In [1]:
```python
# Generic inputs for most ML tasks
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor

pd.options.display.float_format = '{:,.2f}'.format

# setup interactive notebook mode
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from IPython.display import display, HTML
```

Read and pre-process data

In [2]:
```python
# fetch data

diamonds_data = pd.read_csv('diamonds.csv')

diamonds_data.head()
```

Out[2]:

	rownames	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.50	55.00	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.80	61.00	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.90	65.00	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.40	58.00	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.30	58.00	335	4.34	4.35	2.75

In [3]:
```python
#column data types
diamonds_data.dtypes
```

Out[3]:
```
rownames      int64
carat       float64
cut          object
color        object
clarity      object
depth       float64
table       float64
price         int64
x           float64
y           float64
z           float64
dtype: object
```

In [4]:
```
#dropping rownames column

diamonds_data = diamonds_data.drop(columns = ['rownames'], axis=1)

diamonds_data.head()
```

Out[4]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.50	55.00	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.80	61.00	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.90	65.00	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.40	58.00	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.30	58.00	335	4.34	4.35	2.75

In [5]:
```
#NaN values
diamonds_data.isna().sum()
```

Out[5]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

There are no NaN values in the dataframe

In [80]:
```
#No. of rows
print("Number of rows in the dataframe are",diamonds_data.shape[0])
```

```
Number of rows in the dataframe are 53940
```

In [7]:
```
#values taken by the three categorical variables

cut_values = diamonds_data['cut'].unique()
print("Set of values for 'cut':", cut_values)

# To get unique values for the 'color' categorical variable
color_values = diamonds_data['color'].unique()
print("Set of values for 'color':", color_values)

# To get unique values for the 'clarity' categorical variable
clarity_values = diamonds_data['clarity'].unique()
print("Set of values for 'clarity':", clarity_values)
```

```
Set of values for 'cut': ['Ideal' 'Premium' 'Good' 'Very Good' 'Fair']
Set of values for 'color': ['E' 'I' 'J' 'H' 'F' 'G' 'D']
Set of values for 'clarity': ['SI2' 'SI1' 'VS1' 'VS2' 'VVS2' 'VVS1' 'I
1' 'IF']
```

Simple Linear Regression

In [8]:

```
import seaborn as sns
df_without_categorical_values = diamonds_data.drop(columns = ['cut', 'co
corr_matrix = df_without_categorical_values.corr()

# Create a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap between spx, dax, ftse, and nikkei')
plt.show()
```

Out[8]:

