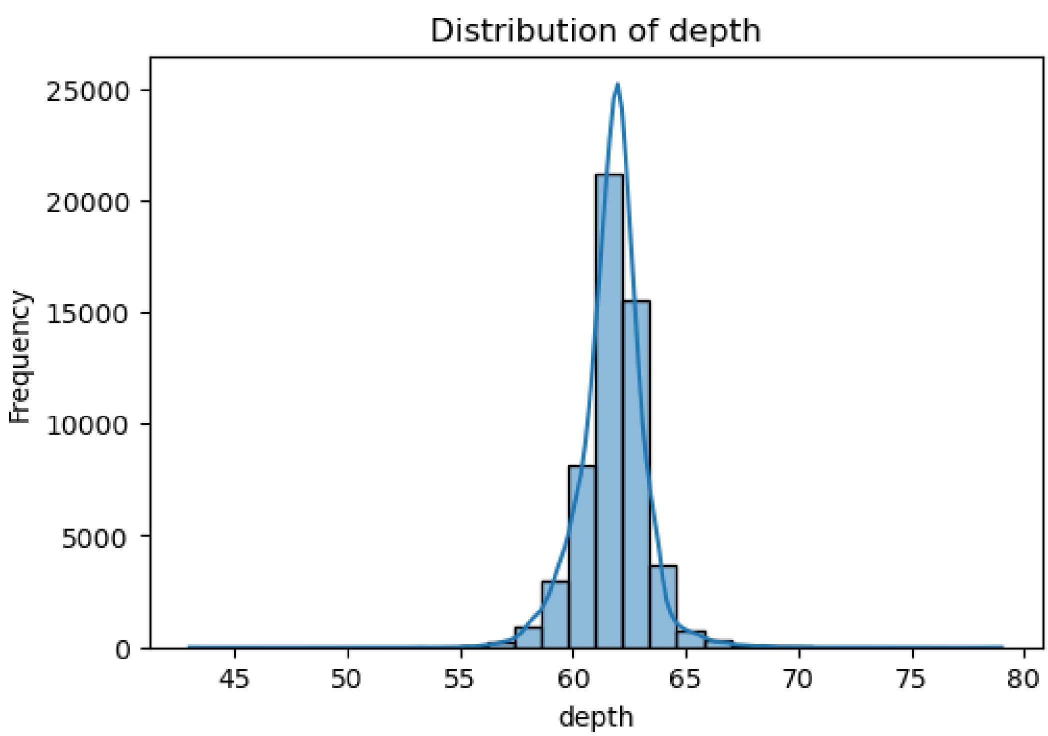

Count of color
Count
10000
8000
6000
4000
2000
0
D
E
F
G
H
I
J
Color

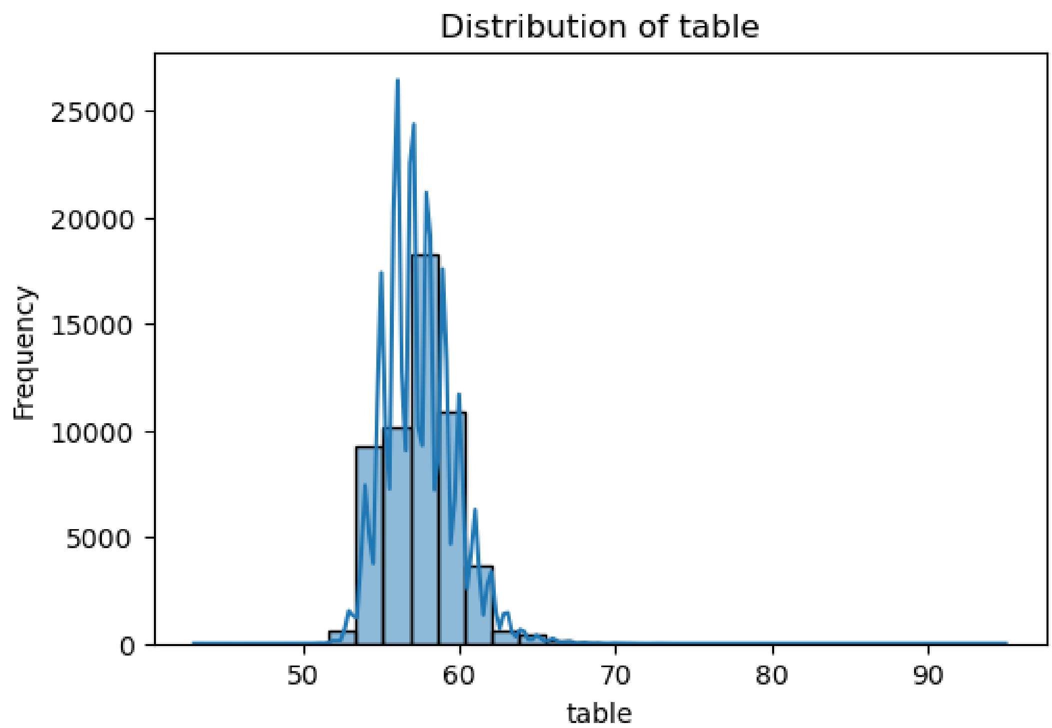


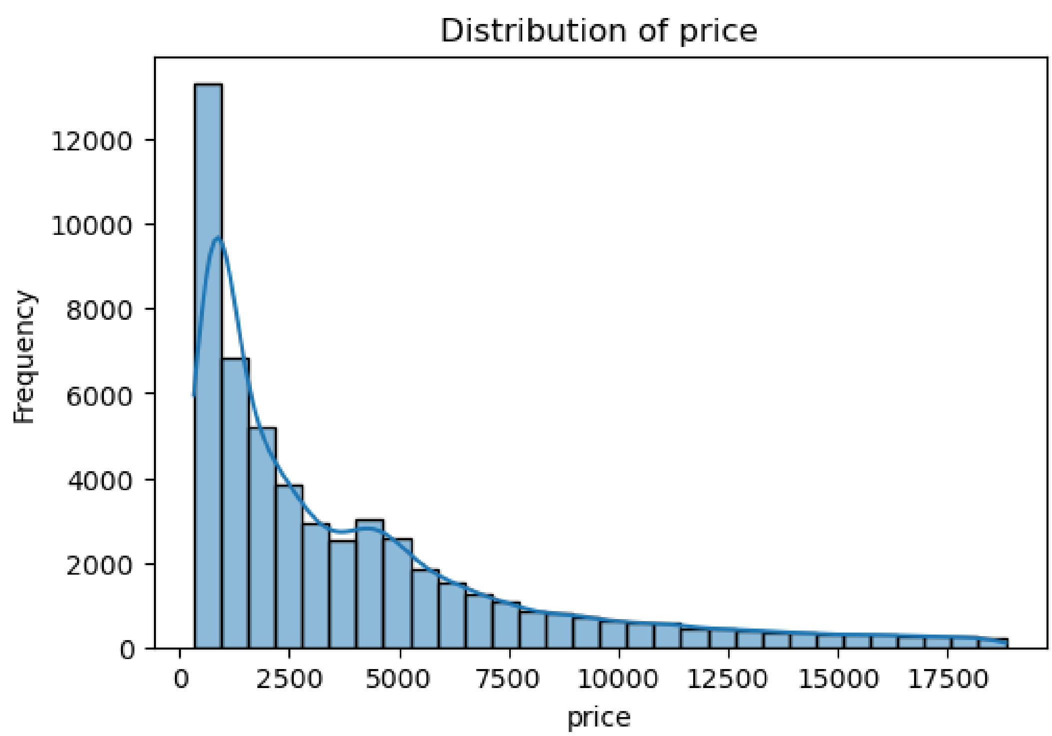
[8]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Load the dataset (adjust file path as needed)
# df = pd.read_csv('/Users/ruilanzeng/Downloads/Diamonds Prices2022.csv')


