In [404…

```
#There isnt enough data
import numpy as np
import pandas as pd

wine = pd.read_excel(r"C:\Users\djbro\OneDrive\Desktop\Multiple Linear Regression\Wine\W
wine1 = pd.read_excel(r"C:\Users\djbro\OneDrive\Desktop\Multiple Linear Regression\Wine\
```

In [405…

```
#Concatenate wine and wine1
a=[wine,wine1]
wine = pd.concat(a)
wine
```

Out[405]:

	Year	Price	WinterRain	AGST	HarvestRain	Age	FrancePop
0	1979	6.9541	717	16.1667	122	4	54835.832
1	1980	6.4979	578	16.0000	74	3	55110.236
0	1952	7.4950	600	17.1167	160	31	43183.569
1	1953	8.0393	690	16.7333	80	30	43495.030
2	1955	7.6858	502	17.1500	130	28	44217.857
3	1957	6.9845	420	16.1333	110	26	45152.252
4	1958	6.7772	582	16.4167	187	25	45653.805
5	1959	8.0757	485	17.4833	187	24	46128.638
6	1960	6.5188	763	16.4167	290	23	46583.995
7	1961	8.4937	830	17.3333	38	22	47128.005
8	1962	7.3880	697	16.3000	52	21	48088.673
9	1963	6.7127	608	15.7167	155	20	48798.990
10	1964	7.3094	402	17.2667	96	19	49356.943
11	1965	6.2518	602	15.3667	267	18	49801.821
12	1966	7.7443	819	16.5333	86	17	50254.966
13	1967	6.8398	714	16.2333	118	16	50650.406
14	1968	6.2435	610	16.2000	292	15	51034.413
15	1969	6.3459	575	16.5500	244	14	51470.276
16	1970	7.5883	622	16.6667	89	13	51918.389
17	1971	7.1934	551	16.7667	112	12	52431.647
18	1972	6.2049	536	14.9833	158	11	52894.183
19	1973	6.6367	376	17.0667	123	10	53332.805
20	1974	6.2941	574	16.3000	184	9	53689.610
21	1975	7.2920	572	16.9500	171	8	53955.042
22	1976	7.1211	418	17.6500	247	7	54159.049
23	1977	6.2587	821	15.5833	87	6	54378.362
24	1978	7.1860	763	15.8167	51	5	54602.193

In [406…

```
import seaborn as sns
```

```
#sns.pairplot(wine)
```

In [407…

```
# Checking for null values
print(wine.info())

# Checking for outliers
print(wine.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27 entries, 0 to 24
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Year         27 non-null     int64  
 1   Price        27 non-null     float64
 2   WinterRain   27 non-null     int64  
 3   AGST         27 non-null     float64
 4   HarvestRain  27 non-null     int64  
 5   Age          27 non-null     int64  
 6   FrancePop    27 non-null     float64
dtypes: float64(3), int64(4)
memory usage: 1.7 KB
None
              Year      Price  WinterRain       AGST  HarvestRain        Age  \
count    27.000000  27.000000   27.000000  27.000000    27.000000  27.000000   
mean   1966.814815   7.041948  608.407407  16.477781   144.814815  16.185185   
std       8.246384   0.634590  129.034956   0.659189    73.065849   8.246384   
min    1952.000000   6.204900  376.000000  14.983300    38.000000   3.000000   
25%    1960.500000   6.508350  543.500000  16.150000    88.000000   9.500000   
50%    1967.000000   6.984500  600.000000  16.416700   123.000000  16.000000   
75%    1973.500000   7.441500  705.500000  17.008350   185.500000  22.500000   
max    1980.000000   8.493700  830.000000  17.650000   292.000000  31.000000   

          FrancePop  
count     27.000000  
mean   50085.443963  
std     3792.998764  
min    43183.569000  
25%    46856.000000  
50%    50650.406000  
75%    53511.207500  
max    55110.236000  
```

In [408…

```
from sklearn.model_selection import train_test_split

# We specify random seed so that the train and test data set always have the same rows,
np.random.seed(0)
df_train, df_test = train_test_split(wine, train_size = 0.7, test_size = 0.3, random_sta
```