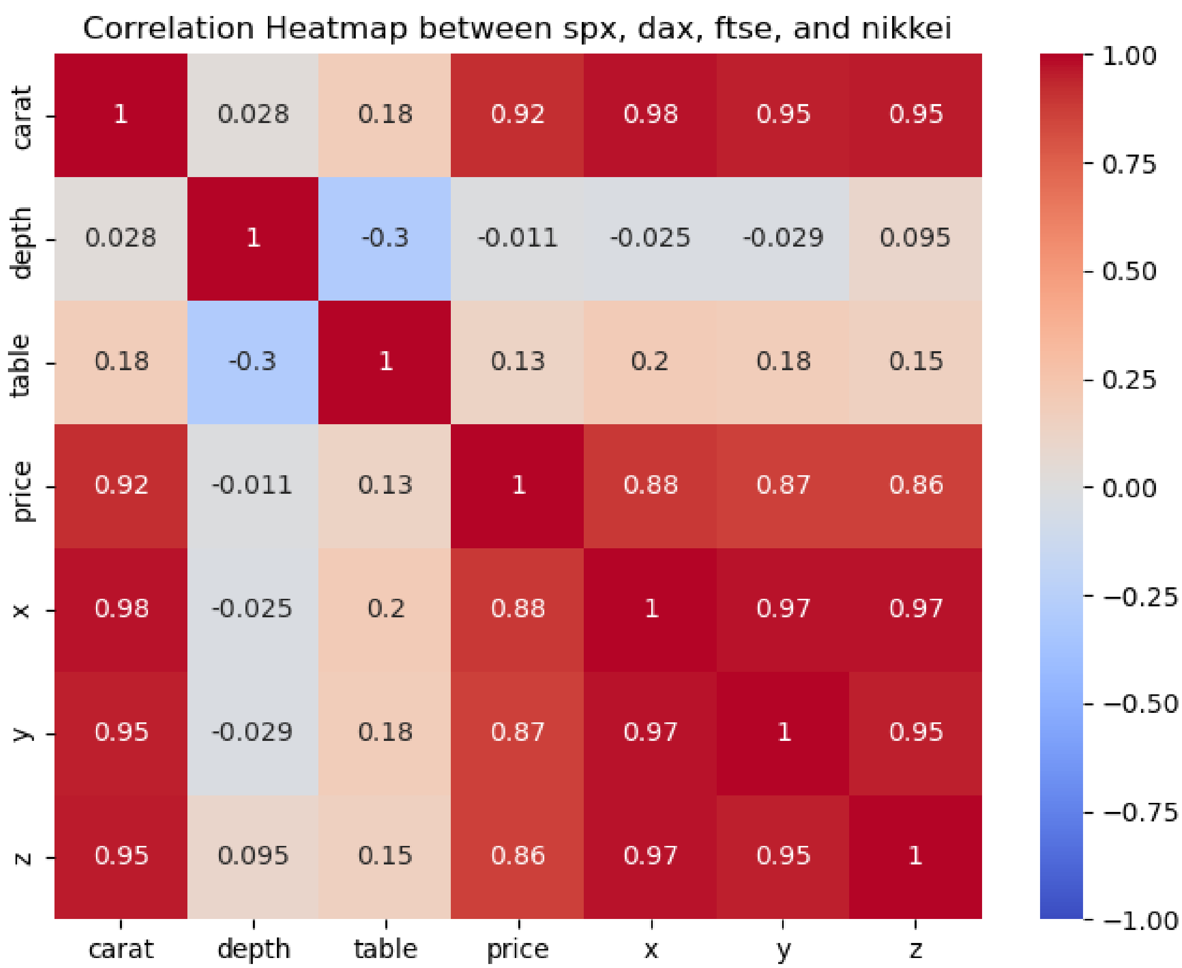
```
<Figure size 800x600 with 0 Axes>
```

Out[8]:

```
<Axes: >
```

Out[8]:

```
Text(0.5, 1.0, 'Correlation Heatmap between spx, dax, ftse, and nikke
i')
```



carat has the highest correlation with price

In [9]:

```
X = diamonds_data['carat']
```

In [10]:

```
Y = diamonds_data['price']
```

In [11]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25
```

In [12]:
```python
train_length = X_train.shape[0]
test_length = X_test.shape[0]

print('Length of train and test data are:', train_length , test_length )
```

```
Length of train and test data are: 40455 13485
```

In [13]:
```python
first_row_index_train = X_train.index[0]
first_row_index_test = X_test.index[0]

print("First row index of X_train:", first_row_index_train)
print("First row index of X_test:", first_row_index_test)
```

```
First row index of X_train: 32301
First row index of X_test: 44751
```

In [14]:
```python
model = LinearRegression(fit_intercept = True)
model.fit(X_train.array.reshape(-1, 1), y_train.array.reshape(-1, 1))
```

Out[14]:
```
▾ LinearRegression
LinearRegression()
```

In [15]:
```python
# The following gives the R-square score
model.score(X_train.array.reshape(-1, 1), y_train.array.reshape(-1, 1)) #

# This is the coefficient Beta_1 (or slope of the Simple Linear Regressi
model.coef_

# This is the coefficient Beta_0
model.intercept_
```

Out[15]:
```
0.8486051211546091
```

Out[15]:
```
array([[7740.34546342]])
```

Out[15]:
```
array([-2244.80400041])
```

An R-square score of approximately 0.85 suggests that the model explains about 85% of the variance in the target variable, which is generally considered a strong correlation and indicates a good fit.

Coefficients

1. Beta_1 (Coefficient for the Independent Variable): 7740.34546342 This coefficient represents the slope of the regression line in the case of simple linear regression (or the effect size of the variable in multiple regression). It indicates how much the dependent variable is expected to increase (or decrease, if the coefficient was negative) when the independent variable increases by one unit. In this context, a coefficient of approximately

7740.35 means that for each one-unit increase in the independent variable, the dependent variable is expected to increase by about 7740.35 units. This value suggests a strong positive relationship between the independent variable and the dependent variable.

2. Beta_0 (Intercept): -2244.80400041 The intercept, or Beta_0, represents the expected value of the dependent variable when all independent variables are equal to zero. An intercept of

In [250]:
```
# Calculate R-squared on test data
model.score(X_test.array.reshape(-1, 1), y_test.array.reshape(-1, 1))
```

Out[250]:
```
0.8514897700970667
```

In [84]:
```
test_output = pd.DataFrame(model.predict(X_test.array.reshape(-1, 1)), in
```

In [17]:
```
test_output.head()
```

Out[17]:

	pred_price
44751	1,702.77
22963	13,313.29
9078	6,114.77
26148	14,242.13
29451	309.51

In [18]:
```
test_output = test_output.merge(y_test, left_index = True, right_index =
test_output.head()
mean_absolute_error = abs(test_output['pred_price'] - test_output['price
print('Mean absolute error is ')
print(mean_absolute_error)
```

Out[18]:

	pred_price	price
44751	1,702.77	1619
22963	13,313.29	11011
9078	6,114.77	4521
26148	14,242.13	15454
29451	309.51	702