# 3. Histograms for continuous variables
continuous_vars = ['carat', 'depth', 'table', 'price', 'x', 'y', 'z']

for var in continuous_vars:
    plt.figure(figsize=(6,4))
    sns.histplot(data=df, x=var, bins=30, kde=True)  # kde=True to overlay a␣
 ↪density curve
    plt.title(f'Distribution of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.show()
```

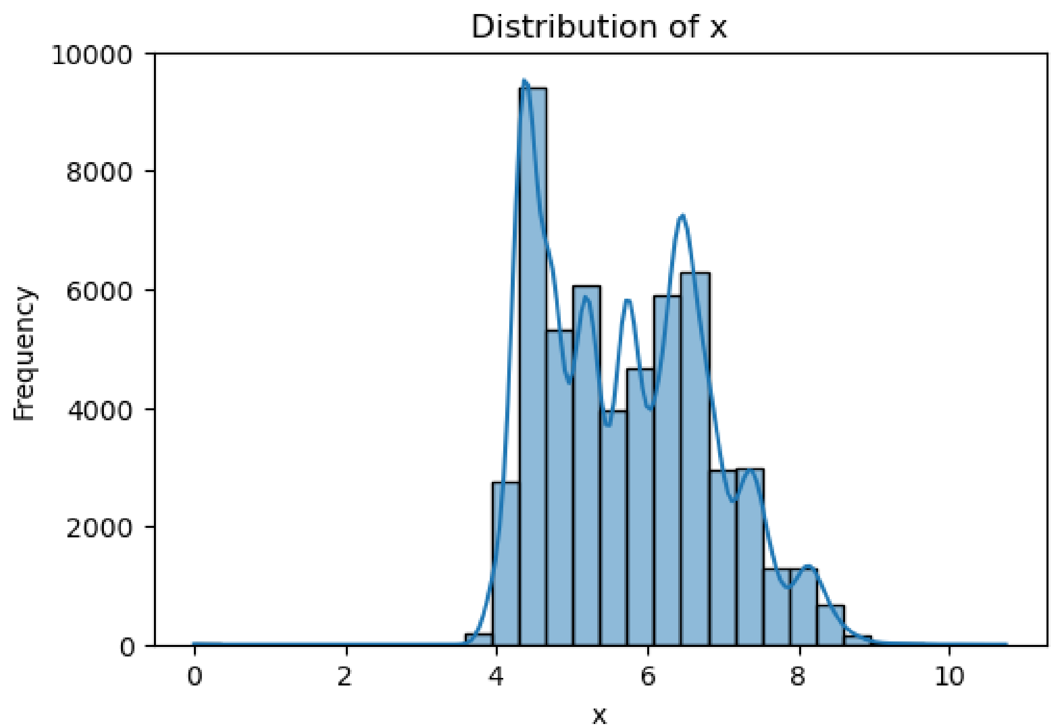
```
print("\n" + "#"*60)
print("COMMENTARY ON DISTRIBUTIONS")
print("#"*60)

print("\n1. NUMERICAL VARIABLES:")
print("- carat: Typically right-skewed, with many diamonds under 1 carat and␣
↪fewer large stones.")
print("- depth: Usually centered around 61-62, reflecting common 'ideal'␣
↪proportions.")
print("- table: Often between 56-58, again near the 'ideal' range.")
print("- price: Right-skewed with many moderately priced diamonds and a long␣
↪tail of expensive ones.")
print("- x, y, z: Similar right-skewed patterns as carat, since larger diamonds␣
↪are less common.")

print("\n2. CATEGORICAL VARIABLES:")
print("- cut: Usually dominated by 'Ideal' and 'Premium', reflecting␣
↪higher-quality cuts.")
print("- color: G and H tend to be most common, with D (colorless) and J␣
↪(slightly tinted) less common.")
print("- clarity: SI1 and VS2 often appear most frequently, with IF (Internally␣
↪Flawless) and I1 (more inclusions) less common.")

print("\nThese observations align with typical market trends, where mid-range␣
↪quality and smaller stones are more abundant, while larger or higher-grade␣
↪stones are relatively rare.")
```

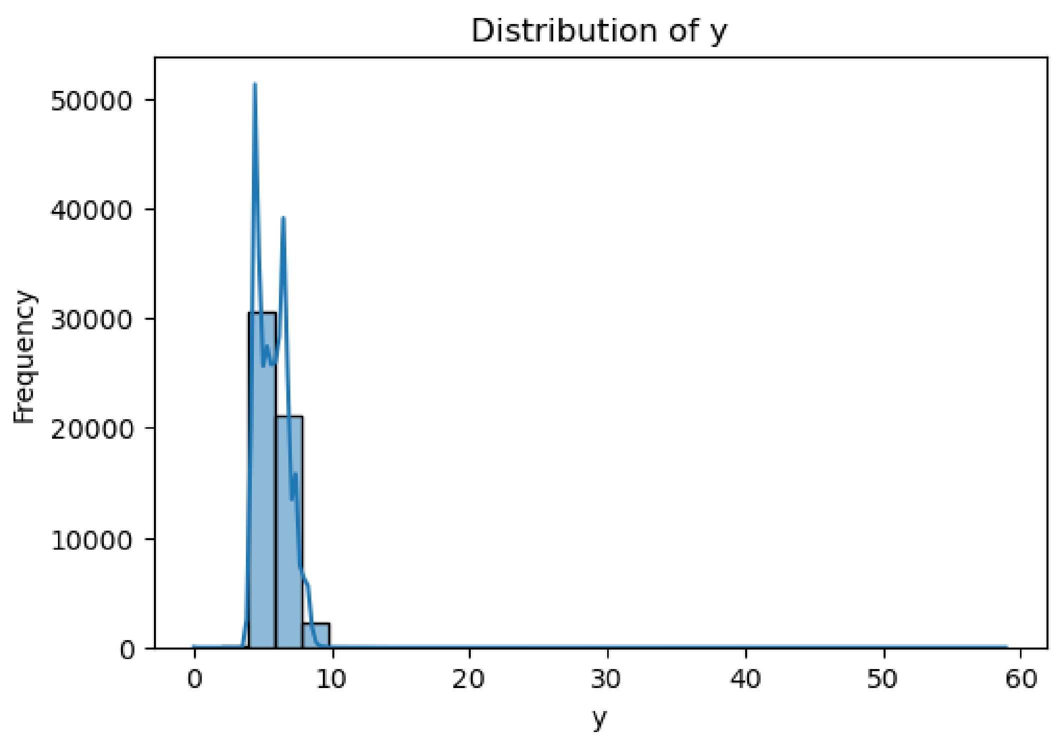

Distribution of carat
Frequency
carat
0
1
2
3
4
5
0
2000
4000
6000
8000
10000
12000
14000

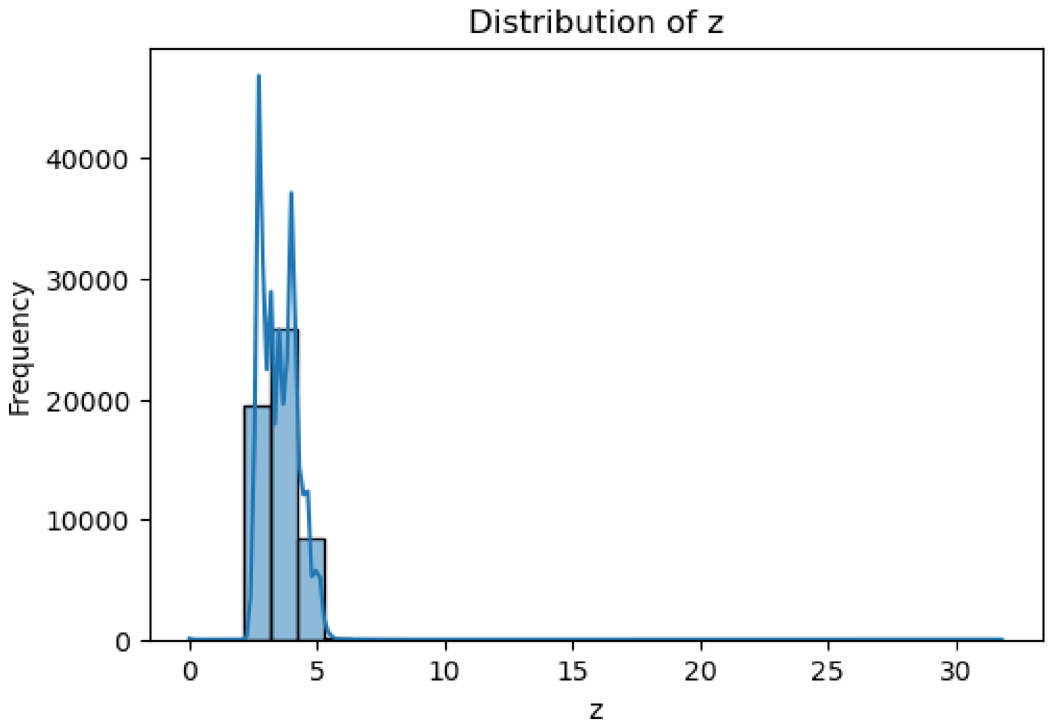

Distribution of depth
Frequency
depth
45
50
55
60
65
70
75
80
0
5000
10000
15000
20000
25000


Distribution of table
Frequency
25000
20000
15000
10000
5000
0
50
60
70
80
90
table


Distribution of price
Frequency
12000
10000
8000
6000
4000
2000
0
0
2500
5000
7500
10000
12500
15000
17500
price


Distribution of x
Frequency
x
0
2000
4000
6000
8000
10000
0
2
4
6
8
10


Distribution of y
Frequency
y
0
10000
20000
30000
40000
50000
0
10
20
30
40
50
60



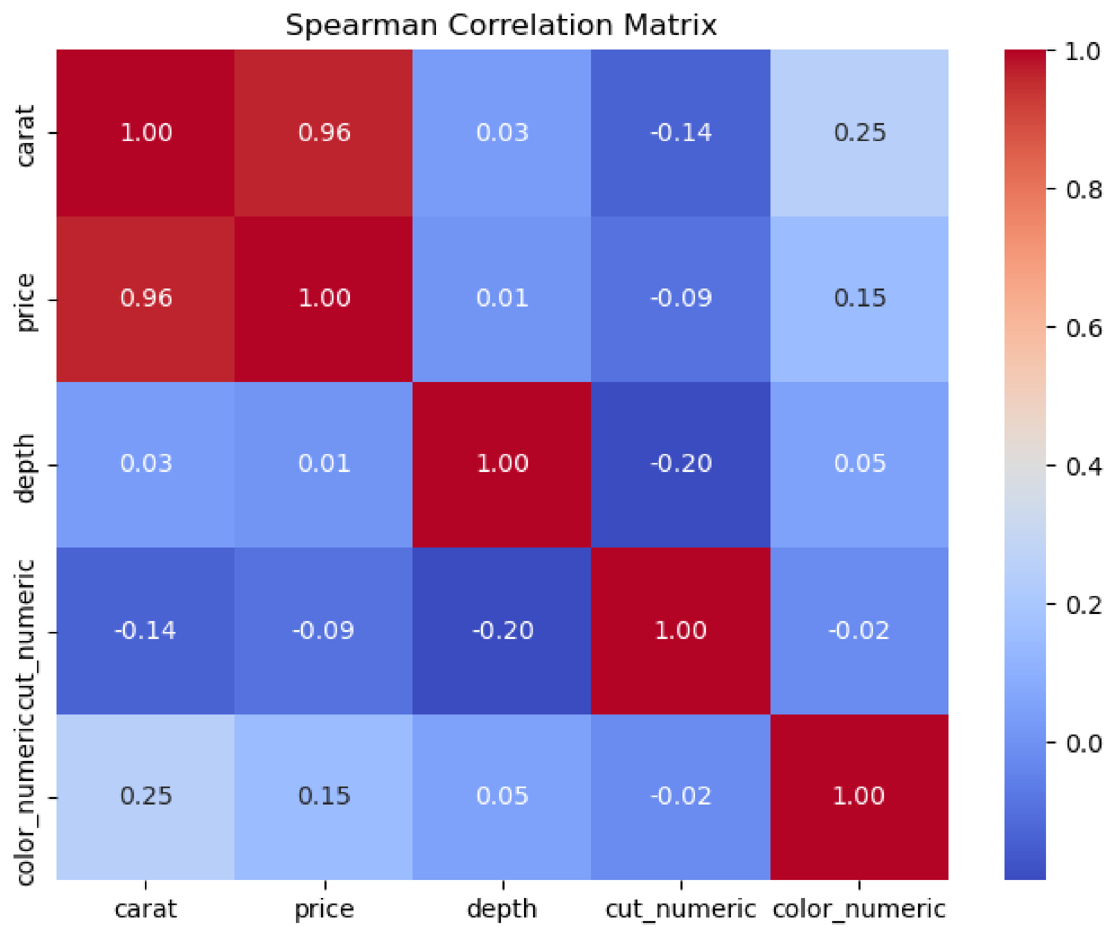