In [409…

```
#Re-scaling the Features
#We can see that all the columns have
#smaller integer values in the dataset
#except the area column. So it is important to
#re-scale the variables so that they all have a comparable scale.
#If we don’t have relative scales, then some of the regression model
#coefficients will be of different units compared to the other coefficients.

#To do that, we use the MinMax scaling method.
```

In [410…

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Applying scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['Year','Price','WinterRain','AGST','HarvestRain','Age','FrancePop']
df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
```

```
df_train
```

Out[410]:

	Year	Price	WinterRain	AGST	HarvestRain	Age	FrancePop
18	0.703704	0.000000	0.352423	0.00000	0.472441	0.296296	0.809211
14	0.555556	0.016865	0.515419	0.48668	1.000000	0.444444	0.649096
1	1.000000	0.128015	0.444934	0.40668	0.141732	0.000000	1.000000
8	0.333333	0.516908	0.707048	0.52668	0.055118	0.666667	0.395485
24	0.925926	0.428653	0.852423	0.33336	0.051181	0.074074	0.956261
23	0.888889	0.023506	0.980176	0.24000	0.192913	0.111111	0.936990
6	0.259259	0.137146	0.852423	0.57336	0.992126	0.740741	0.265941
4	0.185185	0.250044	0.453744	0.57336	0.586614	0.814815	0.185858
2	0.074074	0.647020	0.277533	0.86668	0.362205	0.925926	0.062231
16	0.629630	0.604422	0.541850	0.67336	0.200787	0.370370	0.725201
7	0.296296	1.000000	1.000000	0.94000	0.000000	0.703704	0.312777
5	0.222222	0.817372	0.240088	1.00000	0.586614	0.777778	0.226738
20	0.777778	0.038972	0.436123	0.52668	0.574803	0.222222	0.877693
1	0.000000	0.801468	0.691630	0.70000	0.165354	1.000000	0.000000
0	0.962963	0.327333	0.751101	0.47336	0.330709	0.037037	0.976375
19	0.740741	0.188658	0.000000	0.83336	0.334646	0.259259	0.846974
13	0.518519	0.277394	0.744493	0.50000	0.314961	0.481481	0.616035
10	0.407407	0.482567	0.057269	0.91336	0.228346	0.592593	0.504676

In [411…

```
# Dividing the training data set into X and Y
y_train = df_train.pop('Price')
X_train = df_train
```

In [412…

```
#Build a linear model

import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)

lr_1 = sm.OLS(y_train, X_train_lm).fit()

lr_1.summary()
```

```
C:\Users\djbro\anaconda3\lib\site-packages\scipy\stats\_stats_py.py:1772: UserWarning: k
urtosistest only valid for n>=20 ... continuing anyway, n=18
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[412]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.855
Model:	OLS	**Adj. R-squared:**	0.795
Method:	Least Squares	**F-statistic:**	14.17
Date:	Fri, 23 Dec 2022	**Prob (F-statistic):**	0.000111
Time:	18:09:13	**Log-Likelihood:**	13.551
No. Observations:	18	**AIC:**	-15.10

Df Residuals:	12	**BIC:**	-9.760
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-7.523e+12	1.07e+13	-0.706	0.494	-3.07e+13	1.57e+13
Year	7.523e+12	1.07e+13	0.706	0.494	-1.57e+13	3.07e+13
WinterRain	0.2413	0.157	1.537	0.150	-0.101	0.583
AGST	0.6482	0.186	3.486	0.004	0.243	1.053
HarvestRain	-0.5362	0.145	-3.709	0.003	-0.851	-0.221
Age	7.523e+12	1.07e+13	0.706	0.494	-1.57e+13	3.07e+13
FrancePop	-0.4490	0.902	-0.498	0.627	-2.413	1.515

Omnibus:	1.522	**Durbin-Watson:**	1.499
Prob(Omnibus):	0.467	**Jarque-Bera (JB):**	0.899
Skew:	0.091	**Prob(JB):**	0.638
Kurtosis:	1.920	**Cond. No.**	9.09e+14

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.73e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [413…

```
#Recursive Feature Elimination (RFE)
#RFE is an automatic process where we don't need to select
#variables manually. We follow the same steps we have done earlier
#until Re-scaling the features and dividing the data into X and Y.

#We will use the LinearRegression function from sklearn
#for RFE (which is a utility from sklearn)
```