```
Mean absolute error is
998.4973200867355
```

In [19]:
```
average_price_test = y_test.mean()
```

In [20]:
```
fraction_mae = mean_absolute_error / average_price_test
print("Fraction of MAE to Average Price:", fraction_mae)
```

```
Fraction of MAE to Average Price: 0.2541206399951147
```

A fraction of approximately 0.254 means that the mean absolute error is about 25.4% of the average price in the test set.

An MAE of 998.5 and its fraction of 25.4% of the average price indicate that, despite the good fit, the average prediction error may still represent a significant portion of the price.

Multiple Linear Regression:

In [33]:
```
X2 = diamonds_data.drop(columns = ['cut', 'color', 'clarity', 'price'], a
```

In [34]:
```
X2.head()
```

Out[34]:

	carat	depth	table	x	y	z
0	0.23	61.50	55.00	3.95	3.98	2.43
1	0.21	59.80	61.00	3.89	3.84	2.31
2	0.23	56.90	65.00	4.05	4.07	2.31
3	0.29	62.40	58.00	4.20	4.23	2.63
4	0.31	63.30	58.00	4.34	4.35	2.75

In [35]:
```
Y2 = diamonds_data["price"]
```

In [36]:
```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, Y2, test_siz
```

In [42]:
```
X2_train
X2_test
y2_test
y2_train
```

Out[42]:

	carat	depth	table	x	y	z
32301	0.37	60.70	60.00	4.65	4.68	2.83
39009	0.40	61.70	57.00	4.77	4.73	2.93
22757	1.02	61.40	58.00	6.46	6.43	3.96
15129	1.07	62.30	55.00	6.59	6.54	4.09
17861	1.19	61.70	56.00	6.78	6.81	4.19
...	...	...	...	...	...	...
48417	0.70	62.40	58.00	5.62	5.67	3.52
22637	2.00	60.30	56.00	8.27	8.16	4.94
42891	0.51	62.80	57.00	5.12	5.10	3.21
38368	0.53	63.80	57.00	5.10	5.12	3.26
14000	1.19	62.70	61.00	6.73	6.66	4.20

40455 rows × 6 columns

Out[42]:

	carat	depth	table	x	y	z
44751	0.51	61.40	58.00	5.13	5.09	3.14
22963	2.01	62.90	54.00	8.06	7.93	5.05
9078	1.08	62.10	59.00	6.57	6.53	4.07
26148	2.13	61.50	57.00	8.27	8.34	5.11
29451	0.33	61.90	56.00	4.46	4.49	2.77
...	...	...	...	...	...	...
8884	1.04	62.40	56.00	6.40	6.48	4.02
34947	0.30	61.90	60.00	4.27	4.29	2.65
21821	1.50	60.20	61.00	7.27	7.32	4.39
9942	1.01	61.10	56.00	6.44	6.48	3.95
8296	1.13	61.70	57.00	6.70	6.59	4.13

13485 rows × 6 columns

Out[42]:
```
44751     1619
22963    11011
9078      4521
26148    15454
29451      702
         ...
8884      4486
34947      378
21821     9892
9942      4693
8296      4385
Name: price, Length: 13485, dtype: int64
```

Out[42]:
```
32301      454
39009     1056
22757    10773
15129     6082
17861     7207
         ...
48417     1971
22637    10685
42891     1359
38368     1023
14000     5698
Name: price, Length: 40455, dtype: int64
```

In [43]:
```
train_length2 = X2_train.shape[0]
test_length2 = X2_test.shape[0]

print('Length of train and test data are:', train_length2 , test_length2
```

```
Length of train and test data are: 40455 13485
```

The length of test and train data are same as the lengths of SLR

In [44]:
```
first_row_index_train2 = X2_train.index[0]
first_row_index_test2 = X2_test.index[0]

print("First row index of X_train:", first_row_index_train2)
print("First row index of X_test:", first_row_index_test2)
```