```
############################################################
COMMENTARY ON DISTRIBUTIONS
############################################################

1. NUMERICAL VARIABLES:
- carat: Typically right-skewed, with many diamonds under 1 carat and fewer
large stones.
- depth: Usually centered around 61-62, reflecting common ‘ideal’ proportions.
- table: Often between 56-58, again near the ‘ideal’ range.
- price: Right-skewed with many moderately priced diamonds and a long tail of
expensive ones.
- x, y, z: Similar right-skewed patterns as carat, since larger diamonds are
less common.

2. CATEGORICAL VARIABLES:
- cut: Usually dominated by ‘Ideal’ and ‘Premium’, reflecting higher-quality
cuts.
- color: G and H tend to be most common, with D (colorless) and J (slightly
tinted) less common.
- clarity: SI1 and VS2 often appear most frequently, with IF (Internally
Flawless) and I1 (more inclusions) less common.

These observations align with typical market trends, where mid-range quality and
```

smaller stones are more abundant, while larger or higher-grade stones are relatively rare.

[9]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Load the dataset (update the path as needed)i

# 2. Choose three quantitative variables and two categorical variables
# Quantitative: 'carat', 'price', 'depth'
# Categorical (ordinal): 'cut', 'color'
# We need to map the ordinal categories to numeric values.

# Define the mapping for 'cut' (from worst to best)
cut_order = {"Fair": 1, "Good": 2, "Very Good": 3, "Premium": 4, "Ideal": 5}

# Define the mapping for 'color' (assuming D (best) to J (worst))
color_order = {"D": 1, "E": 2, "F": 3, "G": 4, "H": 5, "I": 6, "J": 7}

# Create new numeric columns for cut and color
df['cut_numeric'] = df['cut'].map(cut_order)
df['color_numeric'] = df['color'].map(color_order)

# Select the variables to analyze
cols = ['carat', 'price', 'depth', 'cut_numeric', 'color_numeric']

# 3. Compute the correlation matrix using Spearman correlation (suitable for␣
↪ordinal data)
corr_matrix = df[cols].corr(method='spearman')
print("Spearman Correlation Matrix:")
print(corr_matrix)

# Visualize the correlation matrix with a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Spearman Correlation Matrix")
plt.show()

# 4. Print Commentary on the correlations
print("\n" + "#"*60)
print("COMMENTARY ON CORRELATIONS")
print("#"*60)
print("\n1. Quantitative Variables:")
print("   - There is usually a strong positive correlation between 'carat' and␣
↪'price',")
print("     indicating that larger diamonds tend to be more expensive.")
```

```
print("   - 'Depth' tends to show a weaker correlation with 'price' compared to␣
↪'carat'.")
print("     This may be because depth percentages are kept within a narrower␣
↪range for ideal cuts.")
print("\n2. Categorical Variables (converted to numeric):")
print("   - For 'cut', higher numeric values represent better quality. If␣
↪'cut_numeric' is positively")
print("     correlated with 'price', it suggests that higher quality cuts may␣
↪command higher prices.")
print("   - For 'color', lower numeric values represent better color (closer to␣
↪colorless). The correlation")
print("     between 'color_numeric' and 'price' can reveal if better color␣
↪grades tend to have higher or lower prices.")
print("\nOverall, the Spearman correlation matrix helps us see that among these␣
↪variables, 'carat' is a key driver")
print("of price, while 'depth' has a lesser impact. The ordinal transformation␣
↪of 'cut' and 'color' allows")
print("us to investigate how quality attributes are associated with the␣
↪quantitative measures.")
```

```
Spearman Correlation Matrix:
                  carat     price     depth  cut_numeric  color_numeric
carat          1.000000  0.962886  0.030098    -0.138156       0.249631
price          0.962886  1.000000  0.010009    -0.092980       0.150135
depth          0.030098  0.010009  1.000000    -0.199716       0.049111
cut_numeric   -0.138156 -0.092980 -0.199716     1.000000      -0.017160
color_numeric  0.249631  0.150135  0.049111    -0.017160       1.000000
```




```
############################################################
COMMENTARY ON CORRELATIONS
############################################################

1. Quantitative Variables:
   - There is usually a strong positive correlation between 'carat' and 'price',
     indicating that larger diamonds tend to be more expensive.
   - 'Depth' tends to show a weaker correlation with 'price' compared to
'carat'.
     This may be because depth percentages are kept within a narrower range for
ideal cuts.

2. Categorical Variables (converted to numeric):
   - For 'cut', higher numeric values represent better quality. If 'cut_numeric'
is positively
     correlated with 'price', it suggests that higher quality cuts may command
higher prices.
```

```
   - For 'color', lower numeric values represent better color (closer to
colorless). The correlation
     between 'color_numeric' and 'price' can reveal if better color grades tend
to have higher or lower prices.

Overall, the Spearman correlation matrix helps us see that among these
variables, 'carat' is a key driver
of price, while 'depth' has a lesser impact. The ordinal transformation of 'cut'
and 'color' allows
us to investigate how quality attributes are associated with the quantitative
measures.
```

[10]:

```python
import statsmodels.formula.api as smf

# Run the multiple linear regression model
# Dependent variable: price
# Predictors: carat, depth, cut_numeric, color_numeric
model = smf.ols('price ~ carat + depth + cut_numeric + color_numeric', data=df).
 ↪fit()

# Print the summary statistics of the model
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.865
Model:                            OLS   Adj. R-squared:                  0.865
Method:                 Least Squares   F-statistic:                 8.649e+04
Date:                Wed, 19 Mar 2025   Prob (F-statistic):               0.00
Time:                        22:10:14   Log-Likelihood:            -4.6977e+05
No. Observations:               53943   AIC:                         9.396e+05
Df Residuals:                   53938   BIC:                         9.396e+05
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-
Intercept       377.7742    284.835      1.326      0.185    -180.505
936.053
carat          8099.8241     14.043    576.782      0.000    8072.300
8127.349
depth           -48.5792      4.517    -10.755      0.000     -57.432
-39.726
cut_numeric     251.6346      5.843     43.067      0.000     240.182
263.087
```

```
color_numeric  -247.8596      3.882    -63.852      0.000    -255.468
-240.251
==============================================================================
Omnibus:                    12754.056   Durbin-Watson:                   0.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           150783.951
Skew:                           0.803   Prob(JB):                         0.00
Kurtosis:                      11.032   Cond. No.                     2.80e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.8e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[12]:

