1) Import the attached CSV files (Diamond.csv) and answer the following questions:

In [38]:
```
import numpy as np
import pandas as pd
```

In [39]:
```
df = pd.read_csv("Stats Assignment.csv")
```

In [40]:
```
df
```

Out[40]:

	carat	cut	color	clarity	depth	table	weight	size	price
0	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	326
1	0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	326
2	0.23	Good	E	VS1	56.9	65.0	4.05	4.07	327
3	0.29	Premium	I	VS2	62.4	58.0	4.20	4.23	334
4	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	335
...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	5.75	5.76	2757
53936	0.72	Good	D	SI1	63.1	55.0	5.69	5.75	2757
53937	0.70	Very Good	D	SI1	62.8	60.0	5.66	5.68	2757
53938	0.86	Premium	H	SI2	61.0	58.0	6.15	6.12	2757
53939	0.75	Ideal	D	SI2	62.2	55.0	5.83	5.87	2757

53940 rows × 9 columns

A. Create 2 dataframes out of this dataframe – 1 with all numerical variables and other with all categorical variables.

In [41]:
```
num_col = [i for i in df.columns if df[i].dtypes != "object"]
df_num = df[num_col]
df_num
```

Out[41]:

	carat	depth	table	weight	size	price
0	0.23	61.5	55.0	3.95	3.98	326
1	0.21	59.8	61.0	3.89	3.84	326
2	0.23	56.9	65.0	4.05	4.07	327
3	0.29	62.4	58.0	4.20	4.23	334
4	0.31	63.3	58.0	4.34	4.35	335
...	...	...	...	...	...	...
53935	0.72	60.8	57.0	5.75	5.76	2757
53936	0.72	63.1	55.0	5.69	5.75	2757
53937	0.70	62.8	60.0	5.66	5.68	2757
53938	0.86	61.0	58.0	6.15	6.12	2757
53939	0.75	62.2	55.0	5.83	5.87	2757

53940 rows × 6 columns

In [42]:
```
cat_col = [i for i in df.columns if df[i].dtypes == "object"]
df_cat = df[cat_col]
df_cat
```

Out[42]:

	cut	color	clarity
0	Ideal	E	SI2
1	Premium	E	SI1
2	Good	E	VS1
3	Premium	I	VS2
4	Good	J	SI2
...	...	...	...
53935	Ideal	D	SI1
53936	Good	D	SI1
53937	Very Good	D	SI1
53938	Premium	H	SI2
53939	Ideal	D	SI2

53940 rows × 3 columns

B. Calculate the measure of central tendency of numerical variables using Pandas and statistics libraries and check if the calculated values are different between these 2 libraries.

In [43]:
```
# Calculate mean, median, and mode for each column using pandas
mean_pandas = df_num.mean()
median_pandas = df_num.median()
mode_pandas = df_num.mode()

print("Pandas Mean:\n", mean_pandas)

print("\nPandas Median:\n", median_pandas)

print("\nPandas Mode:\n", mode_pandas)
```

```
Pandas Mean:
 carat        0.797940
depth       61.749405
table       57.457184
weight       5.731157
size         5.734526
price     3932.799722
dtype: float64

Pandas Median:
 carat        0.70
depth       61.80
table       57.00
weight       5.70
size         5.71
price     2401.00
dtype: float64

Pandas Mode:
    carat  depth  table  weight  size  price
0    0.3   62.0   56.0    4.37  4.34    605
```

In [44]:
```
import statistics as stats

# Calculate mean, median, and mode for each column using statistics
mean_stats = {col: stats.mean(df_num[col]) for col in df_num.columns}
median_stats = {col: stats.median(df_num[col]) for col in df_num.columns}
mode_stats = {col: stats.mode(df_num[col]) for col in df_num.columns}


print("Statistics Mean:\n", mean_stats)


print("\n Statistics Median:\n", median_stats)


print("\n Statistics Mode:\n", mode_stats)
```

```
Statistics Mean:
 {'carat': 0.7979397478680015, 'depth': 61.74940489432703, 'table': 57.4571839080459
8, 'weight': 5.731157211716722, 'size': 5.734525954764553, 'price': 3932.79972191323
7}

 Statistics Median:
 {'carat': 0.7, 'depth': 61.8, 'table': 57.0, 'weight': 5.7, 'size': 5.71, 'price':
2401.0}

 Statistics Mode:
 {'carat': 0.3, 'depth': 62.0, 'table': 56.0, 'weight': 4.37, 'size': 4.34, 'price':
605}
```

In [45]:
```
# From the above values we found that we get same values of central tendencies irre
```

C) C. Check the skewness of all numeric variables. Mention against each variable if its highly skewed/light skewed/ Moderately skwewed.