```
First row index of X_train: 32301
First row index of X_test: 44751
```

First row index of both test and train data are same as that of SLR

In [45]:
```
model2 = LinearRegression(fit_intercept = True)
model2.fit(X2_train, y2_train)
```

Out[45]:
```
▾ LinearRegression
LinearRegression()
```

In [46]:
```python
# The following gives the R-square score on train data
model2.score(X2_train, y2_train)

# This is the coefficient Beta_1, ..., Beta_7
model2.coef_

# This is the coefficient Beta_0
model2.intercept_
```

Out[46]:
```
0.8580892707297321
```

Out[46]:
```
array([10572.42070164,  -212.30889735,  -100.98829263, -1339.48535974,
          28.72790744,   207.71501298])
```

Out[46]:
```
21184.40695343088
```

The R-square score for model2 is approximately 0.858, which means that about 85.8% of the variance in the dependent variable is explained by the model. This is a strong score that suggests a good fit of the model to the data, especially in the context of multiple regression where more than one independent variable is used to predict the outcome.

In [251]:
```python
# The following gives the R-square score on train data
model2.score(X2_test, y2_test)
```

Out[251]:
```
0.8623985511113466
```

In [57]:
```python
test_output2 = pd.DataFrame(model2.predict(X2_test), index = X2_test.inde
```

In [58]:
```python
test_output2.head()
```

Out[58]:

	pred_price
44751	1,610.14
22963	14,107.90
9078	5,692.50
26148	15,113.80
29451	606.29

In [59]:
```
test_output2 = test_output2.merge(y2_test, left_index = True, right_inde
test_output2.head()
mean_absolute_error2 = abs(test_output2['pred_price'] - test_output2['pr
print('Mean absolute error is ')
print(mean_absolute_error2)
```

Out[59]:

	pred_price	price
44751	1,610.14	1619
22963	14,107.90	11011
9078	5,692.50	4521
26148	15,113.80	15454
29451	606.29	702

```
Mean absolute error is
882.599634844387
```

In [60]:
```
average_price_test2 = y2_test.mean()
```

In [61]:
```
fraction_mae2 = mean_absolute_error2 / average_price_test2
print("Fraction of MAE to Average Price:", fraction_mae2)
```

```
Fraction of MAE to Average Price: 0.224624322523597
```

This fraction indicates that the mean absolute error is about 22.5% of the average price in the test set.

Comparison with SLR For the SLR model, the reported MAE was approximately 998.5, and the fraction of MAE to the average price was about 25.4%. Compared to these values, the MLR model has shown an improvement in both the absolute size of the error (a lower MAE) and the error relative to the average price (a lower fraction).

Conclusion Yes, the predictions in the test set have improved with the MLR model compared to the SLR model. The reduction in both the MAE and its fraction to the average price indicates that incorporating multiple features into the model has enhanced its predictive accuracy, leading to closer predictions to the actual values and a reduction in the average error magnitude relative to the price scale.

Multiple Linear Regression with Categorical Values

In [197]:
```
X3 = diamonds_data.drop(columns = ['price'], axis=1)
X3.head()
```

Out[197]:

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	Ideal	E	SI2	61.50	55.00	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.80	61.00	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.90	65.00	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.40	58.00	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.30	58.00	4.34	4.35	2.75

In [198]:
```
Y3 = diamonds_data['price']
Y3.head()
```

Out[198]:
```
0    326
1    326
2    327
3    334
4    335
Name: price, dtype: int64
```

In [199]:
```
print(diamonds_data['cut'].unique())
print(diamonds_data['color'].unique())
print(diamonds_data['clarity'].unique())
```