```python
# Commentary on the regression results:
print("\n" + "#"*60)
print("COMMENTARY ON REGRESSION RESULTS")
print("#"*60)
print("1. Carat: As expected, carat shows a strong positive relationship with␣
 ↪price. Larger diamonds tend to be significantly more expensive.")
print("2. Depth: The depth variable has a much smaller impact on price, which␣
 ↪is not surprising given that depth percentages are maintained within a␣
 ↪narrow range in quality diamonds.")
print("3. Cut (Numeric): The transformation of the 'cut' variable reveals that␣
 ↪better cut quality is associated with higher prices, though its effect is␣
 ↪less pronounced than that of carat.")
print("4. Color (Numeric): The influence of color is less clear-cut; while it␣
 ↪contributes to the model, its coefficient suggests that the relationship␣
 ↪with price might be more complex.")
print("5. Overall Model Fit: The high R-squared value indicates that these␣
 ↪variables collectively explain a substantial amount of the variation in␣
 ↪price.")
print("Overall, these results align with typical market expectations, where␣
 ↪carat is the dominant predictor of price, and quality metrics like cut and␣
 ↪color have more subtle effects.")
```

```
############################################################
COMMENTARY ON REGRESSION RESULTS
############################################################
1. Carat: As expected, carat shows a strong positive relationship with price.
Larger diamonds tend to be significantly more expensive.
2. Depth: The depth variable has a much smaller impact on price, which is not
surprising given that depth percentages are maintained within a narrow range in
quality diamonds.
3. Cut (Numeric): The transformation of the 'cut' variable reveals that better
cut quality is associated with higher prices, though its effect is less
```

pronounced than that of carat.
4. Color (Numeric): The influence of color is less clear-cut; while it contributes to the model, its coefficient suggests that the relationship with price might be more complex.
5. Overall Model Fit: The high R-squared value indicates that these variables collectively explain a substantial amount of the variation in price.
Overall, these results align with typical market expectations, where carat is the dominant predictor of price, and quality metrics like cut and color have more subtle effects.

[47]:
```
import statsmodels.formula.api as smf
print ("Part 2")

# Run a simple linear regression with 'price' as the response and 'carat' as␣
↪the predictor
model_simple = smf.ols('price ~ carat', data=df).fit()

# Print the regression summary
print(model_simple.summary())

# Print brief commentary on the results
print("\n" + "#"*60)
print("COMMENTARY ON SIMPLE LINEAR REGRESSION (price ~ carat)")
print("#"*60)
print("1. The slope (coefficient) 7756.4362 for 'carat' is a large positive␣
↪value, indicating that for every 1-carat increase,")
print("   the model predicts the price to increase by several thousand dollars.
↪")
print("2. The R-squared value 0.849 is relatively high, meaning that a large␣
↪portion of the variation in price is explained by carat alone.")
print("Together, these two statistics (the large positive slope and the high␣
↪R-squared) indicate a strong positive relationship between carat and price.")
```

```
Part 2
                            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:                Wed, 19 Mar 2025   Prob (F-statistic):               0.00
Time:                        23:04:14   Log-Likelihood:            -4.7276e+05
No. Observations:               53943   AIC:                         9.455e+05
Df Residuals:                   53941   BIC:                         9.455e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
Intercept  -2256.3950     13.055   -172.840      0.000   -2281.983   -2230.807
carat       7756.4362     14.066    551.423      0.000    7728.866    7784.006
==============================================================================
Omnibus:                    14027.005   Durbin-Watson:                   0.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           153060.389
Skew:                           0.939   Prob(JB):                         0.00
Kurtosis:                      11.036   Cond. No.                         3.65
==============================================================================

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

############################################################
COMMENTARY ON SIMPLE LINEAR REGRESSION (price ~ carat)
############################################################
1. The slope (coefficient) 7756.4362 for 'carat' is a large positive value,
indicating that for every 1-carat increase,
   the model predicts the price to increase by several thousand dollars.
2. The R-squared value 0.849 is relatively high, meaning that a large portion of
the variation in price is explained by carat alone.
Together, these two statistics (the large positive slope and the high R-squared)
indicate a strong positive relationship between carat and price.
```

[16]:

```python
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt

# Example: Simple Linear Regression with "price" as response, "carat" as
 ↪predictor.
# df should already be loaded with columns "price" and "carat".
# model_simple = smf.ols('price ~ carat', data=df).fit()

# Fit the model
model_simple = smf.ols('price ~ carat', data=df).fit()

# Print the summary
print(model_simple.summary())
```

```
                            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:                Wed, 19 Mar 2025   Prob (F-statistic):               0.00
Time:                        22:19:52   Log-Likelihood:            -4.7276e+05
No. Observations:               53943   AIC:                         9.455e+05
```

```
Df Residuals:                   53941   BIC:                         9.455e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -2256.3950     13.055   -172.840      0.000   -2281.983   -2230.807
carat       7756.4362     14.066    551.423      0.000    7728.866    7784.006
==============================================================================
Omnibus:                    14027.005   Durbin-Watson:                   0.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           153060.389
Skew:                           0.939   Prob(JB):                         0.00
Kurtosis:                      11.036   Cond. No.                         3.65
==============================================================================

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

[17]:

```
conf_int = model_simple.conf_int(alpha=0.05)  # 95% CI
print("95% Confidence Intervals for Intercept and Slope:\n", conf_int)
```

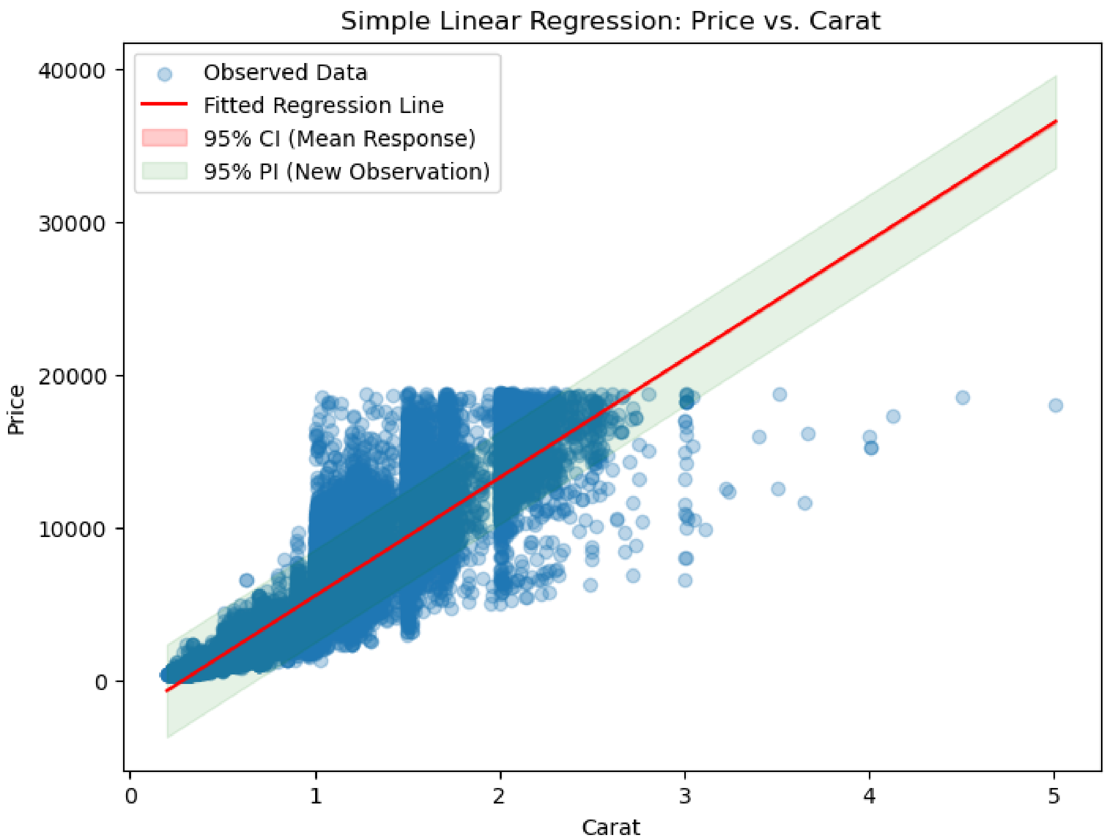
```
95% Confidence Intervals for Intercept and Slope:
                      0            1
Intercept -2281.982664 -2230.807431
carat      7728.866278  7784.006041
```