In [ ]:
```
```

In [46]:
```
df_num.isnull().sum()
```

Out[46]:
```
carat     0
depth     0
table     0
weight    0
size      0
price     0
dtype: int64
```

In [47]:
```
# Classify skewness
def classify_skewness(s):
    if s > 1 or s < -1:
        return 'Highly Skewed'
    elif 0.5 < s <= 1 or -1 <= s < -0.5:
        return 'Moderately Skewed'
    else:
        return 'Lightly Skewed'

# Skewness
skewness = df_num[df_num.columns].skew()

# Apply classification
skewness_classification = skewness.apply(classify_skewness)

# Print skewness and its classification
print("Skewness of Numeric Variables:")
for column in df_num.columns:
    print(f"{column}: Skewness = {skewness[column]:.2f}, Classification = {skewness
```

```
Skewness of Numeric Variables:
carat: Skewness = 1.12, Classification = Highly Skewed
depth: Skewness = -0.08, Classification = Lightly Skewed
table: Skewness = 0.80, Classification = Moderately Skewed
weight: Skewness = 0.38, Classification = Lightly Skewed
size: Skewness = 2.43, Classification = Highly Skewed
price: Skewness = 1.62, Classification = Highly Skewed
```

D) Use the different transformation techniques to convert skewed data found in previous question into normal distribution.

In [48]:
```
# Log Transformation
df_num[['log_carat','log_table', 'log_size', 'log_price']] = np.log(df_num[['carat'

# Square Root Transformation
df_num[['sqrt_carat', 'sqrt_table', 'sqrt_size', 'sqrt_price']] = np.sqrt(df_num[['

# Cube Root Transformation
df_num[['cbrt_carat', 'cbrt_table', 'cbrt_size', 'cbrt_price']] = np.cbrt(df_num[['
```

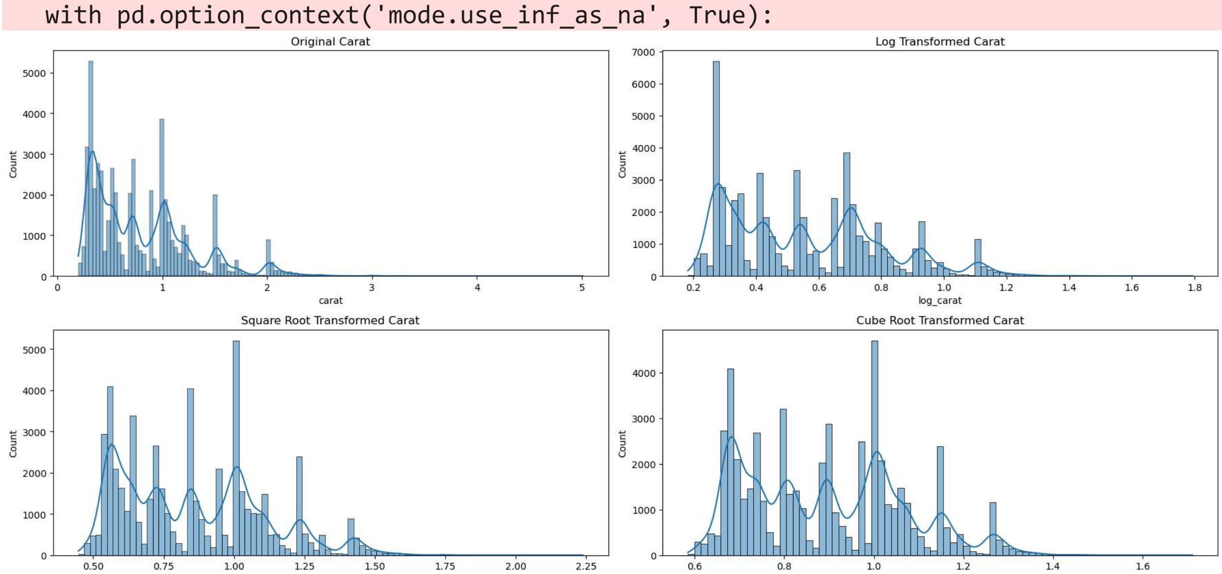
```
C:\Users\user\AppData\Local\Temp\ipykernel_40536\2517777454.py:2: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_num[['log_carat','log_table', 'log_size', 'log_price']] = np.log(df_num[['cara
t', 'table', 'size', 'price']] + 1)  # Adding 1 to avoid log(0)
C:\Users\user\AppData\Local\Temp\ipykernel_40536\2517777454.py:2: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_num[['log_carat','log_table', 'log_size', 'log_price']] = np.log(df_num[['cara
t', 'table', 'size', 'price']] + 1)  # Adding 1 to avoid log(0)
```

In [49]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Calculate skewness after transformations
transformed_skewness = df_num[['log_carat', 'sqrt_carat', 'cbrt_carat']].skew()
print("Transformed Skewness:")
print(transformed_skewness)

# Visualize the transformed distributions
plt.figure(figsize=(18, 12))

# Original Distribution
plt.subplot(3, 2, 1)
sns.histplot(df_num['carat'], kde=True)
plt.title('Original Carat')
```

```python
# Log Transformation
plt.subplot(3, 2, 2)
sns.histplot(df_num['log_carat'], kde=True)
plt.title('Log Transformed Carat')

# Square Root Transformation
plt.subplot(3, 2, 3)
sns.histplot(df_num['sqrt_carat'], kde=True)
plt.title('Square Root Transformed Carat')