```
['Ideal' 'Premium' 'Good' 'Very Good' 'Fair']
['E' 'I' 'J' 'H' 'F' 'G' 'D']
['SI2' 'SI1' 'VS1' 'VS2' 'VVS2' 'VVS1' 'I1' 'IF']
```

In [200]:
```
diamonds_data['cut'].value_counts()
diamonds_data['color'].value_counts()
diamonds_data['clarity'].value_counts()
```

Out[200]:
```
cut
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: count, dtype: int64
```

Out[200]:
```
color
G    11292
E     9797
F     9542
H     8304
D     6775
I     5422
J     2808
Name: count, dtype: int64
```

Out[200]:
```
clarity
SI1     13065
VS2     12258
SI2      9194
VS1      8171
VVS2     5066
VVS1     3655
IF       1790
I1        741
Name: count, dtype: int64
```

In [201]:
```
from sklearn.preprocessing import OneHotEncoder

def get_ohe(df, col):
    ohe = OneHotEncoder(drop='first', handle_unknown='error', sparse_out
    ohe.fit(df[[col]])
    temp_df = pd.DataFrame(data=ohe.transform(df[[col]]), columns=ohe.ge
    # If you have a newer version, replace with columns=ohe.get_feature_
    df.drop(columns=[col], axis=1, inplace=True)
    df = pd.concat([df.reset_index(drop=True), temp_df], axis=1)
    return df
```

In [202]:
```python
X3 = get_ohe(X3, 'cut')
X3 = get_ohe(X3, 'color')
X3 = get_ohe(X3, 'clarity')

X3.head(20)
```

Out[202]:

	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	...	color_H
0	0.23	61.50	55.00	3.95	3.98	2.43	0	1	0	0	...	0
1	0.21	59.80	61.00	3.89	3.84	2.31	0	0	1	0	...	0
2	0.23	56.90	65.00	4.05	4.07	2.31	1	0	0	0	...	0
3	0.29	62.40	58.00	4.20	4.23	2.63	0	0	1	0	...	0
4	0.31	63.30	58.00	4.34	4.35	2.75	1	0	0	0	...	0
5	0.24	62.80	57.00	3.94	3.96	2.48	0	0	0	1	...	0
6	0.24	62.30	57.00	3.95	3.98	2.47	0	0	0	1	...	0
7	0.26	61.90	55.00	4.07	4.11	2.53	0	0	0	1	...	1
8	0.22	65.10	61.00	3.87	3.78	2.49	0	0	0	0	...	0
9	0.23	59.40	61.00	4.00	4.05	2.39	0	0	0	1	...	1
10	0.30	64.00	55.00	4.25	4.28	2.73	1	0	0	0	...	0
11	0.23	62.80	56.00	3.93	3.90	2.46	0	1	0	0	...	0
12	0.22	60.40	61.00	3.88	3.84	2.33	0	0	1	0	...	0
13	0.31	62.20	54.00	4.35	4.37	2.71	0	1	0	0	...	0
14	0.20	60.20	62.00	3.79	3.75	2.27	0	0	1	0	...	0
15	0.32	60.90	58.00	4.38	4.42	2.68	0	0	1	0	...	0
16	0.30	62.00	54.00	4.31	4.34	2.68	0	1	0	0	...	0
17	0.30	63.40	54.00	4.23	4.29	2.70	1	0	0	0	...	0
18	0.30	63.80	56.00	4.23	4.26	2.71	1	0	0	0	...	0
19	0.30	62.70	59.00	4.21	4.27	2.66	0	0	0	1	...	0

20 rows × 23 columns

Here's how we determine the number of columns in the eventual set after one-hot encoding:

let's consider the unique categories in each of the encoded categorical columns minus one (since we're dropping the first column):

'cut': If it originally has 5 unique values (Fair, Good, Very Good, Premium, Ideal), one-hot encoding with drop='first' will result in 4 columns. 'color': If it has 7 unique values (D through J), one-hot encoding with drop='first' will result in 6 columns. 'clarity': If it has 8 unique values (I1, SI2, SI1, VS2, VS1, VVS2, VVS1, IF), one-hot encoding with drop='first' will result in 7 columns.

Total Columns After Encoding Original non-encoded columns: 6 Encoded columns for 'cut': 4 Encoded columns for 'color': 6 Encoded columns for 'clarity': 7 Total = 6 (original) + 4 (cut) + 6 (color) + 7 (clarity) = 23 columns

In [204]:
```
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, Y3, test_siz
```

In [205]:
```
model3 = LinearRegression(fit_intercept = True)
model3.fit(X3_train, y3_train)
```

Out[205]:
```
▾ LinearRegression
LinearRegression()
```

In [206]:
```
# The following gives the R-square score on train data
model3.score(X3_train, y3_train)