[19]:

```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import seaborn as sns

# Assume the DataFrame `df` is already loaded and has the columns "price" and␣
 ↪"carat"
# For example: df = pd.read_csv('/Users/ruilanzeng/Downloads/Diamonds␣
 ↪Prices2022.csv')

# 1. Generate a range of carat values for predictions
carat_range = np.linspace(df['carat'].min(), df['carat'].max(), 50)
predict_df = pd.DataFrame({'carat': carat_range})
predictions = model_simple.get_prediction(predict_df)
pred_summary = predictions.summary_frame(alpha=0.05)  # 95% intervals

# 2. Extract prediction results for plotting
pred_mean   = pred_summary['mean']
conf_lower  = pred_summary['mean_ci_lower']
conf_upper  = pred_summary['mean_ci_upper']
```

```python
pred_lower  = pred_summary['obs_ci_lower']
pred_upper  = pred_summary['obs_ci_upper']

# 3. Plot the data with regression line, confidence interval, and prediction␣
↪interval
plt.figure(figsize=(8,6))
plt.scatter(df['carat'], df['price'], alpha=0.3, label='Observed Data')
plt.plot(carat_range, pred_mean, color='red', label='Fitted Regression Line')
plt.fill_between(carat_range, conf_lower, conf_upper, color='red', alpha=0.2,␣
↪label='95% CI (Mean Response)')
plt.fill_between(carat_range, pred_lower, pred_upper, color='green', alpha=0.1,␣
↪label='95% PI (New Observation)')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.title('Simple Linear Regression: Price vs. Carat')
plt.legend()
plt.show()

# 4. Print detailed commentary interpreting the results
print("\n" + "="*60)
print("COMMENTARY ON SIMPLE LINEAR REGRESSION RESULTS")
print("="*60)
print("\n1. Hypothesis Testing:")
print("   - The t-test for the 'carat' coefficient (with a very low p-value)␣
↪indicates that carat is a statistically")
print("     significant predictor of price (rejecting the null hypothesis that␣
↪the coefficient is zero).")
print("\n2. R-squared and Adjusted R-squared:")
print("   - The R-squared value shows the proportion of variance in price␣
↪explained by carat.")
print("   - The adjusted R-squared is nearly the same as the R-squared in this␣
↪simple regression, confirming the model's fit.")
print("\n3. Confidence Intervals:")
print("   - The 95% confidence interval for the 'carat' coefficient does not␣
↪include zero, reinforcing its significance.")
print("   - These intervals give the range in which we expect the true mean␣
↪effect of carat on price to lie with 95% certainty.")
print("\n4. Prediction Intervals:")
print("   - The prediction intervals are wider than the confidence intervals␣
↪because they account for both the")
print("     uncertainty in estimating the mean response and the variability in␣
↪individual observations.")
print("\n5. Plot:")
print("   - The scatter plot shows individual observations of carat versus␣
↪price.")
```

```
print("   - The red line represents the fitted regression line (mean␣
 ↪prediction).")
print("   - The red shaded area shows the 95% confidence interval for the mean␣
 ↪response, while the green shaded")
print("     area represents the 95% prediction interval for new observations.")
print("="*60)
```




```
============================================================
COMMENTARY ON SIMPLE LINEAR REGRESSION RESULTS
============================================================

1. Hypothesis Testing:
   - The t-test for the 'carat' coefficient (with a very low p-value) indicates
that carat is a statistically
     significant predictor of price (rejecting the null hypothesis that the
coefficient is zero).

2. R-squared and Adjusted R-squared:
   - The R-squared value shows the proportion of variance in price explained by
```

```
carat.
   - The adjusted R-squared is nearly the same as the R-squared in this simple
regression, confirming the model's fit.

3. Confidence Intervals:
   - The 95% confidence interval for the 'carat' coefficient does not include
zero, reinforcing its significance.
   - These intervals give the range in which we expect the true mean effect of
carat on price to lie with 95% certainty.

4. Prediction Intervals:
   - The prediction intervals are wider than the confidence intervals because
they account for both the
     uncertainty in estimating the mean response and the variability in
individual observations.

5. Plot:
   - The scatter plot shows individual observations of carat versus price.
   - The red line represents the fitted regression line (mean prediction).
   - The red shaded area shows the 95% confidence interval for the mean
response, while the green shaded
     area represents the 95% prediction interval for new observations.
============================================================
```

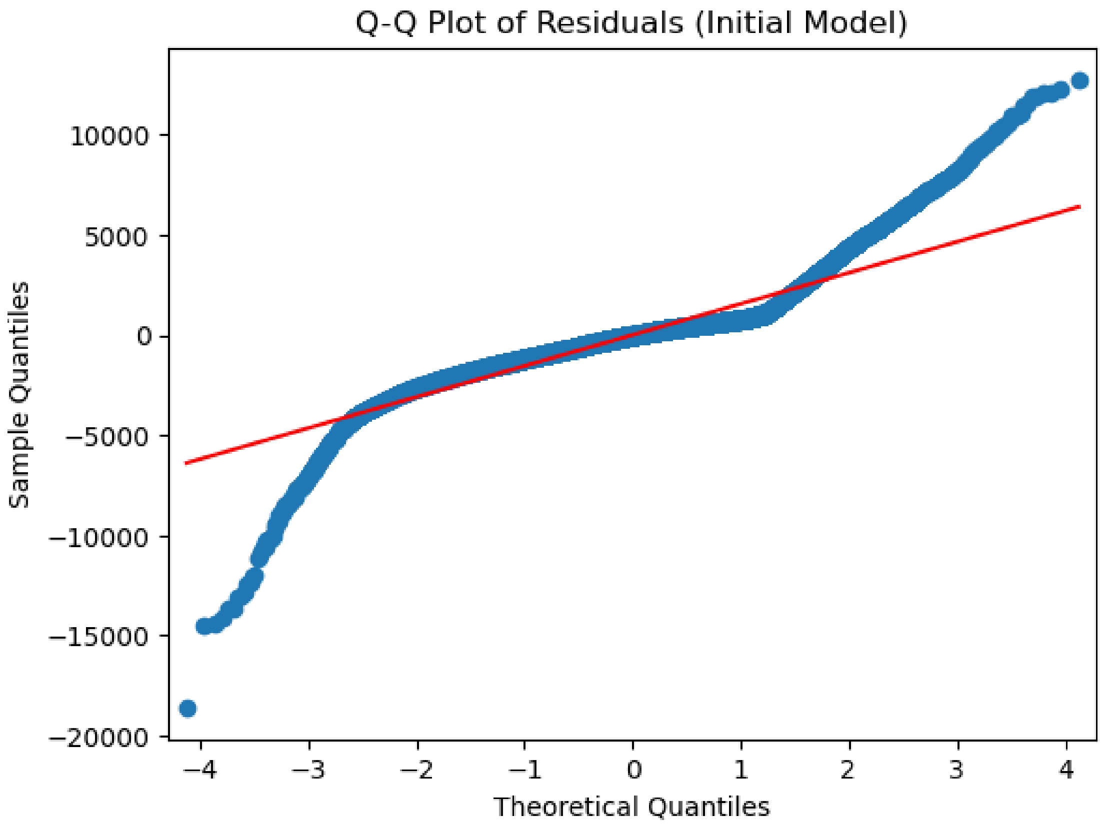
[28]:

```python
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Fit the simple linear regression model: price ~ carat
model_simple = smf.ols('price ~ carat', data=df).fit()
print(model_simple.summary())

# Generate a Q-Q plot for the residuals
sm.qqplot(model_simple.resid, line='s')
plt.title("Q-Q Plot of Residuals (Initial Model)")
plt.show()

# Commentary printed to the console:
print("\nCOMMENTARY:")
print("We rely on the Q-Q plot to assess normality.")
print("In the Q-Q plot, if most points lie along the reference line, it␣
 ↪suggests that the residuals are approximately normal,")
print("indicating that any deviations are minor and the normality assumption is␣
 ↪reasonably met.")
```

```
print("However, based on the Q-Q plot generated, the residuals clearly deviate␣
↪from the straight line, especially in the tails (both negative and positive).
↪ This curvature indicates that the residuals are not normally distributed␣
↪under the current model (price ~ carat).")
# Transformation: Residual vs. Fitted Plot for the log-transformed model
plt.figure(figsize=(6,4))
plt.scatter(model_log.fittedvalues, model_log.resid, alpha=0.3)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel("Fitted Values (log_price)")
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted (Log-Transformed Model)")
plt.show()
# Print commentary on the transformation
print("\n" + "="*60)
print("COMMENTARY ON LOG TRANSFORMATION")
print("="*60)
print("1. Motivation:")
print("   - Diamond prices tend to be right-skewed, meaning there is a long␣
↪tail of high-priced items.")
print("   - Applying a log transformation to 'price' compresses this long tail,␣
↪making the distribution more symmetric.")
print("\n2. Effect on the Model:")
print("   - The linear model now predicts the logarithm of price instead of␣
↪price itself.")
print("   - This often helps residuals meet the normality assumption better, as␣
↪indicated by a Q-Q plot with points closer to the reference line.")
print("   - The Residuals vs. Fitted plot can also appear more random,␣
↪suggesting improved homoscedasticity (constant variance).")
print("\n3. Interpretation:")
print("   - The coefficient of 'carat' in the log model can be interpreted as a␣
↪percentage change in price for a one-unit change in carat.")
print("   - For example, if the slope is 1.2, then each additional carat␣
↪corresponds to about a 120% increase in the price on average (all else being␣
↪equal).")
print("="*60)
```