# Cube Root Transformation
plt.subplot(3, 2, 4)
sns.histplot(df_num['cbrt_carat'], kde=True)
plt.title('Cube Root Transformed Carat')

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

```
Transformed Skewness:
log_carat     0.580654
sqrt_carat    0.548471
cbrt_carat    0.386589
dtype: float64
```

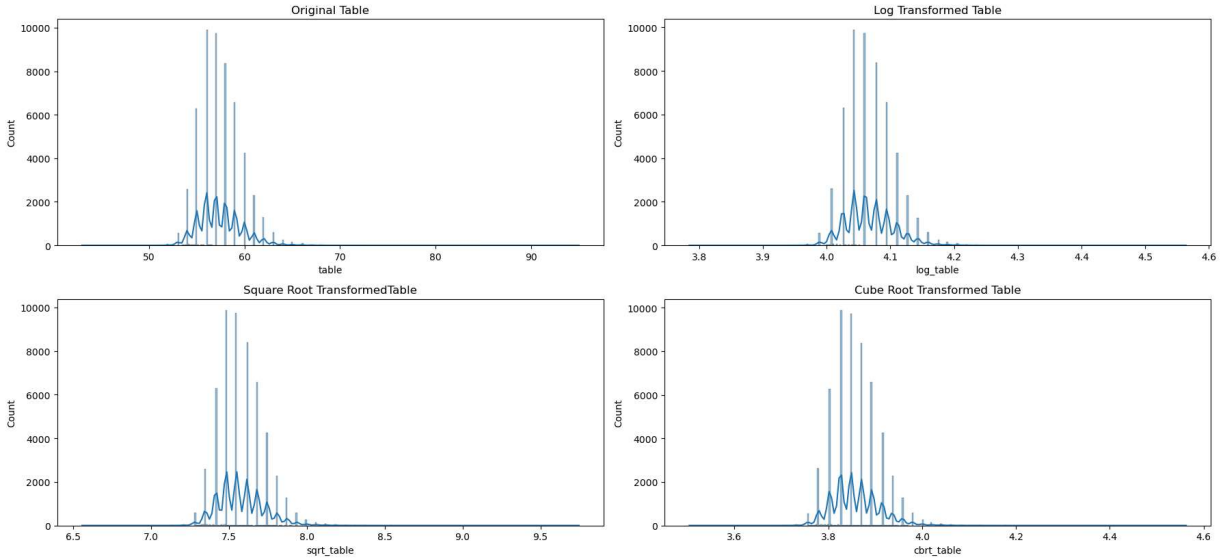
```
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [50]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Calculate skewness after transformations
transformed_skewness = df_num[['log_table', 'sqrt_table', 'cbrt_table']].skew()
print("Transformed Skewness:")
print(transformed_skewness)

# Visualize the transformed distributions
plt.figure(figsize=(18, 12))

# Original Distribution
plt.subplot(3, 2, 1)
sns.histplot(df_num['table'], kde=True)
plt.title('Original Table')

# Log Transformation
plt.subplot(3, 2, 2)
sns.histplot(df_num['log_table'], kde=True)
plt.title('Log Transformed Table')

# Square Root Transformation
plt.subplot(3, 2, 3)
sns.histplot(df_num['sqrt_table'], kde=True)
plt.title('Square Root TransformedTable')

# Cube Root Transformation
plt.subplot(3, 2, 4)
sns.histplot(df_num['cbrt_table'], kde=True)
plt.title('Cube Root Transformed Table')

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

```
Transformed Skewness:
log_table     0.602231
sqrt_table    0.692227
cbrt_table    0.660170
dtype: float64
```

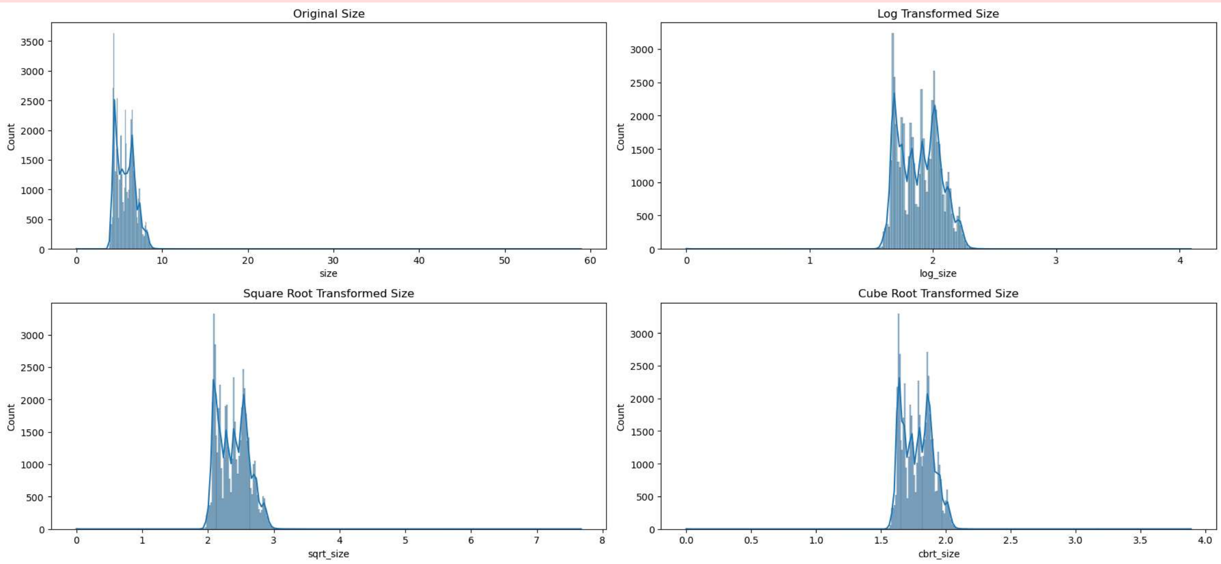
```
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [51]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Calculate skewness after transformations
transformed_skewness = df_num[['log_size', 'sqrt_size', 'cbrt_size']].skew()
print("Transformed Skewness:")
print(transformed_skewness)