In [414…

```
# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

In [415…

```
# Running RFE with the output number of the variable equal to 10
lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm,n_features_to_select=10)             # running RFE
rfe = rfe.fit(X_train, y_train)

list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

Out[415]:

```
[('Year', True, 1),
 ('WinterRain', True, 1),
 ('AGST', True, 1),
 ('HarvestRain', True, 1),
 ('Age', True, 1),
 ('FrancePop', True, 1)]
```

In [416…
```python
# Creating X_test dataframe with RFE selected variables
col = ['Year','WinterRain','AGST','HarvestRain','Age','FrancePop']
X_train_rfe = X_train[col]

# Adding a constant variable
import statsmodels.api as sm
X_train_rfe = sm.add_constant(X_train_rfe)

lm = sm.OLS(y_train,X_train_rfe).fit()   # Running the linear model

print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.855
Model:                            OLS   Adj. R-squared:                  0.795
Method:                 Least Squares   F-statistic:                     14.17
Date:                Fri, 23 Dec 2022   Prob (F-statistic):           0.000111
Time:                        18:09:13   Log-Likelihood:                 13.551
No. Observations:                  18   AIC:                            -15.10
Df Residuals:                      12   BIC:                            -9.760
Df Model:                           5
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const       -7.523e+12   1.07e+13     -0.706      0.494   -3.07e+13    1.57e+13
Year         7.523e+12   1.07e+13      0.706      0.494   -1.57e+13    3.07e+13
WinterRain      0.2413      0.157      1.537      0.150      -0.101       0.583
AGST            0.6482      0.186      3.486      0.004       0.243       1.053
HarvestRain    -0.5362      0.145     -3.709      0.003      -0.851      -0.221
Age          7.523e+12   1.07e+13      0.706      0.494   -1.57e+13    3.07e+13
FrancePop      -0.4490      0.902     -0.498      0.627      -2.413       1.515
==============================================================================
Omnibus:                        1.522   Durbin-Watson:                   1.499
Prob(Omnibus):                  0.467   Jarque-Bera (JB):                0.899
Skew:                           0.091   Prob(JB):                        0.638
Kurtosis:                       1.920   Cond. No.                     9.09e+14
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The smallest eigenvalue is 5.73e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

```
C:\Users\djbro\anaconda3\lib\site-packages\scipy\stats\_stats_py.py:1772: UserWarning: k
urtosistest only valid for n>=20 ... continuing anyway, n=18
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

In [417…
```python
X_train_new = X_train_rfe.drop(["Year"], axis = 1)
#X_train_new = X_train_rfe
# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new)

lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.848
Model:                            OLS   Adj. R-squared:                  0.785
Method:                 Least Squares   F-statistic:                     13.38
Date:                Fri, 23 Dec 2022   Prob (F-statistic):           0.000147
Time:                        18:09:13   Log-Likelihood:                 13.114
No. Observations:                  18   AIC:                            -14.23
```

```
Df Residuals:                      12   BIC:                            -8.886
Df Model:                           5
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.5737      1.005      0.571      0.579      -1.617       2.764
WinterRain      0.1900      0.143      1.332      0.208      -0.121       0.501
AGST            0.6233      0.187      3.332      0.006       0.216       1.031
HarvestRain    -0.4856      0.129     -3.774      0.003      -0.766      -0.205
Age            -0.2898      0.937     -0.309      0.762      -2.331       1.751
FrancePop      -0.6222      0.889     -0.700      0.497      -2.559       1.314
==============================================================================
Omnibus:                        1.931   Durbin-Watson:                   1.696
Prob(Omnibus):                  0.381   Jarque-Bera (JB):                1.281
Skew:                           0.402   Prob(JB):                        0.527
Kurtosis:                       1.970   Cond. No.                         74.0
==============================================================================

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

```
C:\Users\djbro\anaconda3\lib\site-packages\scipy\stats\_stats_py.py:1772: UserWarning: k
urtosistest only valid for n>=20 ... continuing anyway, n=18
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