# This is the coefficient Beta_1, ..., Beta_26
model3.coef_

# This is the coefficient Beta_0
model3.intercept_
```

Out[206]:
```
0.9197535054831985
```

Out[206]:
```
array([ 1.11814801e+04, -5.85181599e+01, -2.43103736e+01, -9.47154791e+
02,
        4.71160598e+00, -9.21343078e+01,  5.64156045e+02,  8.35219660e+
02,
        7.65684380e+02,  7.33319794e+02, -2.25861063e+02, -2.76687793e+
02,
       -4.79782911e+02, -9.73649105e+02, -1.46411040e+03, -2.39532202e+
03,
        5.45238205e+03,  3.78052359e+03,  2.82037545e+03,  4.70977853e+
03,
        4.38909093e+03,  5.12101322e+03,  5.07443217e+03])
```

Out[206]:
```
1499.3392254312134
```

The enhanced multiple linear regression model, incorporating extra features like categorical data, significantly improves our result interpretation compared to its predecessor. It achieves a score close to 0.920, signifying a robust fit. The intercept, Beta_0, is estimated to be approximately 1499.339. Each of the model's coefficients indicates the degree to which a particular feature affects the outcome variable. This advancement provides a deeper insight into the various elements affecting predictions, thereby increasing the model's overall usefulness.

In [252]:
```
# The following gives the R-square score on test data
model3.score(X3_test, y3_test)
```

Out[252]:
```
0.9197839099301064
```

In [207]:
```
test_output3 = pd.DataFrame(model3.predict(X3_test), index = X3_test.ind
```

In [208]:
```
test_output3.head()
```

Out[208]:

	pred_price
44751	1,952.74
22963	12,178.85
9078	5,091.65
26148	15,140.76
29451	137.09

In [209]:
```
test_output3 = test_output3.merge(y3_test, left_index = True, right_inde
test_output3.head()
mean_absolute_error3 = abs(test_output3['pred_price'] - test_output3['pr
print('Mean absolute error is ')
print(mean_absolute_error3)
```

Out[209]:

	pred_price	price
44751	1,952.74	1619
22963	12,178.85	11011
9078	5,091.65	4521
26148	15,140.76	15454
29451	137.09	702

```
Mean absolute error is
744.8276400124315
```

An MAE of approximately 744.82 indicates that, on average, the model's predictions are about 744.82 units away from the actual values.

In [210]:
```
average_price_test3 = y3_test.mean()
```

In [211]:
```
fraction_mae3 = mean_absolute_error3 / average_price_test3
print("Fraction of MAE to Average Price:", fraction_mae3)
```

```
Fraction of MAE to Average Price: 0.18956092596179258
```

An average absolute error of approximately 745, coupled with its ratio to the mean price being roughly 0.19, indicates that our enhanced model, which includes additional variables, performs commendably in forecasting results. Given that the model now aligns more closely with the data, evidenced by an R-square value close to 0.92, it logically follows that there has been a reduction in predictive errors. This marks a significant enhancement from the previous error ratio of around 0.23, highlighting the beneficial impact of incorporating more variables. Essentially, it signifies an improvement in the predictive accuracy of our model.

Quantile Regression with Categorical Variables

In [212]:
```
X4 = diamonds_data.drop(columns = ['price'], axis=1)
X4.head()
```

Out[212]:

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	Ideal	E	SI2	61.50	55.00	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.80	61.00	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.90	65.00	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.40	58.00	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.30	58.00	4.34	4.35	2.75

In [213]:
```
Y4 = diamonds_data['price']
Y4.head()
```

Out[213]:
```
0    326
1    326
2    327
3    334
4    335
Name: price, dtype: int64
```

In [214]:
```
X4 = get_ohe(X4, 'cut')
X4 = get_ohe(X4, 'color')
X4 = get_ohe(X4, 'clarity')