# Visualize the transformed distributions
plt.figure(figsize=(18, 12))

# Original Distribution
plt.subplot(3, 2, 1)
sns.histplot(df_num['size'], kde=True)
plt.title('Original Size')

# Log Transformation
plt.subplot(3, 2, 2)
sns.histplot(df_num['log_size'], kde=True)
plt.title('Log Transformed Size')

# Square Root Transformation
plt.subplot(3, 2, 3)
sns.histplot(df_num['sqrt_size'], kde=True)
plt.title('Square Root Transformed Size')

# Cube Root Transformation
plt.subplot(3, 2, 4)
sns.histplot(df_num['cbrt_size'], kde=True)
plt.title('Cube Root Transformed Size')

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

```
Transformed Skewness:
log_size     0.006600
sqrt_size    0.363648
cbrt_size   -0.132075
dtype: float64
```

```
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

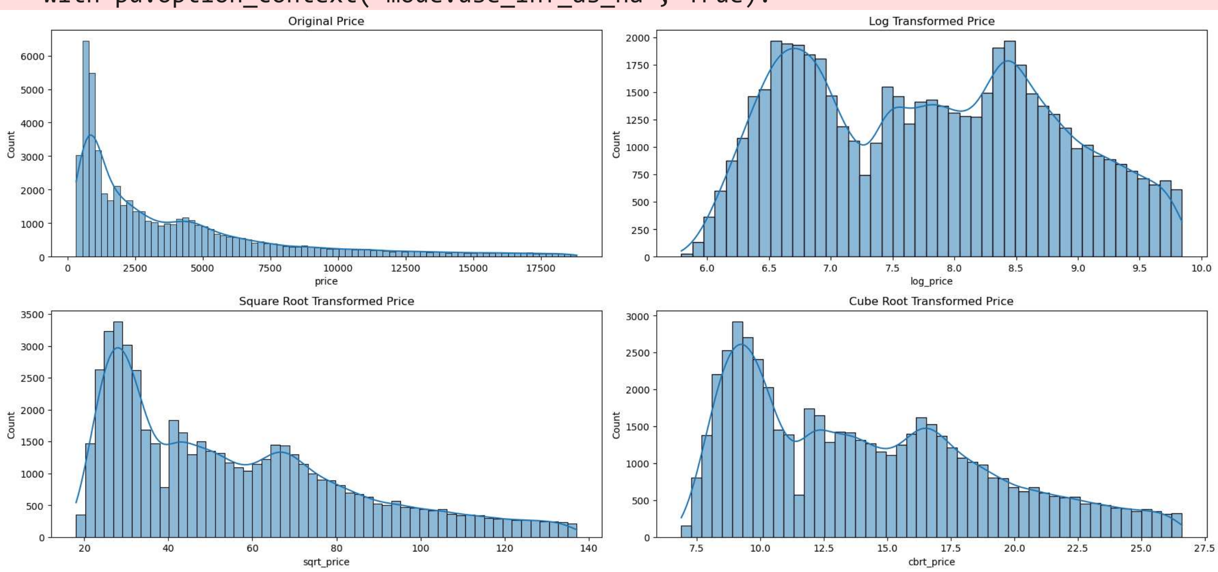


In [52]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
# Calculate skewness after transformations
transformed_skewness = df_num[['log_price', 'sqrt_price', 'cbrt_price']].skew()
print("Transformed Skewness:")
print(transformed_skewness)

# Visualize the transformed distributions
plt.figure(figsize=(18, 12))

# Original Distribution
plt.subplot(3, 2, 1)
sns.histplot(df_num['price'], kde=True)
plt.title('Original Price')

# Log Transformation
plt.subplot(3, 2, 2)
sns.histplot(df_num['log_price'], kde=True)
plt.title('Log Transformed Price')

# Square Root Transformation
plt.subplot(3, 2, 3)
```

```
sns.histplot(df_num['sqrt_price'], kde=True)
plt.title('Square Root Transformed Price')

# Cube Root Transformation
plt.subplot(3, 2, 4)
sns.histplot(df_num['cbrt_price'], kde=True)
plt.title('Cube Root Transformed Price')

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

```
Transformed Skewness:
log_price     0.115926
sqrt_price    0.844396
cbrt_price    0.591189
dtype: float64
```

```
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



E) Create a user defined function in python to check the outliers using IQR method. Then pass all numeric variables in that function to check outliers.