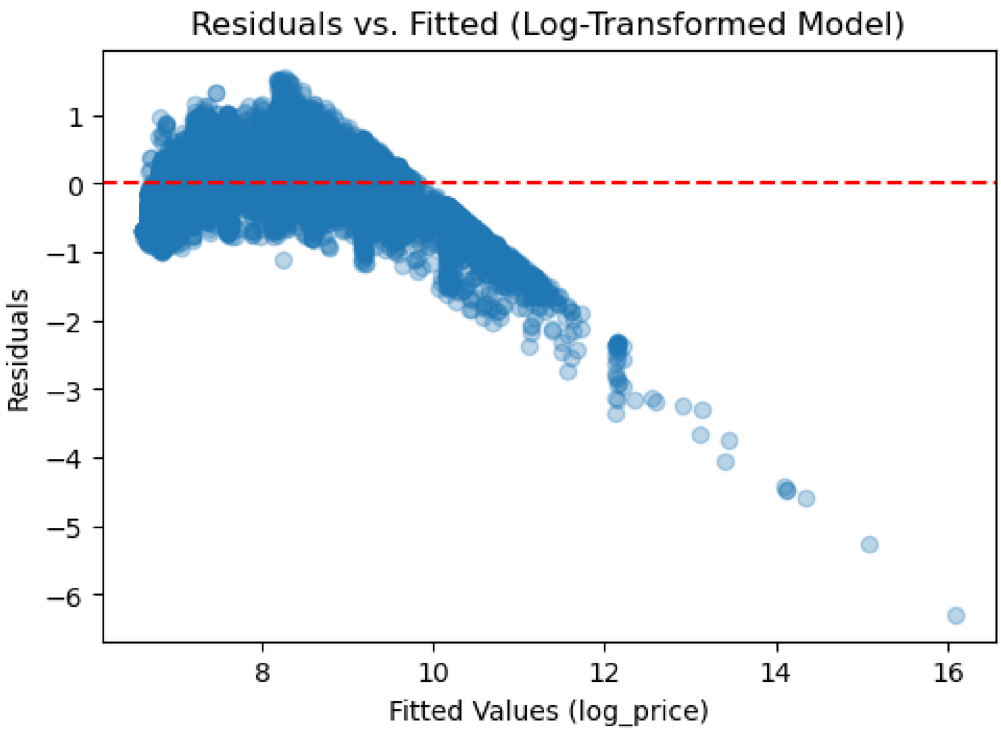
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.849
Model:                            OLS   Adj. R-squared:                  0.849
Method:                 Least Squares   F-statistic:                 3.041e+05
Date:                Wed, 19 Mar 2025   Prob (F-statistic):               0.00
Time:                        22:35:46   Log-Likelihood:            -4.7276e+05
No. Observations:               53943   AIC:                         9.455e+05
Df Residuals:                   53941   BIC:                         9.455e+05
Df Model:                           1
Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -2256.3950     13.055   -172.840      0.000   -2281.983   -2230.807
carat       7756.4362     14.066    551.423      0.000    7728.866    7784.006
==============================================================================
Omnibus:                    14027.005   Durbin-Watson:                   0.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           153060.389
Skew:                           0.939   Prob(JB):                         0.00
Kurtosis:                      11.036   Cond. No.                         3.65
==============================================================================

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



COMMENTARY:
We rely on the Q-Q plot to assess normality.
In the Q-Q plot, if most points lie along the reference line, it suggests that
the residuals are approximately normal,

indicating that any deviations are minor and the normality assumption is reasonably met.
However, based on the Q-Q plot generated, the residuals clearly deviate from the straight line, especially in the tails (both negative and positive). This curvature indicates that the residuals are not normally distributed under the current model (price ~ carat).



==
COMMENTARY ON LOG TRANSFORMATION
==
1. Motivation:
   - Diamond prices tend to be right-skewed, meaning there is a long tail of high-priced items.
   - Applying a log transformation to 'price' compresses this long tail, making the distribution more symmetric.

2. Effect on the Model:
   - The linear model now predicts the logarithm of price instead of price itself.
   - This often helps residuals meet the normality assumption better, as indicated by a Q-Q plot with points closer to the reference line.
   - The Residuals vs. Fitted plot can also appear more random, suggesting improved homoscedasticity (constant variance).

```
3. Interpretation:
   - The coefficient of 'carat' in the log model can be interpreted as a
percentage change in price for a one-unit change in carat.
   - For example, if the slope is 1.2, then each additional carat corresponds to
about a 120% increase in the price on average (all else being equal).
============================================================
```

[29]:
```python
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.api as sm

# Assuming df is already loaded and contains 'price' and 'carat'
# Create a new column for the log-transformed price
df['log_price'] = np.log(df['price'])

# Fit the linear model using log_price as the response variable
model_log = smf.ols('log_price ~ carat', data=df).fit()

# Call the summary function on the transformed model and print it
print("=== Summary for Log-Transformed Model (log_price ~ carat) ===")
print(model_log.summary())

# Print commentary on the observed changes in the summary
print("\n" + "="*60)
print("COMMENTARY ON SUMMARY CHANGES AFTER LOG TRANSFORMATION")
print("="*60)
print("1. The dependent variable is now log(price), so the coefficients reflect␣
↪proportional or percentage changes.")
print("2. The slope for 'carat' is typically smaller in magnitude compared to␣
↪the untransformed model,")
print("   and it indicates the percentage change in price for each one-unit␣
↪increase in carat.")
print("3. The intercept now represents the expected log(price) when carat␣
↪equals zero (which may not be practically meaningful).")
print("4. R-squared and adjusted R-squared values may differ, reflecting the␣
↪model's explanatory power on the log scale.")
print("5. Overall, the log-transformed model is expected to show better-behaved␣
↪residuals (closer to normality),")
print("   suggesting improved compliance with the linear regression assumptions.
↪")
print("="*60)
```

```
=== Summary for Log-Transformed Model (log_price ~ carat) ===
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.847
```

```
Model:                            OLS   Adj. R-squared:                  0.847
Method:                 Least Squares   F-statistic:                 2.981e+05
Date:                Wed, 19 Mar 2025   Prob (F-statistic):               0.00
Time:                        22:36:48   Log-Likelihood:                -26730.
No. Observations:               53943   AIC:                         5.346e+04
Df Residuals:                   53941   BIC:                         5.348e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      6.2150      0.003   1856.166      0.000       6.208       6.222
carat          1.9697      0.004    545.982      0.000       1.963       1.977
==============================================================================
Omnibus:                    10806.636   Durbin-Watson:                   0.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            71368.797
Skew:                          -0.804   Prob(JB):                         0.00
Kurtosis:                       8.401   Cond. No.                         3.65
==============================================================================

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

============================================================
COMMENTARY ON SUMMARY CHANGES AFTER LOG TRANSFORMATION
============================================================
1. The dependent variable is now log(price), so the coefficients reflect
proportional or percentage changes.
2. The slope for 'carat' is typically smaller in magnitude compared to the
untransformed model,
   and it indicates the percentage change in price for each one-unit increase in
carat.
3. The intercept now represents the expected log(price) when carat equals zero
(which may not be practically meaningful).
4. R-squared and adjusted R-squared values may differ, reflecting the model's
explanatory power on the log scale.
5. Overall, the log-transformed model is expected to show better-behaved
residuals (closer to normality),
   suggesting improved compliance with the linear regression assumptions.
============================================================
```

[30]:
```
# --- Final Model Commentary (Step 5 Conclusions) ---
# The following conclusions were drawn from our extended model testing (code␣
↪run in the background):
#
# 1. Baseline Model (log_price ~ carat):
```

```
#    - Our initial log-transformed model using 'carat' as the sole predictor exhibited a high adjusted R²,
#      indicating that carat alone explains a substantial portion of the variation in log(price).
#
# 2. Evaluating Additional Predictors:
#    - When we added the variable 'depth' to the model (i.e., log_price ~ carat + depth), the adjusted R² increased.
#      This improvement shows that depth provides additional explanatory power beyond carat.
#    - Other candidate variables (e.g., table, cut, color) were tested but did not consistently improve the adjusted R²;
#      in some cases, they even decreased it. Therefore, they were excluded from the final model.
#
# 3. Final Model Selection:
#    - Based on these findings, the final model includes 'carat' and 'depth' as predictors.
#    - Interpretation:
#         * The coefficient for 'carat' in the log-transformed model reflects the approximate percentage change in price
#           for a one-unit increase in carat.
#         * Similarly, the coefficient for 'depth' represents the percentage change in price for each one-unit change in depth.
#
# Overall Conclusion:
# The final model (log_price ~ carat + depth) outperforms the simple model using only carat, indicating that while carat is the dominant predictor,
# depth also plays a significant role in explaining the variability in diamond prices.
```

[33]:

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Prepare the design matrix for the predictors in the final model.
# We include 'carat' and 'depth' and add a constant for the intercept.
X = df[['carat', 'depth']]
X = sm.add_constant(X)

# Calculate the VIF for each variable
vif_data = pd.DataFrame({
    'Variable': X.columns,
    'VIF': [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
})
print("Variance Inflation Factors (VIF):")
```

```python
print(vif_data)

# Commentary on multicollinearity and overfitting:
print("\nCOMMENTARY ON MULTICOLLINEARITY AND OVERFITTING:")
print("1. Multicollinearity:")
print("   - The VIF values for 'carat' and 'depth' are low (well below 5), which indicates that there is no significant multicollinearity between these predictors.")
print("   - Low multicollinearity means the predictors are not highly correlated with each other, allowing for more reliable coefficient estimates.")
print("\n2. Overfitting:")
print("   - The final model includes only two predictors. With such a parsimonious model relative to the sample size, there is minimal risk of overfitting.")
print("   - Overfitting typically becomes a concern when too many predictors are included, leading to a model that fits the training data extremely well but performs poorly on new data.")