In [418…

```python
X_train_new = X_train_new.drop(["WinterRain"], axis = 1)

# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new)

lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.825
Model:                            OLS   Adj. R-squared:                  0.772
Method:                 Least Squares   F-statistic:                     15.37
Date:                Fri, 23 Dec 2022   Prob (F-statistic):           7.52e-05
Time:                        18:09:13   Log-Likelihood:                 11.873
No. Observations:                  18   AIC:                            -13.75
Df Residuals:                      13   BIC:                            -9.295
Df Model:                           4
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           1.2164      0.908      1.340      0.203      -0.745       3.178
AGST            0.5132      0.173      2.971      0.011       0.140       0.886
HarvestRain    -0.5278      0.128     -4.110      0.001      -0.805      -0.250
Age            -0.7155      0.906     -0.789      0.444      -2.673       1.243
FrancePop      -1.0687      0.847     -1.261      0.229      -2.899       0.762
==============================================================================
Omnibus:                        4.628   Durbin-Watson:                   1.770
Prob(Omnibus):                  0.099   Jarque-Bera (JB):                1.452
Skew:                           0.088   Prob(JB):                        0.484
Kurtosis:                       1.620   Cond. No.                         63.1
==============================================================================

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

```
C:\Users\djbro\anaconda3\lib\site-packages\scipy\stats\_stats_py.py:1772: UserWarning: k
```

```
urtosistest only valid for n>=20 ... continuing anyway, n=18
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

In [419…

```
vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
```

```
      Features         VIF
0        const  684.789942
1         AGST    1.598020
2  HarvestRain    1.101181
3          Age   65.360862
4    FrancePop   64.139693
```

In [420…

```
X_train_new = X_train_new.drop(["FrancePop"], axis = 1)

# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new)

lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model
print(lm.summary())
```

```
C:\Users\djbro\anaconda3\lib\site-packages\scipy\stats\_stats_py.py:1772: UserWarning: k
urtosistest only valid for n>=20 ... continuing anyway, n=18
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.804
Model:                            OLS   Adj. R-squared:                  0.762
Method:                 Least Squares   F-statistic:                     19.16
Date:                Fri, 23 Dec 2022   Prob (F-statistic):           3.17e-05
Time:                        18:09:13   Log-Likelihood:                 10.834
No. Observations:                  18   AIC:                            -13.67
Df Residuals:                      14   BIC:                            -10.11
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.0782      0.103      0.762      0.459      -0.142       0.298
AGST            0.5103      0.176      2.894      0.012       0.132       0.888
HarvestRain    -0.5511      0.130     -4.249      0.001      -0.829      -0.273
Age             0.4132      0.147      2.813      0.014       0.098       0.728
==============================================================================
Omnibus:                        1.570   Durbin-Watson:                   1.872
Prob(Omnibus):                  0.456   Jarque-Bera (JB):                0.917
Skew:                           0.107   Prob(JB):                        0.632
Kurtosis:                       1.915   Cond. No.                         8.22
==============================================================================

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

In [421…

```
vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
```

```
      Features       VIF
0        const  8.392431
1         AGST  1.597728
```

```
2  HarvestRain  1.078253
3          Age  1.646593
```

In [422…

```
X_train_new = X_train_new.drop(["const"], axis = 1)

# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new)

lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model
print(lm.summary())

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.804
Model:                            OLS   Adj. R-squared:                  0.762
Method:                 Least Squares   F-statistic:                     19.16
Date:                Fri, 23 Dec 2022   Prob (F-statistic):           3.17e-05
Time:                        18:09:13   Log-Likelihood:                 10.834
No. Observations:                  18   AIC:                            -13.67
Df Residuals:                      14   BIC:                            -10.11
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const           0.0782      0.103      0.762      0.459      -0.142       0.298
AGST            0.5103      0.176      2.894      0.012       0.132       0.888
HarvestRain    -0.5511      0.130     -4.249      0.001      -0.829      -0.273
Age             0.4132      0.147      2.813      0.014       0.098       0.728
==============================================================================
Omnibus:                        1.570   Durbin-Watson:                   1.872
Prob(Omnibus):                  0.456   Jarque-Bera (JB):                0.917
Skew:                           0.107   Prob(JB):                        0.632
Kurtosis:                       1.915   Cond. No.                         8.22
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
      Features       VIF
0         AGST  5.317162
1  HarvestRain  2.188327
2          Age  5.476022
```

```
C:\Users\djbro\anaconda3\lib\site-packages\scipy\stats\_stats_py.py:1772: UserWarning: k
urtosistest only valid for n>=20 ... continuing anyway, n=18
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

In [423…

```
#Since the p-values and VIF are in the desired range, we’ll move forward with the analys
```