In [53]:
```
df_num = df_num[['carat', 'depth','table','weight', 'size', 'price']]
df_num
```

Out[53]:

	carat	depth	table	weight	size	price
0	0.23	61.5	55.0	3.95	3.98	326
1	0.21	59.8	61.0	3.89	3.84	326
2	0.23	56.9	65.0	4.05	4.07	327
3	0.29	62.4	58.0	4.20	4.23	334
4	0.31	63.3	58.0	4.34	4.35	335
...	...	...	...	...	...	...
53935	0.72	60.8	57.0	5.75	5.76	2757
53936	0.72	63.1	55.0	5.69	5.75	2757
53937	0.70	62.8	60.0	5.66	5.68	2757
53938	0.86	61.0	58.0	6.15	6.12	2757
53939	0.75	62.2	55.0	5.83	5.87	2757

53940 rows × 6 columns

In [54]:
```
# Define a function to check for outliers using the IQR method
def detect_outliers_iqr(df):
    outliers = {}
    for column in df.columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Detect outliers
        column_outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound
        if not column_outliers.empty:
            outliers[column] = column_outliers

    return outliers

# Apply the function to check for outliers
outliers = detect_outliers_iqr(df_num)

# Print outliers for each numeric column
for column, data in outliers.items():
    print(f"Outliers in column '{column}':")
    print(data[column])
```

```
Outliers in column 'carat':
12246    2.06
13002    2.14
13118    2.15
13757    2.22
13991    2.01
         ...
27741    2.15
27742    2.04
27744    2.29
27746    2.07
27749    2.29
Name: carat, Length: 1889, dtype: float64
Outliers in column 'depth':
2        56.9
8        65.1
24       58.1
35       58.2
42       65.2
         ...
53882    65.4
53886    58.0
53890    57.9
53895    57.8
53927    58.1
Name: depth, Length: 2545, dtype: float64
Outliers in column 'table':
2        65.0
91       69.0
145      64.0
219      64.0
227      67.0
         ...
53695    65.0
53697    65.0
53756    64.0
53757    64.0
53785    65.0
Name: table, Length: 605, dtype: float64
Outliers in column 'weight':
11182     0.00
11963     0.00
15951     0.00
22741     9.54
22831     9.38
23644     9.53
24131     9.44
24297     9.49
24328     9.65
24520     0.00
24816     9.42
25460     9.44
25850     9.32
25998    10.14
25999    10.02
26243     0.00
```

```
26431     9.42
26444    10.01
26534     9.86
26932     9.30
27130    10.00
27415    10.74
27429     0.00
27514     9.36
27630    10.23
27638     9.51
27649     9.44
27679     9.66
27684     9.35
27685     9.41
49556     0.00
49557     0.00
Name: weight, dtype: float64
Outliers in column 'size':
11963     0.00
15951     0.00
22741     9.38
22831     9.31
23644     9.48
24067    58.90
24131     9.40
24297     9.42
24328     9.59
24520     0.00
25460     9.37
25998    10.10
25999     9.94
26243     0.00
26431     9.34
26444     9.94
26534     9.81
27130     9.85
27415    10.54
27429     0.00
27514     9.31
27630    10.16
27638     9.46
27649     9.38
27679     9.63
27685     9.32
49189    31.80
49556     0.00
49557     0.00
Name: size, dtype: float64
Outliers in column 'price':
23820    11886
23821    11886
23822    11888
23823    11888
23824    11888
         ...
27745    18803
```

```
27746    18804
27747    18806
27748    18818
27749    18823
Name: price, Length: 3540, dtype: int64
```

F) Convert categorical variables into numerical variables using LabelEncoder technique.

In [55]:
```
from sklearn.preprocessing import LabelEncoder
```

In [56]:
```
# Initialize LabelEncoder
label_encoders = {}
encoded_labels = {}

# Encode categorical variables and store the mappings
for column in df_cat.columns:
    le = LabelEncoder()
    le.fit(df_cat[column])
    df[column] = le.transform(df_cat[column])
    label_encoders[column] = le
    # Create a DataFrame to display original and encoded labels
    encoded_labels[column] = pd.DataFrame({
        'Original': le.classes_,
        'Encoded': le.transform(le.classes_)
    })

# Print the encoded labels mappings
for column, mapping in encoded_labels.items():
    print(f"\nEncoded Labels for column '{column}':")
    print(mapping)
```

```
Encoded Labels for column 'cut':
    Original  Encoded
0       Fair        0
1       Good        1
2      Ideal        2
3    Premium        3
4  Very Good        4

Encoded Labels for column 'color':
  Original  Encoded
0        D        0
1        E        1
2        F        2
3        G        3
4        H        4
5        I        5
6        J        6

Encoded Labels for column 'clarity':
  Original  Encoded
0       I1        0
1       IF        1
2      SI1        2
3      SI2        3
4      VS1        4
5      VS2        5
6     VVS1        6
7     VVS2        7
```

G) Use both the feature scaling techniques (standardscaler/min max scaler) on all the variables.