X4.head()
```

Out[214]:

	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	...	color_H
0	0.23	61.50	55.00	3.95	3.98	2.43	0	1	0	0	...	0
1	0.21	59.80	61.00	3.89	3.84	2.31	0	0	1	0	...	0
2	0.23	56.90	65.00	4.05	4.07	2.31	1	0	0	0	...	0
3	0.29	62.40	58.00	4.20	4.23	2.63	0	0	1	0	...	0
4	0.31	63.30	58.00	4.34	4.35	2.75	1	0	0	0	...	0

5 rows × 23 columns

In [215]:
```
X4_train, X4_test, y4_train, y4_test = train_test_split(X4, Y4, test_siz
```

In [217]:
```
import statsmodels.formula.api as smf
import statsmodels.api as sm
```

In [253]:
```
# Create the quantile regression model
mod = sm.QuantReg(Y4, X4)
# Fit the model using the desired quantile
res = mod.fit(q=0.5)  # For median (50th percentile)

# Display the results
print(res.summary())
```

```
                         QuantReg Regression Results
========================================================================
=======
Dep. Variable:                  price   Pseudo R-squared:
0.7709
Model:                       QuantReg   Bandwidth:
54.45
Method:                 Least Squares   Sparsity:
946.8
Date:                Mon, 12 Feb 2024   No. Observations:
53940
Time:                        22:41:29   Df Residuals:
53917
                                        Df Model:
23
========================================================================
==========
                    coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------
----------
carat          1.293e+04     19.211    672.807      0.000    1.29e+04
1.3e+04
depth            55.0542      0.839     65.585      0.000      53.409
56.699
table           -20.2867      0.898    -22.591      0.000     -22.047
-18.527
x              -559.4452     12.904    -43.353      0.000    -584.738
-534.152
y               131.1420      8.078     16.234      0.000     115.308
146.976
z             -2586.5157     13.423   -192.698      0.000   -2612.824
-2560.207
cut_Good        358.7946     13.718     26.155      0.000     331.907
385.682
cut_Ideal       520.4984     12.547     41.485      0.000     495.907
545.090
cut_Premium     504.3973     12.709     39.688      0.000     479.487
529.307
cut_Very Good   423.3799     12.733     33.251      0.000     398.424
448.336
color_E        -138.8535      7.493    -18.530      0.000    -153.541
-124.167
color_F        -205.8204      7.578    -27.160      0.000    -220.673
-190.967
color_G        -271.0289      7.421    -36.520      0.000    -285.575
-256.483
color_H        -482.3621      7.891    -61.132      0.000    -497.828
-466.897
color_I        -842.1548      8.865    -94.997      0.000    -859.530
-824.779
color_J       -1632.7617     10.946   -149.160      0.000   -1654.217
-1611.307
clarity_IF     3311.2173     21.181    156.328      0.000    3269.702
3352.733
clarity_SI1    2582.9559     18.183    142.054      0.000    2547.317
2618.594
```

```
clarity_SI2     1899.7364      18.250     104.095       0.000     1863.966
1935.507
clarity_VS1     3040.0118      18.526     164.098       0.000     3003.702
3076.322
clarity_VS2     2910.6472      18.250     159.484       0.000     2874.876
2946.418
clarity_VVS1    3197.3169      19.584     163.260       0.000     3158.932
3235.702
clarity_VVS2    3168.3902      19.055     166.274       0.000     3131.042
3205.739
========================================================================
==========

The condition number is large, 1.99e+03. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

In [259]:
```
# Calculate predicted prices
# Assuming 'res' is the result object from fitting your quantile regress
predicted_prices = X4_test.dot(res.params)
```

In [262]:
```
import pandas as pd

# Create a DataFrame with predictions
test_output4 = pd.DataFrame(predicted_prices, columns=['pred_price'])
test_output4 = test_output4.merge(y4_test, left_index = True, right_inde
test_output4.head()
```

Out[262]:

	pred_price	price
44751	1,680.79	1619
22963	12,603.92	11011
9078	5,096.39	4521
26148	15,062.74	15454
29451	430.84	702

In [267]:
```
mean_absolute_error4 = abs(test_output4['pred_price'] - test_output4['pr
print('Mean absolute error is ')
print(mean_absolute_error4)
```

```
Mean absolute error is
644.5251556319222
```

In [268]:
```
average_price_test4 = y4_test.mean()
```

In [269]:
```
fraction_mae4 = mean_absolute_error4 / average_price_test4
print("Fraction of MAE to Average Price:", fraction_mae4)
```

```
Fraction of MAE to Average Price: 0.1640336350906855
```