In [424…

```
#The next step is the residual analysis of error terms.

#Residual Analysis
#So, let’s check if the error terms are also normally distributed using a histogram.
```

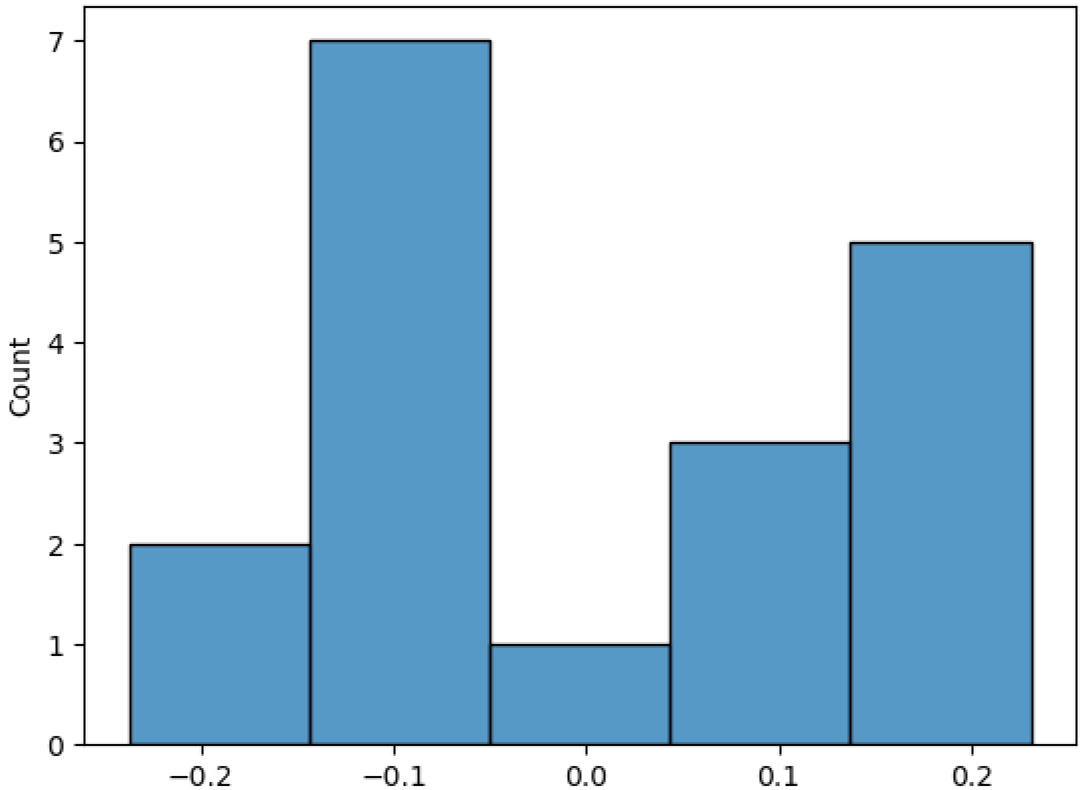
In [425…

```
y_train_price = lm.predict(X_train_lm)
# Importing the required libraries for plots.
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline

# Plot the histogram of the error terms
fig = plt.figure()
sns.histplot((y_train - y_train_price), bins = 5)
```

Out[425]: <AxesSubplot:ylabel='Count'>



In [426…

```
num_vars = ['Year','Price','WinterRain','AGST','HarvestRain','Age','FrancePop']
df_test[num_vars] = scaler.transform(df_test[num_vars])

y_test = df_test.pop('Price')
X_test = df_test

# Now let's use our model to make predictions.

# Creating X_test_new dataframe by dropping variables from X_test
X_test_new = X_test[X_train_new.columns]
# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)

# Making predictions
y_pred = lm.predict(X_test_new)
```

In [427…

```
from sklearn.metrics import r2_score
r2_score(y_true = y_test, y_pred = y_pred)
```

Out[427]: 0.5760134554123846

In [428…

```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test, y_pred))
print(metrics.mean_squared_error(y_test, y_pred))
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
0.11568220627300144
0.01802667325935115
0.13426344721982655
```

In [429…

```
#The R² value for the test data = 0.6481740917926483,
#which is pretty similar to the train data.

#Since the R² values for both the train and
#test data are almost equal, the model we built is the best-fitted model.
```