In [57]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
# Encode categorical data
label_encoder = LabelEncoder()
df_encoded = df_cat.copy()
for column in df_encoded.columns:
    df_encoded[column] = label_encoder.fit_transform(df_encoded[column])

# Scale data
standard_scaler = StandardScaler()
df_standard_scaled = pd.DataFrame(standard_scaler.fit_transform(df),
                                  columns=num_col+cat_col)

min_max_scaler = MinMaxScaler()
df_min_max_scaled = pd.DataFrame(min_max_scaler.fit_transform(df),
                                 columns=num_col+cat_col)

# Combine processed data
df_processed_standard = pd.concat([df_encoded, df_standard_scaled], axis=1)
df_processed_min_max = pd.concat([df_encoded, df_min_max_scaled], axis=1)

# Print results

print("\nStandard Scaled Numerical DataFrame:")
```

```
print(df_standard_scaled)

print("\nMinMax Scaled Numerical DataFrame:")
print(df_min_max_scaled)

print("\nProcessed DataFrame with Standard Scaling:")
print(df_processed_standard)

print("\nProcessed DataFrame with MinMax Scaling:")
print(df_processed_min_max)
```

```
Standard Scaled Numerical DataFrame:
          carat     depth     table    weight      size     price       cut  \
0     -1.198168 -0.538099 -0.937163 -0.484264 -0.174092 -1.099672 -1.587837
1     -1.240361  0.434949 -0.937163 -1.064117 -1.360738  1.585529 -1.641325
2     -1.198168 -1.511147 -0.937163  0.095589 -3.385019  3.375663 -1.498691
3     -1.071587  0.434949  1.414272  0.675442  0.454133  0.242928 -1.364971
4     -1.029394 -1.511147  2.002131 -0.484264  1.082358  0.242928 -1.240167
...         ...       ...       ...       ...       ...       ...       ...
53935 -0.164427 -0.538099 -1.525021 -1.064117 -0.662711 -0.204605  0.016798
53936 -0.164427 -1.511147 -1.525021 -1.064117  0.942753 -1.099672 -0.036690
53937 -0.206621  1.407998 -1.525021 -1.064117  0.733344  1.137995 -0.063434
53938  0.130927  0.434949  0.826413 -0.484264 -0.523105  0.242928  0.373383
53939 -0.101137 -0.538099 -1.525021 -0.484264  0.314528 -1.099672  0.088115

          color   clarity
0     -1.536196 -0.904095
1     -1.658774 -0.904095
2     -1.457395 -0.903844
3     -1.317305 -0.902090
4     -1.212238 -0.901839
...         ...       ...
53935  0.022304 -0.294731
53936  0.013548 -0.294731
53937 -0.047741 -0.294731
53938  0.337506 -0.294731
53939  0.118616 -0.294731

[53940 rows x 9 columns]

MinMax Scaled Numerical DataFrame:
          carat  depth     table    weight      size     price       cut  \
0      0.006237   0.50  0.166667  0.428571  0.513889  0.230769  0.367784
1      0.002079   0.75  0.166667  0.285714  0.466667  0.346154  0.362197
2      0.006237   0.25  0.166667  0.571429  0.386111  0.423077  0.377095
3      0.018711   0.75  0.833333  0.714286  0.538889  0.288462  0.391061
4      0.022869   0.25  1.000000  0.428571  0.563889  0.288462  0.404097
...         ...    ...       ...       ...       ...       ...       ...
53935  0.108108   0.50  0.000000  0.285714  0.494444  0.269231  0.535382
53936  0.108108   0.25  0.000000  0.285714  0.558333  0.230769  0.529795
53937  0.103950   1.00  0.000000  0.285714  0.550000  0.326923  0.527002
53938  0.137214   0.75  0.666667  0.428571  0.500000  0.288462  0.572626
53939  0.114345   0.50  0.000000  0.428571  0.533333  0.230769  0.542831