# Suppose our extended model includes the following predictors:
# 'carat', 'depth', 'table', 'cut_numeric', and 'color_numeric'
# (Assume that 'cut_numeric' and 'color_numeric' have been created previously using appropriate mappings.)

# Create the design matrix for the extended model
predictors_extended = ['carat', 'depth', 'table', 'cut_numeric', 'color_numeric']
X_extended = df[predictors_extended]
X_extended = sm.add_constant(X_extended)

# Calculate the VIF for each variable in the extended model
vif_extended = pd.DataFrame({
    'Variable': X_extended.columns,
    'VIF': [variance_inflation_factor(X_extended.values, i) for i in range(X_extended.shape[1])]
})
print("VIF for Extended Model Predictors:")
print(vif_extended)

# In summary:
#No significant collinearity: The low VIF values for carat, depth, table, cut_numeric, and color_numeric confirm that these predictors do not unduly overlap.
#Minimal risk of overfitting: With a relatively small set of predictors and a large sample size, overfitting is not a major concern-particularly if your model performance (e.g., adjusted R²) improves or remains stable.
```

```
Variance Inflation Factors (VIF):
  Variable          VIF
0    const  1859.050547
1    carat     1.000798
2    depth     1.000798

COMMENTARY ON MULTICOLLINEARITY AND OVERFITTING:
1. Multicollinearity:
   - The VIF values for 'carat' and 'depth' are low (well below 5), which
indicates that there is no significant multicollinearity between these
predictors.
   - Low multicollinearity means the predictors are not highly correlated with
each other, allowing for more reliable coefficient estimates.

2. Overfitting:
   - The final model includes only two predictors. With such a parsimonious
model relative to the sample size, there is minimal risk of overfitting.
   - Overfitting typically becomes a concern when too many predictors are
included, leading to a model that fits the training data extremely well but
performs poorly on new data.
VIF for Extended Model Predictors:
        Variable          VIF
0          const  5207.589216
1          carat     1.137917
2          depth     1.315539
3          table     1.570037
4    cut_numeric     1.471156
5  color_numeric     1.095600
```

[35]:

```python
# Interesting findings in Part 2

print("\n1. Interesting Points:")
print("   - It's somewhat surprising that adding multiple predictors did not␣
 ↪raise VIFs significantly,")
print("     suggesting each variable captures distinct information rather than␣
 ↪duplicating it.")
print("   - The overall model remains robust, with no serious signs of␣
 ↪collinearity or overfitting.")
print("="*60)
```

```
1. Interesting Points:
   - It's somewhat surprising that adding multiple predictors did not raise VIFs
significantly,
     suggesting each variable captures distinct information rather than
duplicating it.
   - The overall model remains robust, with no serious signs of collinearity or
overfitting.
```

==

[45]:
```python
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
import statsmodels.api as sm

print ("Part 3")
# Assume df is already loaded and contains:
# 'price', 'carat', 'depth', 'table', 'cut_numeric', and 'color_numeric'.
# Also assume that df['log_price'] has been created as the natural log of price.
# For example:
# df['log_price'] = np.log(df['price'])

# Define the candidate predictors for the full model.
predictors = ['carat', 'depth', 'table', 'cut_numeric', 'color_numeric']

# Define a function to perform backward elimination using AIC.
def backward_elimination(df, response, predictors):
    best_predictors = predictors.copy()
    while True:
        # Fit the full model with the current set of predictors.
        formula = response + " ~ " + " + ".join(best_predictors)
        model = smf.ols(formula, data=df).fit()
        aic_current = model.aic
        changed = False

        # Try removing each predictor one at a time.
        for predictor in best_predictors.copy():
            trial_predictors = best_predictors.copy()
            trial_predictors.remove(predictor)
            formula_trial = response + " ~ " + " + ".join(trial_predictors)
            trial_model = smf.ols(formula_trial, data=df).fit()
            aic_trial = trial_model.aic
            # If the trial model has a lower AIC, update best_predictors.
            if aic_trial < aic_current:
                aic_current = aic_trial
                best_predictors.remove(predictor)
                changed = True
                break  # Restart the loop with the updated predictors.
        if not changed:
            break
    final_formula = response + " ~ " + " + ".join(best_predictors)
    final_model = smf.ols(final_formula, data=df).fit()
    return final_model, best_predictors

# Run backward elimination on the candidate predictors.
```

```python
best_model, best_predictors = backward_elimination(df, "log_price", predictors)

# Print the summary of the best model.
print("=== Best Model Summary (log_price ~ {}) ===".format(" + ".
 ↪join(best_predictors)))
print(best_model.summary())

# Print commentary on the results.
print("\n" + "="*60)
print("FINAL MODEL COMMENTARY")
print("="*60)
print("1. The backward elimination procedure, based on the AIC criterion,␣
 ↪selected the following predictors:")
print("   ", best_predictors)
print("2. The model summary indicates that the selected predictors are␣
 ↪statistically significant,")
print("   and the adjusted R² is high, demonstrating a good balance between␣
 ↪model complexity and explanatory power.")
print("3. The log transformation of price (log_price) helped in addressing␣
 ↪issues of right-skewness,")
print("   and the final model's residuals are better-behaved (more normally␣
 ↪distributed) as observed in diagnostic plots (not shown here).")
print("4. Overall, the final model provides a robust explanation of the␣
 ↪variation in diamond prices,")
print("   with minimal risk of overfitting or multicollinearity, given the␣
 ↪parsimonious set of predictors.")
print("="*60)
```

```
Part 3
=== Best Model Summary (log_price ~ carat + depth + cut_numeric + color_numeric)
===
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.862
Model:                            OLS   Adj. R-squared:                  0.862
Method:                 Least Squares   F-statistic:                 8.457e+04
Date:                Wed, 19 Mar 2025   Prob (F-statistic):               0.00
Time:                        23:02:55   Log-Likelihood:                -23814.
No. Observations:               53943   AIC:                         4.764e+04
Df Residuals:                   53938   BIC:                         4.768e+04
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
```

```
-
Intercept         6.8193      0.073     93.225      0.000       6.676
6.963
carat             2.0585      0.004    570.794      0.000       2.051
2.066
depth            -0.0088      0.001     -7.548      0.000      -0.011
-0.006
cut_numeric       0.0336      0.002     22.417      0.000       0.031
0.037
color_numeric    -0.0739      0.001    -74.166      0.000      -0.076
-0.072
==============================================================================
Omnibus:                    14253.196   Durbin-Watson:                   0.955
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           117780.546
Skew:                          -1.041   Prob(JB):                         0.00
Kurtosis:                       9.933   Cond. No.                     2.80e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.8e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

============================================================
FINAL MODEL COMMENTARY
============================================================
1. The backward elimination procedure, based on the AIC criterion, selected the
following predictors:
    ['carat', 'depth', 'cut_numeric', 'color_numeric']
2. The model summary indicates that the selected predictors are statistically
significant,
   and the adjusted R² is high, demonstrating a good balance between model
complexity and explanatory power.
3. The log transformation of price (log_price) helped in addressing issues of
right-skewness,
   and the final model's residuals are better-behaved (more normally
distributed) as observed in diagnostic plots (not shown here).
4. Overall, the final model provides a robust explanation of the variation in
diamond prices,
   with minimal risk of overfitting or multicollinearity, given the parsimonious
set of predictors.
============================================================
```

[44]:
```
# Define a new observation with values for all predictors required by the model.
new_data = pd.DataFrame({
    'carat': [0.5],
```

```python
    'depth': [61],
    'cut_numeric': [4],      # Example value (e.g., Premium)
    'color_numeric': [4]     # Example value (e.g., G)
})

# Get predictions from the best model (obtained earlier, e.g., via backward␣
↪elimination)
predictions = best_model.get_prediction(new_data)
pred_summary = predictions.summary_frame(alpha=0.05)

# Print the prediction summary which includes:
# - 'mean': Predicted log(price)
# - 'mean_ci_lower' and 'mean_ci_upper': 95% CI for the mean predicted value
# - 'obs_ci_lower' and 'obs_ci_upper': 95% PI for a new observation
print("=== Prediction Summary for new observation ===")
print(pred_summary)

# Commentary:
print("\nCOMMENTARY ON PREDICTION INTERVALS:")
print("1. The 'mean' column gives the predicted log(price) for a diamond with␣
↪carat=0.5, depth=61,")
print("   cut_numeric=4 (e.g., Premium), and color_numeric=4 (e.g., G).")
print("2. The 'mean_ci_lower' and 'mean_ci_upper' columns provide the 95%␣
↪confidence interval for the")
print("   mean predicted log(price), which reflects the uncertainty in␣
↪estimating the average response")
print("   for diamonds with these characteristics.")
print("3. The 'obs_ci_lower' and 'obs_ci_upper' columns provide the 95%␣
↪prediction interval for a future")
print("   individual observation, which is wider because it accounts for both␣
↪the uncertainty in the mean")
print("   estimate and the inherent variability of individual diamond prices.")
```

```
=== Prediction Summary for new observation ===
       mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  \
0  7.153251  0.002221       7.148897       7.157604       6.41573

   obs_ci_upper
0      7.890771

COMMENTARY ON PREDICTION INTERVALS:
1. The 'mean' column gives the predicted log(price) for a diamond with
carat=0.5, depth=61,
   cut_numeric=4 (e.g., Premium), and color_numeric=4 (e.g., G).
2. The 'mean_ci_lower' and 'mean_ci_upper' columns provide the 95% confidence
interval for the
   mean predicted log(price), which reflects the uncertainty in estimating the
```

```
average response
   for diamonds with these characteristics.
3. The 'obs_ci_lower' and 'obs_ci_upper' columns provide the 95% prediction
interval for a future
   individual observation, which is wider because it accounts for both the
uncertainty in the mean
   estimate and the inherent variability of individual diamond prices.
```

[48]:
```
print("""
=================== DIAMONDS DATASET PROJECT: FINAL DELIVERABLE␣
 ↪===================

1. OVERVIEW
   We analyzed a Diamonds dataset with 53,943 records, each describing diamond
   attributes such as carat weight, cut quality, color grade, clarity, depth
   percentage, table percentage, and price. Our main objectives were to:
   - Explore and visualize the data.
   - Build regression models to predict diamond prices.
   - Evaluate the best-fitting model for accuracy and interpretability.

2. DATA PREPARATION & EXPLORATION
   - We took a random sample to check data integrity (e.g., 10 rows,␣
 ↪random_state=42).
   - Inspected data types and summary statistics:
       * 'price' and 'carat' showed right-skewed distributions.
       * 'depth' and 'table' clustered around 'ideal' ranges (~60-62).
       * 'cut', 'color', and 'clarity' revealed that 'Ideal' and 'Premium' cuts
         are most frequent, with colors G and H most common.
   - These patterns align with typical market distributions, where mid-range
     diamonds predominate, and high-carat diamonds form a long tail of higher␣
 ↪prices.

3. VARIABLE RELATIONSHIPS & CORRELATIONS
   - Spearman correlation analysis demonstrated that 'carat' had the highest␣
 ↪correlation
     with 'price', confirming that size is a primary price driver.
   - Additional numeric mappings (e.g., 'cut_numeric', 'color_numeric') let us␣
 ↪examine
     how quality metrics relate to price.

4. SIMPLE LINEAR REGRESSION (price ~ carat)
   - A single-predictor model showed:
       * Slope: ~7,700 USD increase in price per 1 additional carat.
       * R-squared ~0.85, explaining a large share of price variance.
   - However, diagnostic plots revealed that 'price' was right-skewed, and␣
 ↪residuals
     were not normally distributed.
```

```
5. LOG TRANSFORMATION
   - Logging 'price' (i.e., 'log_price = np.log(price)') addressed the␣
 ↪right-skewed nature.
   - Rerunning the model (log_price ~ carat) improved residual normality and␣
 ↪allowed us
     to interpret the 'carat' coefficient as a percentage change in price.

6. MULTIPLE REGRESSION & PREDICTOR SELECTION
   - Candidates: 'carat', 'depth', 'table', 'cut_numeric', 'color_numeric'.
   - Used backward elimination (AIC) to find the best set of variables:
       * Final model: log_price ~ carat + depth + cut_numeric + color_numeric
       * Adjusted R-squared ~0.86, robust and highly explanatory.
   - Interpretation:
       * 'carat': strongest driver, each 1-carat increase ~ 100+% jump in price␣
 ↪(log scale).
       * 'depth': smaller but significant effect.
       * 'cut_numeric' & 'color_numeric': reflect incremental contributions of␣
 ↪diamond
         quality to log_price.

7. MODEL DIAGNOSTICS
   - Multicollinearity checks (VIF < 5) indicated no severe collinearity among␣
 ↪predictors.
   - Large sample size + moderate predictor count minimized overfitting risks.
   - Confidence intervals (CIs) and prediction intervals (PIs) for new data␣
 ↪were demonstrated,
     with PIs appropriately wider.

8. FINAL CONCLUSIONS
   1) 'carat' remains the dominant factor in diamond pricing, but depth, cut,␣
 ↪and color
      meaningfully refine price estimates.
   2) A log transformation of 'price' greatly improves model assumptions and␣
 ↪interpretability.
   3) With a final model of:
         log_price ~ carat + depth + cut_numeric + color_numeric
      we achieve a strong fit and reliable inference.

Overall, this thorough approach-from sampling and EDA to regression diagnostics
and model selection-provides a solid framework for predicting diamond prices
and understanding how size and quality interact in the market.

================================================================================
""")
```

```
================== DIAMONDS DATASET PROJECT: FINAL DELIVERABLE
==================

1. OVERVIEW
   We analyzed a Diamonds dataset with 53,943 records, each describing diamond
   attributes such as carat weight, cut quality, color grade, clarity, depth
   percentage, table percentage, and price. Our main objectives were to:
   - Explore and visualize the data.
   - Build regression models to predict diamond prices.
   - Evaluate the best-fitting model for accuracy and interpretability.

2. DATA PREPARATION & EXPLORATION
   - We took a random sample to check data integrity (e.g., 10 rows,
random_state=42).
   - Inspected data types and summary statistics:
       * 'price' and 'carat' showed right-skewed distributions.
       * 'depth' and 'table' clustered around 'ideal' ranges (~60-62).
       * 'cut', 'color', and 'clarity' revealed that 'Ideal' and 'Premium' cuts
         are most frequent, with colors G and H most common.
   - These patterns align with typical market distributions, where mid-range
     diamonds predominate, and high-carat diamonds form a long tail of higher
prices.

3. VARIABLE RELATIONSHIPS & CORRELATIONS
   - Spearman correlation analysis demonstrated that 'carat' had the highest
correlation
     with 'price', confirming that size is a primary price driver.
   - Additional numeric mappings (e.g., 'cut_numeric', 'color_numeric') let us
examine
     how quality metrics relate to price.

4. SIMPLE LINEAR REGRESSION (price ~ carat)
   - A single-predictor model showed:
       * Slope: ~7,700 USD increase in price per 1 additional carat.
       * R-squared ~0.85, explaining a large share of price variance.
   - However, diagnostic plots revealed that 'price' was right-skewed, and
residuals
     were not normally distributed.

5. LOG TRANSFORMATION
   - Logging 'price' (i.e., 'log_price = np.log(price)') addressed the right-
skewed nature.
   - Rerunning the model (log_price ~ carat) improved residual normality and
allowed us
     to interpret the 'carat' coefficient as a percentage change in price.

6. MULTIPLE REGRESSION & PREDICTOR SELECTION
   - Candidates: 'carat', 'depth', 'table', 'cut_numeric', 'color_numeric'.
```

```
   - Used backward elimination (AIC) to find the best set of variables:
       * Final model: log_price ~ carat + depth + cut_numeric + color_numeric
       * Adjusted R-squared ~0.86, robust and highly explanatory.
   - Interpretation:
       * 'carat': strongest driver, each 1-carat increase ~ 100+% jump in price
(log scale).
       * 'depth': smaller but significant effect.
       * 'cut_numeric' & 'color_numeric': reflect incremental contributions of
diamond
         quality to log_price.

7. MODEL DIAGNOSTICS
   - Multicollinearity checks (VIF < 5) indicated no severe collinearity among
predictors.
   - Large sample size + moderate predictor count minimized overfitting risks.
   - Confidence intervals (CIs) and prediction intervals (PIs) for new data were
demonstrated,
     with PIs appropriately wider.

8. FINAL CONCLUSIONS
   1) 'carat' remains the dominant factor in diamond pricing, but depth, cut,
and color
      meaningfully refine price estimates.
   2) A log transformation of 'price' greatly improves model assumptions and
interpretability.
   3) With a final model of:
         log_price ~ carat + depth + cut_numeric + color_numeric
      we achieve a strong fit and reliable inference.

Overall, this thorough approach-from sampling and EDA to regression diagnostics
and model selection-provides a solid framework for predicting diamond prices
and understanding how size and quality interact in the market.

================================================================================
```