          color   clarity
0      0.067572  0.000000
1      0.065195  0.000000
2      0.069100  0.000054
3      0.071817  0.000433
4      0.073854  0.000487
...         ...       ...
53935  0.097793  0.131427
53936  0.097623  0.131427
53937  0.096435  0.131427
53938  0.103905  0.131427
53939  0.099660  0.131427
```

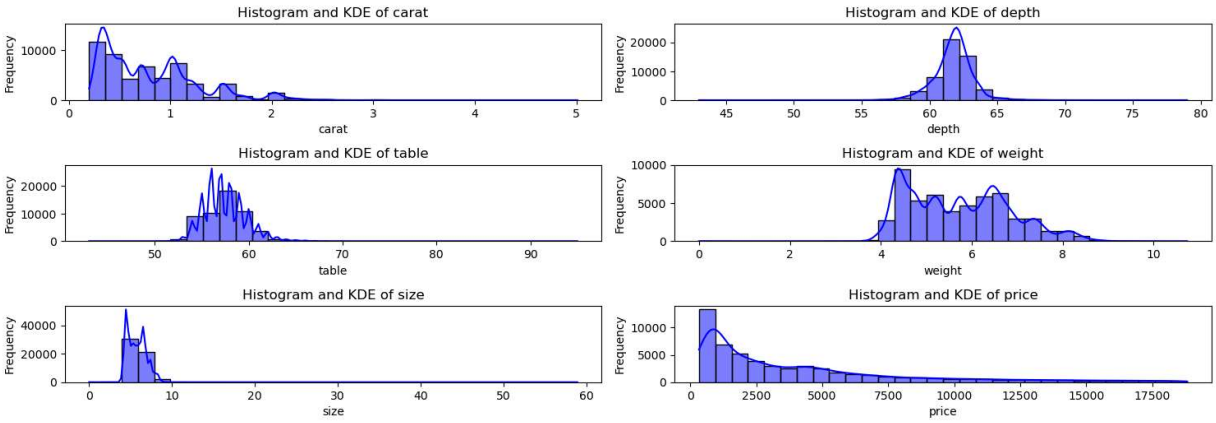
```
[53940 rows x 9 columns]

Processed DataFrame with Standard Scaling:
       cut  color  clarity     carat     depth     table    weight      size  \
0        2      1        3 -1.198168 -0.538099 -0.937163 -0.484264 -0.174092
1        3      1        2 -1.240361  0.434949 -0.937163 -1.064117 -1.360738
2        1      1        4 -1.198168 -1.511147 -0.937163  0.095589 -3.385019
3        3      5        5 -1.071587  0.434949  1.414272  0.675442  0.454133
4        1      6        3 -1.029394 -1.511147  2.002131 -0.484264  1.082358
...    ...    ...      ...       ...       ...       ...       ...       ...
53935    2      0        2 -0.164427 -0.538099 -1.525021 -1.064117 -0.662711
53936    1      0        2 -0.164427 -1.511147 -1.525021 -1.064117  0.942753
53937    4      0        2 -0.206621  1.407998 -1.525021 -1.064117  0.733344
53938    3      4        3  0.130927  0.434949  0.826413 -0.484264 -0.523105
53939    2      0        3 -0.101137 -0.538099 -1.525021 -0.484264  0.314528

          price       cut     color   clarity
0     -1.099672 -1.587837 -1.536196 -0.904095
1      1.585529 -1.641325 -1.658774 -0.904095
2      3.375663 -1.498691 -1.457395 -0.903844
3      0.242928 -1.364971 -1.317305 -0.902090
4      0.242928 -1.240167 -1.212238 -0.901839
...         ...       ...       ...       ...
53935 -0.204605  0.016798  0.022304 -0.294731
53936 -1.099672 -0.036690  0.013548 -0.294731
53937  1.137995 -0.063434 -0.047741 -0.294731
53938  0.242928  0.373383  0.337506 -0.294731
53939 -1.099672  0.088115  0.118616 -0.294731

[53940 rows x 12 columns]

Processed DataFrame with MinMax Scaling:
       cut  color  clarity     carat  depth     table    weight      size  \
0        2      1        3  0.006237   0.50  0.166667  0.428571  0.513889
1        3      1        2  0.002079   0.75  0.166667  0.285714  0.466667
2        1      1        4  0.006237   0.25  0.166667  0.571429  0.386111
3        3      5        5  0.018711   0.75  0.833333  0.714286  0.538889
4        1      6        3  0.022869   0.25  1.000000  0.428571  0.563889
...    ...    ...      ...       ...    ...       ...       ...       ...
53935    2      0        2  0.108108   0.50  0.000000  0.285714  0.494444
53936    1      0        2  0.108108   0.25  0.000000  0.285714  0.558333
53937    4      0        2  0.103950   1.00  0.000000  0.285714  0.550000
53938    3      4        3  0.137214   0.75  0.666667  0.428571  0.500000
53939    2      0        3  0.114345   0.50  0.000000  0.428571  0.533333

          price       cut     color   clarity
0      0.230769  0.367784  0.067572  0.000000
1      0.346154  0.362197  0.065195  0.000000
2      0.423077  0.377095  0.069100  0.000054
3      0.288462  0.391061  0.071817  0.000433
4      0.288462  0.404097  0.073854  0.000487
...         ...       ...       ...       ...
53935  0.269231  0.535382  0.097793  0.131427
53936  0.230769  0.529795  0.097623  0.131427
53937  0.326923  0.527002  0.096435  0.131427
53938  0.288462  0.572626  0.103905  0.131427
```

```
53939  0.230769  0.542831  0.099660  0.131427

[53940 rows x 12 columns]
```

H) Create the Histogram for all numeric variables and draw the KDE plot on that.

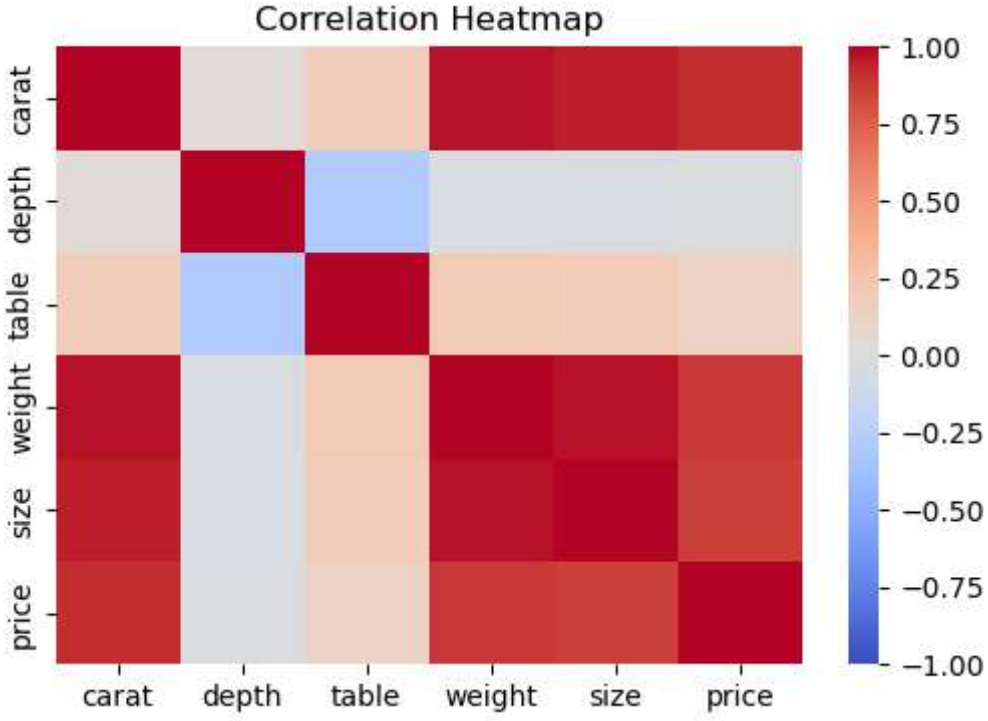
In [58]:
```
# Set up the matplotlib figure
plt.figure(figsize=(15, 10))

# Loop through each numeric column
for i, col in enumerate(df_num.columns):
    plt.subplot(len(df_num.columns), 2, i + 1)
    sns.histplot(df_num[col], kde=True, bins=30, color='blue')
    plt.title(f'Histogram and KDE of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()
```

```
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert i
nf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



I) Check the correlation between all the numeric variables using HeatMap and try to draw some conclusion about the data.

In [59]:
```
# Compute the correlation matrix
corr_matrix = df_num.corr()

# Plot the heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(corr_matrix, cmap='coolwarm', vmin=-1, vmax=1, fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



In [63]:
```
# From the Heatmap, we can say that all ,except the 'depth' column are positively c
# so we can say that if the depth of the diamond increases, its sparkling or shinen
```

2) Explain Gradient descent in detail. How changing the values of learning rate can impact the convergence in Gradient Descent.

Gradient descent is an optimization algorithm used to minimize a function by descending a slope to reach lowest point on that surface. It's a technique in machine learning and optimization for finding the minimum of a function, often used in training algorithms like linear regression, logistic regression, and neural networks.

How Gradient Descent Works Initialization: Find the slope of the objective function with respect to 'x'. Start with an initial guess for the parameters of the function you want to optimize. This is often done randomly or based on some heuristic.

Compute the Gradient: Calculate the gradient (or partial derivatives) of the function with respect to each parameter. The gradient is a vector that points in the direction of the steepest ascent.

Update Parameters: Adjust the parameters in the opposite direction of the gradient to minimize the function. This step involves subtracting a fraction of the gradient from the current parameters. Mathematically, the update rule for each parameter

Step Size = Gradient * Learning Rate

Iterate: Repeat the gradient computation and parameter update steps until convergence, i.e., until changes in the parameters are smaller than a predetermined threshold or a maximum number of iterations is reached.

Learning Rate and Its Impact The learning rate α α controls the size of the steps taken towards the minimum. It's a crucial hyperparameter in gradient descent, and its choice can significantly affect the algorithm's performance:

Small Learning Rate:

Pros: The algorithm makes small, cautious updates, which can lead to a more precise convergence. Cons: It may converge very slowly, taking a long time to reach the minimum. In some cases, it might get stuck in local minima or saddle points because the steps are too small to escape.

Large Learning Rate:

Pros: The algorithm can converge faster because it takes larger steps. Cons: If too large, the steps might overshoot the minimum, leading to divergence or oscillations around the minimum. In extreme cases, it can cause the algorithm to diverge, as the updates become too erratic to find the minimum effectively. Adaptive Learning Rates:

Summary Gradient descent is a powerful optimization technique, but its efficiency depends significantly on the choice of the learning rate. A well-chosen learning rate balances between convergence speed and stability. While smaller rates can ensure careful progress, larger rates can speed up convergence but risk instability. Adaptive methods offer a way to fine-tune the learning rate dynamically, improving the chances of efficient and stable convergence.

In [ ]: