Second-Hand-Cars-Price-Predection

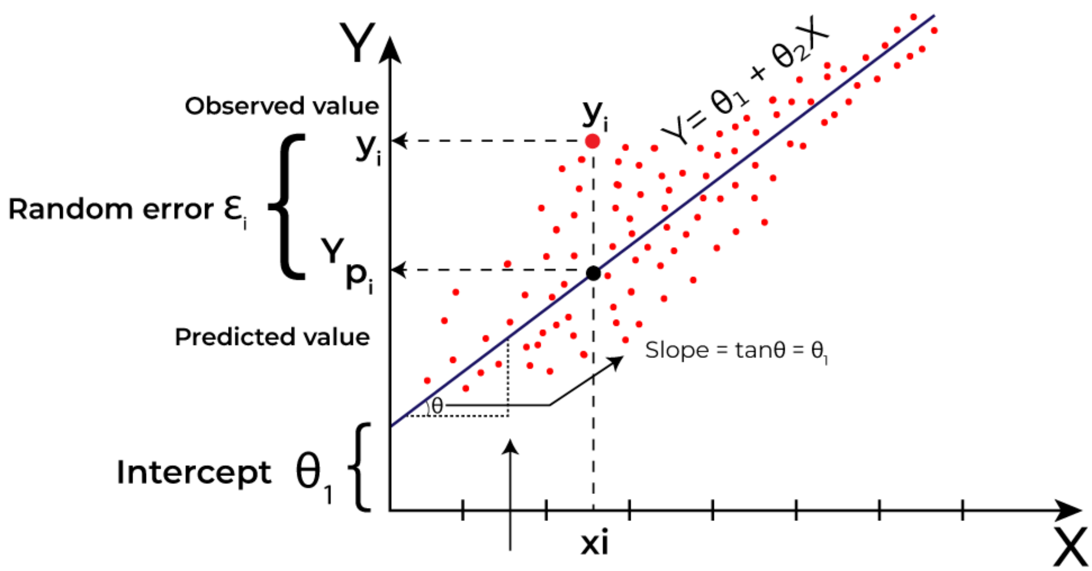
June 1, 2024

1 Objective : Price Prediction - On Second Hand Cars

2 EDA - Python

3 Insights - Patterns

4 Regression



5 1. Import Python Modules

[1]:
```python
# Load necessary python modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tabulate import tabulate
```

```python
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline


from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression,Lasso, Ridge,ElasticNet

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

6 2. Load Cars Price Dataset

[2]:
```python
file_path = r"Cars-SecondHand.xlsx"
cars_price_df = pd.read_excel(file_path)
cars_price_df
```

[2]:

	Brand	Price	Mileage	EngineV	Year
0	Mercedes-Benz	222000.0	1	6.3	2016
1	Mercedes-Benz	177000.0	1	5.5	2016
2	Mercedes-Benz	177777.0	1	5.5	2016
3	Mercedes-Benz	199999.0	1	5.5	2016
4	Mercedes-Benz	199999.0	1	5.5	2016
…	…	…	…	…	…
3998	Toyota	600.0	10	1.5	1979
3999	Mercedes-Benz	2990.0	300	2.8	1979
4000	Mercedes-Benz	2300.0	261	2.3	1978
4001	Mercedes-Benz	5500.0	440	2.0	1978
4002	Mercedes-Benz	34999.0	150	2.8	1969

[4003 rows x 5 columns]

7 3. Basic Inspection on dataset

[3]:
```python
def basic_inspection_dataset(table):
    """Generates a basic inspection dataset from the given table."""

    print("top 5 rows - using head")
    print(table.head())
    print()

    print("bottom 5 rows using tail")
    print(table.tail())
    print()

    print("numbers of samples and columns")
    print(table.shape)
    print()

    print("numbers of samples ")
    print(len(table))
    print()

    print("numbers of entries in the data frame")
    print(table.size)
    print()

    print("Columns Names")
    print(table.columns)
    print()

    print("Columns dtypes")
    print(table.dtypes)
    print()

    print("Dataframe info")
    print(table.info())
    print()

    print()
    print("check the missing value in each column")
    print(table.isnull().sum())

    print()
    print("check the missing value in each column")
    print(table.isna().sum())

    print()
    print("table summary ")
```

```
    print(table.describe())

basic_inspection_dataset(cars_price_df)
```

```
top 5 rows - using head
           Brand     Price  Mileage  EngineV  Year
0  Mercedes-Benz  222000.0        1      6.3  2016
1  Mercedes-Benz  177000.0        1      5.5  2016
2  Mercedes-Benz  177777.0        1      5.5  2016
3  Mercedes-Benz  199999.0        1      5.5  2016
4  Mercedes-Benz  199999.0        1      5.5  2016

bottom 5 rows using tail
              Brand    Price  Mileage  EngineV  Year
3998         Toyota    600.0       10      1.5  1979
3999  Mercedes-Benz   2990.0      300      2.8  1979
4000  Mercedes-Benz   2300.0      261      2.3  1978
4001  Mercedes-Benz   5500.0      440      2.0  1978
4002  Mercedes-Benz  34999.0      150      2.8  1969

numbers of samples and columns
(4003, 5)

numbers of samples
4003

numbers of entries in the data frame
20015

Columns Names
Index(['Brand', 'Price', 'Mileage', 'EngineV', 'Year'], dtype='object')

Columns dtypes
Brand       object
Price      float64
Mileage      int64
EngineV    float64
Year         int64
dtype: object

Dataframe info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4003 entries, 0 to 4002
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Brand    4003 non-null   object
 1   Price    4003 non-null   float64
```

```
 2   Mileage  4003 non-null   int64
 3   EngineV  4003 non-null   float64
 4   Year     4003 non-null   int64
dtypes: float64(2), int64(2), object(1)
memory usage: 156.5+ KB
None


check the missing value in each column
Brand      0
Price      0
Mileage    0
EngineV    0
Year       0
dtype: int64

check the missing value in each column
Brand      0
Price      0
Mileage    0
EngineV    0
Year       0
dtype: int64

table summary
               Price      Mileage      EngineV         Year
count    4003.000000  4003.000000  4003.000000  4003.000000
mean    19619.014218   163.419935     2.467732  2006.395703
std     25868.124801   103.406160     0.975549     6.695288
min       600.000000     1.000000     0.600000  1969.000000
25%      7000.000000    90.000000     1.800000  2003.000000
50%     11500.000000   158.000000     2.200000  2008.000000
75%     21900.000000   230.000000     3.000000  2012.000000
max    300000.000000   980.000000     6.500000  2016.000000
```

8 4. Handling Missing Values - Cat

[4]:
```
# There is no missing values in cat columns
```

9 5. Categorical- Variable - Analysis -Using Pipeline

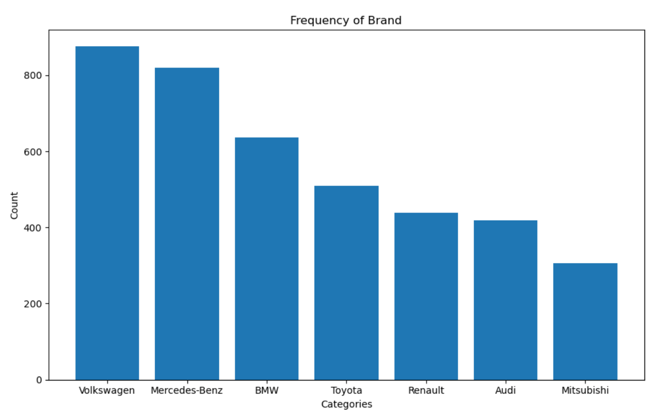
[2]:
```
class BarPieChartTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
```

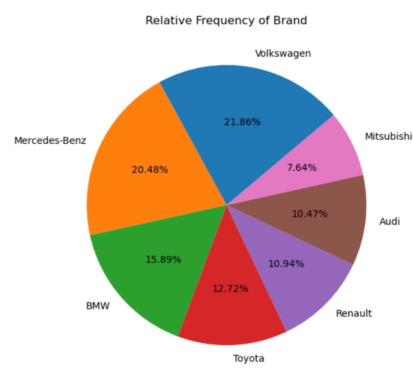
```
        return self

    def transform(self, X):
        df=X.copy()
        cat_cols = df.select_dtypes(include='object').columns
        for cat_name in cat_cols:
            value_counts = df[cat_name].value_counts().reset_index()
            # Rename the columns
            value_counts.columns = ['Class', 'Frequency']

            # Print the result as a table
            print(f"{cat_name} frequency table")
            print(tabulate(value_counts, headers='keys', tablefmt='pretty'))

            # Calculate relative frequency
            total_count = value_counts['Frequency'].sum()
            value_counts['Relative Frequency %'] =␣
↪round((value_counts['Frequency'] / total_count)*100,2)

            # Print the result as a table
            print(f"{cat_name} Relative frequency table")
            print(tabulate(value_counts, headers='keys', tablefmt='pretty'))

            # Extract the values and index from value counts
            value_counts = df[cat_name].value_counts()
            values = value_counts.values
            labels = value_counts.index

            fig, axs = plt.subplots(1, 2, figsize=(18, 6))  # 1 row, 2 columns
            # Create a bar graph
            axs[0].bar(labels, values)
            axs[0].set_title(f'Frequency of {cat_name}')
            axs[0].set_xlabel('Categories')  # Set x-label
            axs[0].set_ylabel('Count')       # Set y-label

            axs[1].pie(value_counts.values, labels=value_counts.index,␣
↪autopct='%0.2f%%', startangle=40)
            axs[1].set_title(f'Relative Frequency of {cat_name}')
            plt.tight_layout()
            # Show the plot
            plt.show()
```

[6]:
```
pipeline_cat_var = Pipeline([
    ('bar_pie_chart', BarPieChartTransformer())
])
```

```python
# Fit and transform your data using the pipeline
processed_data = pipeline_cat_var.fit_transform(cars_price_df)
```

```
Brand frequency table
+---+---------------+-----------+
|   |     Class     | Frequency |
+---+---------------+-----------+
| 0 |  Volkswagen   |    875    |
| 1 | Mercedes-Benz |    820    |
| 2 |      BMW      |    636    |
| 3 |    Toyota     |    509    |
| 4 |    Renault    |    438    |
| 5 |     Audi      |    419    |
| 6 |  Mitsubishi   |    306    |
+---+---------------+-----------+
Brand Relative frequency table
+---+---------------+-----------+----------------------+
|   |     Class     | Frequency | Relative Frequency % |
+---+---------------+-----------+----------------------+
| 0 |  Volkswagen   |    875    |        21.86         |
| 1 | Mercedes-Benz |    820    |        20.48         |
| 2 |      BMW      |    636    |        15.89         |
| 3 |    Toyota     |    509    |        12.72         |
| 4 |    Renault    |    438    |        10.94         |
| 5 |     Audi      |    419    |        10.47         |
| 6 |  Mitsubishi   |    306    |         7.64         |
+---+---------------+-----------+----------------------+
```





Observations 1. Volkswagen with highest numbers of sales with 875 2. Mitsubishi with lowest numbers of sales with 306

10 6. Handling Missing Values in Numerical Columns

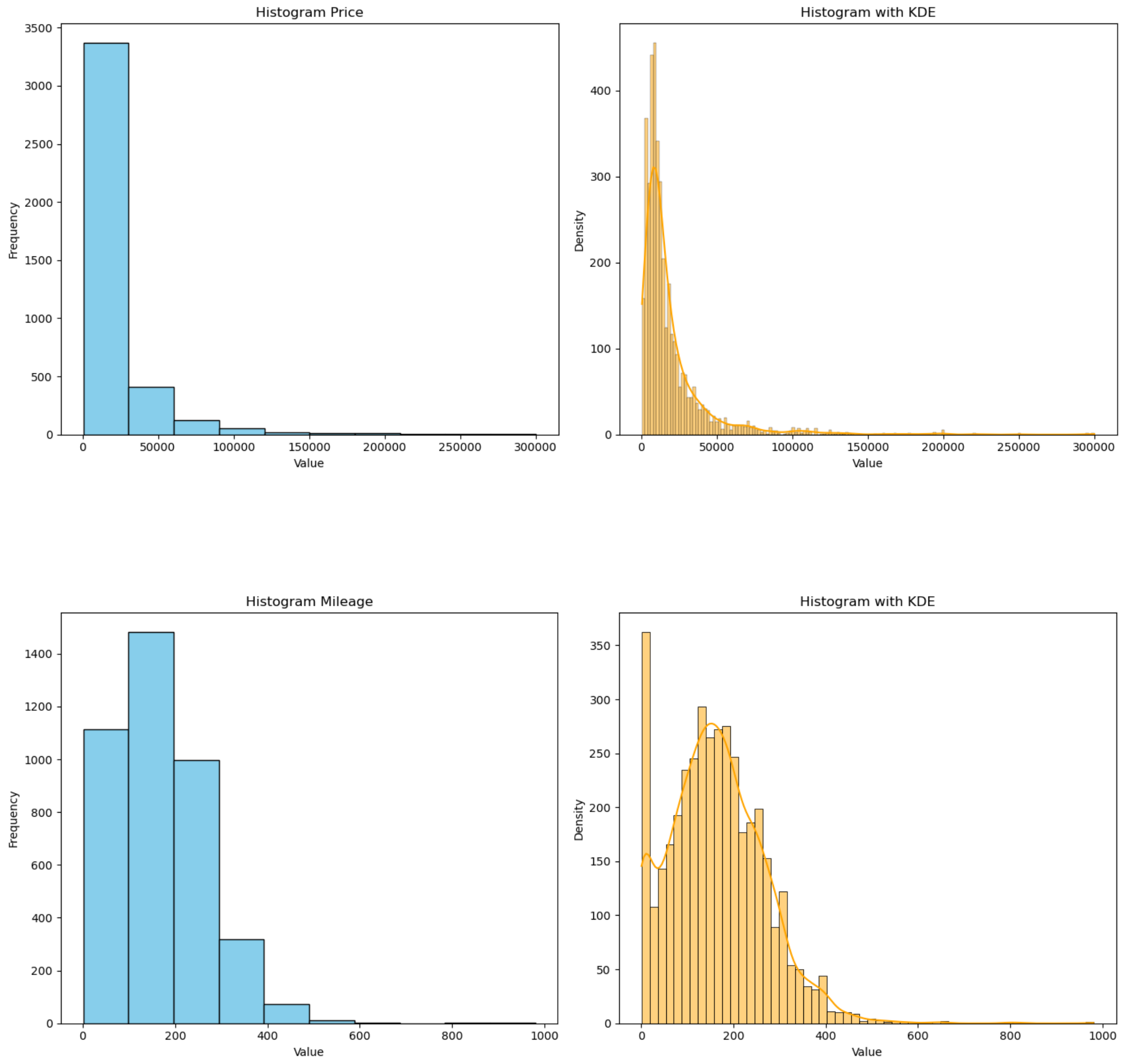
[7]:
```
# There is no missing values in num columns
```

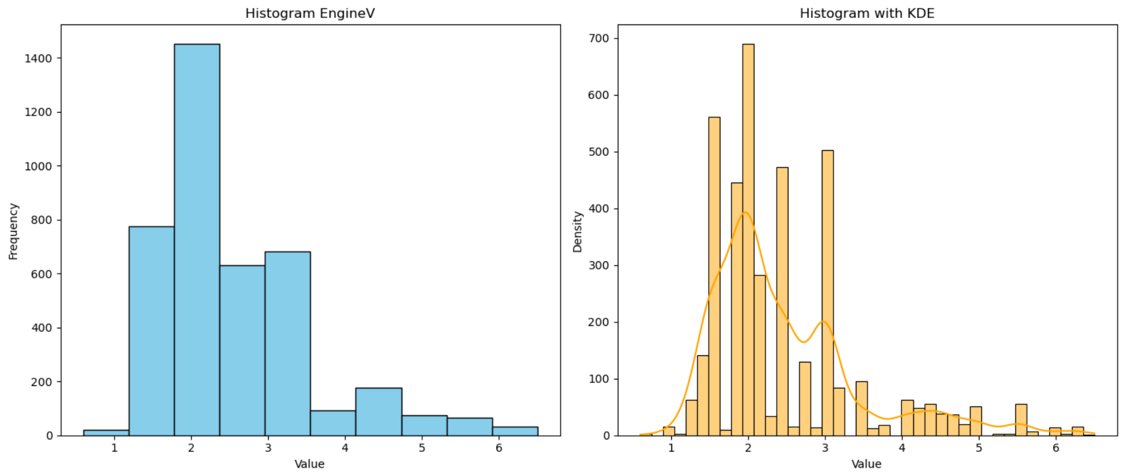
11 7. Numerical - Variables - Analysis - Using -Pipeline

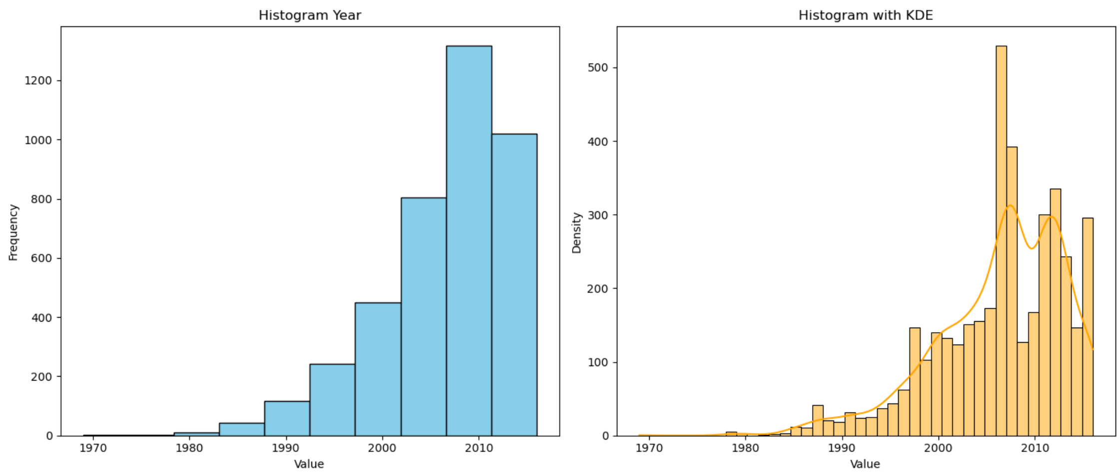
[8]:
```
class HistBoxChartTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        df=X.copy()
        num_cols = df.select_dtypes(exclude='object').columns
        for con_var in num_cols:


            # Create a figure and axes object
            fig, axes = plt.subplots(1, 2, figsize=(14, 6))

            # Plot histogram without KDE on the left
            axes[0].hist(df[con_var], color='skyblue', edgecolor='black')
            axes[0].set_xlabel('Value')
            axes[0].set_ylabel('Frequency')
            axes[0].set_title(f'Histogram {con_var}')

            # Plot histogram with KDE on the right
            sns.histplot(data=df, x=con_var, kde=True, color='orange',␣
 ↪edgecolor='black', ax=axes[1])
            axes[1].set_xlabel('Value')
            axes[1].set_ylabel('Density')
            axes[1].set_title('Histogram with KDE')

            # Adjust layout
            plt.tight_layout()

            # Show the combined plot
            plt.show()
```

[9]:
```
pipeline_num_var = Pipeline([
    ('hist_box_chart', HistBoxChartTransformer())
])
```

```python
cars_price_num_df = cars_price_df[['Price', 'Mileage', 'EngineV', 'Year']]
# Fit and transform your data using the pipeline
processed_data = pipeline_num_var.fit_transform(cars_price_num_df)
```







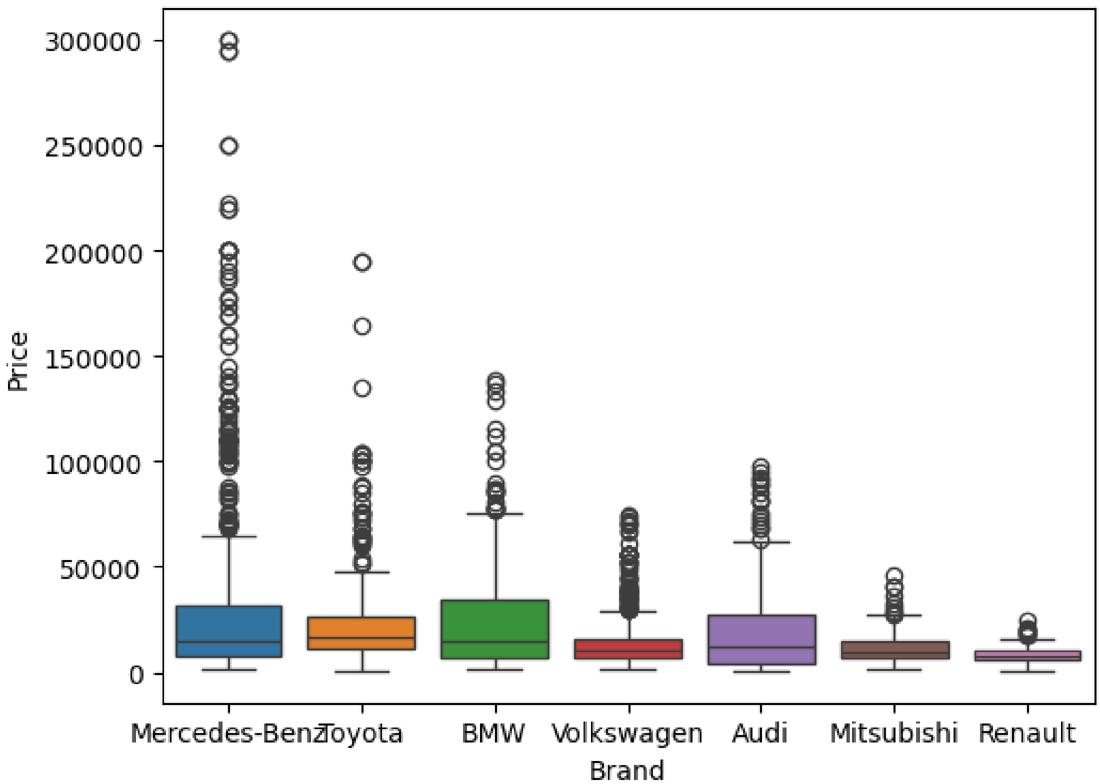
Observations 1. 'Price' , 'Mileage', 'EngineV', 'Year' are numberical columns 2. All are not normally distributed

8. Numerical - Variables -Outliers Analysis

12 9. Bi Variate Analysis

12.1 Cat Vs Num

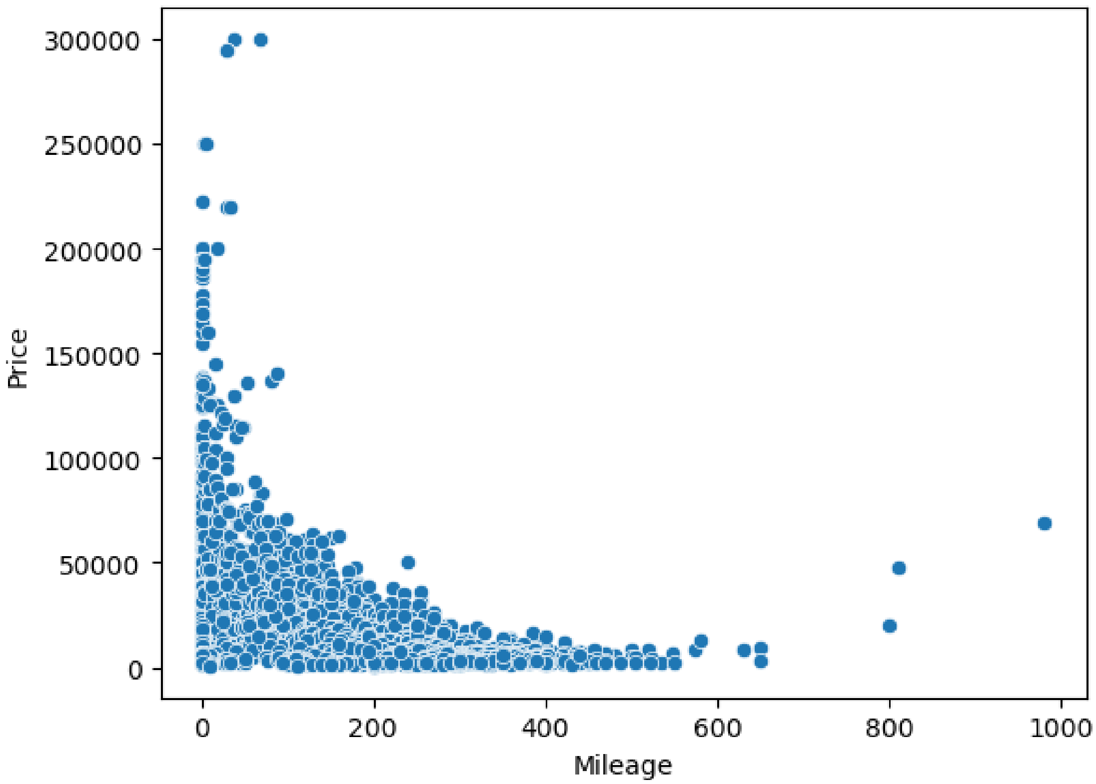
[10]:
```
# Create a box plot with hue
sns.boxplot(x='Brand' ,y='Price', hue='Brand', data=cars_price_df)
# Show the plot
plt.show()
```

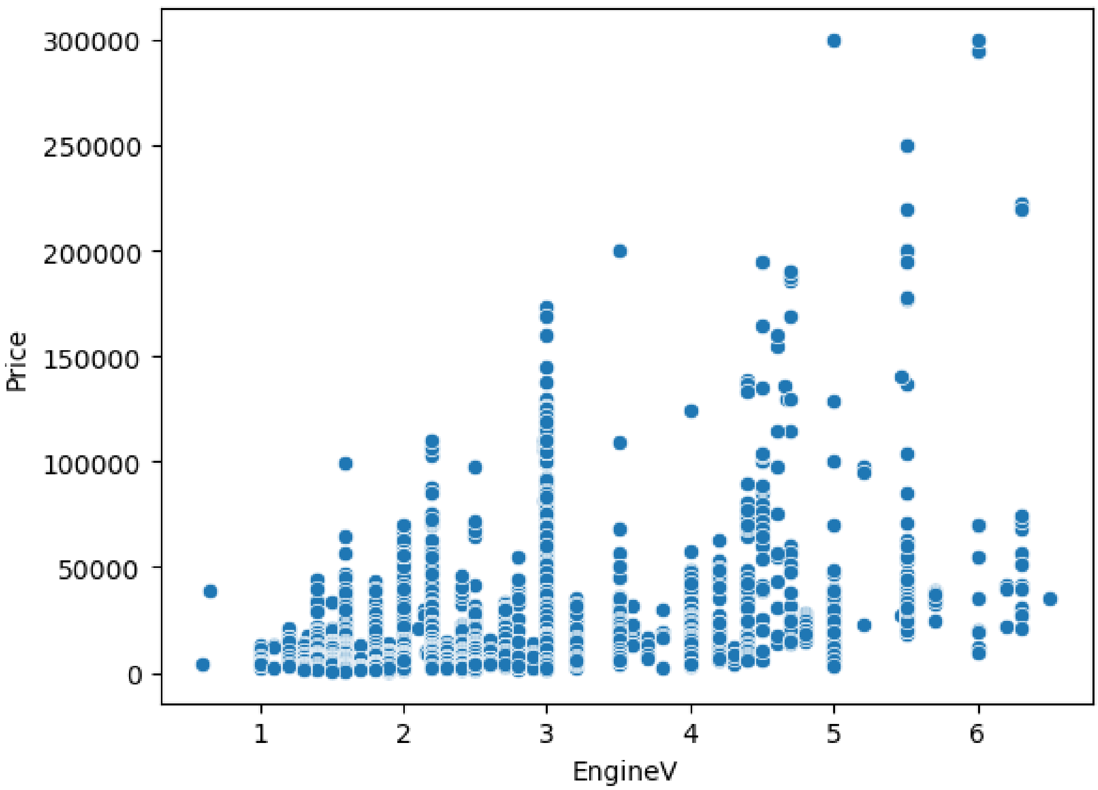


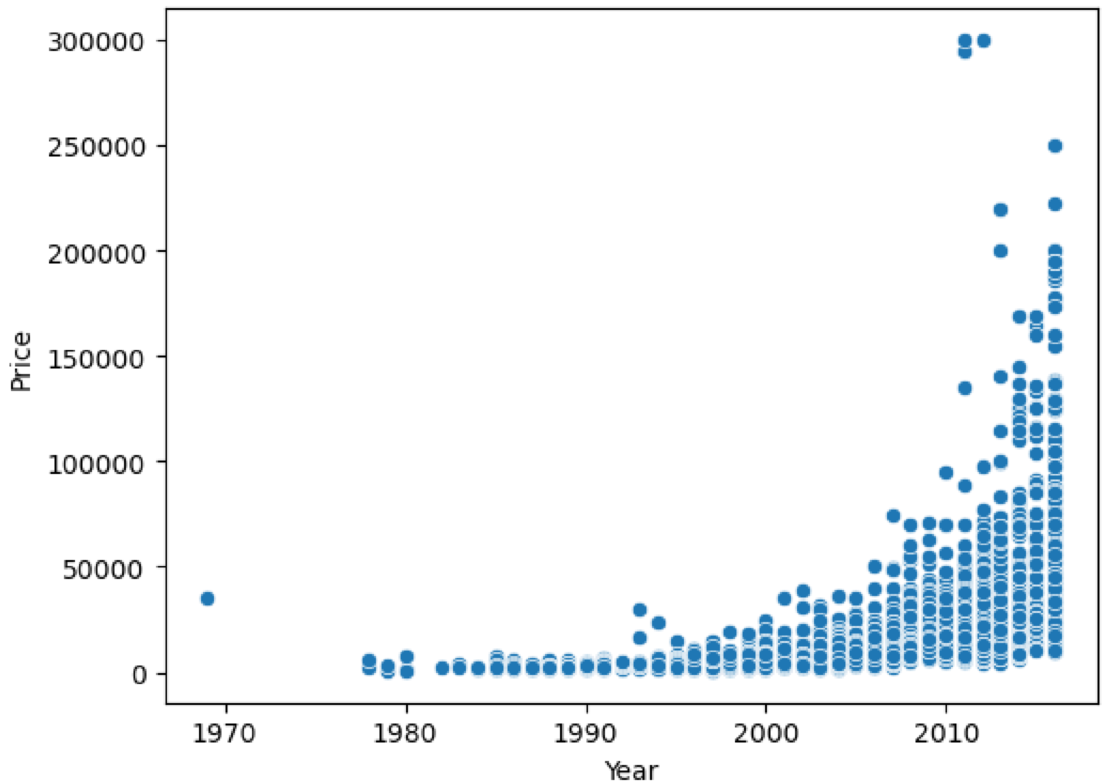
Observations 1. Mercedes-Benz with Highest price 2. Renault with lowest price 3. There are 7 brands are available in the data frame in Brand Column

12.2 Num Vs Num

[11]:
```
#print(cars_price_df.columns)
for num_var in [ 'Mileage', 'EngineV','Year']:
    sns.scatterplot(data=cars_price_df,y='Price',x=num_var)
    plt.show()
```


Price
Mileage
300000
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000


300000
250000
200000
150000
100000
50000
0
Price
1
2
3
4
5
6
EngineV



Observations 1. There is relationship b/w varaibles to price

[12]:
```
cars_price_df[['Price',  'Mileage', 'EngineV','Year']].corr()
```

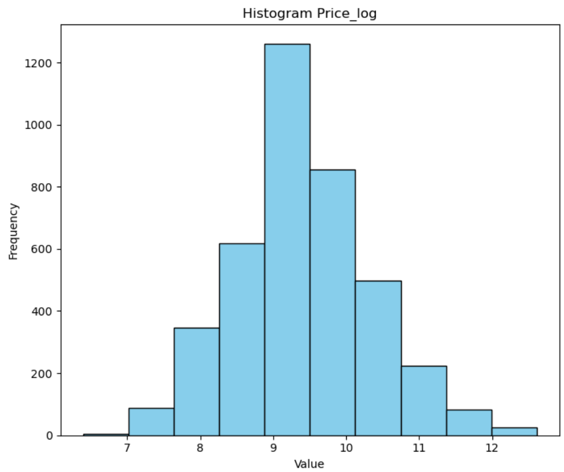
[12]:

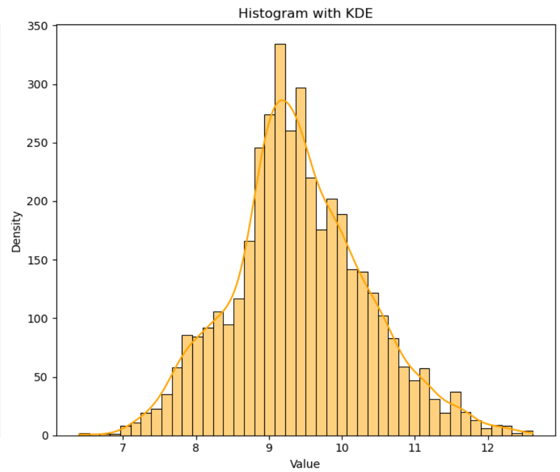
	Price	Mileage	EngineV	Year
Price	1.000000	-0.473036	0.448590	0.485717
Mileage	-0.473036	1.000000	-0.034214	-0.664027
EngineV	0.448590	-0.034214	1.000000	0.038890
Year	0.485717	-0.664027	0.038890	1.000000

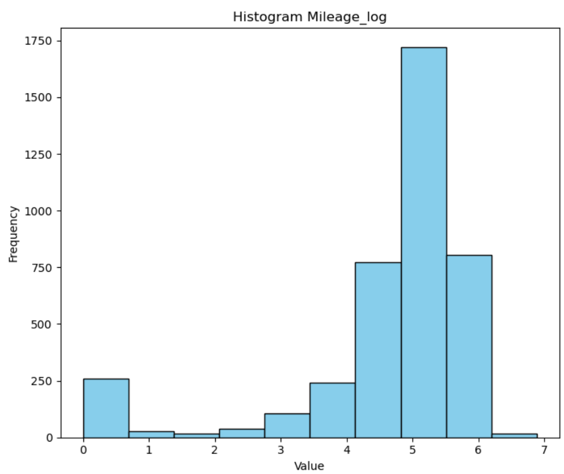
13 10. Data Transformation

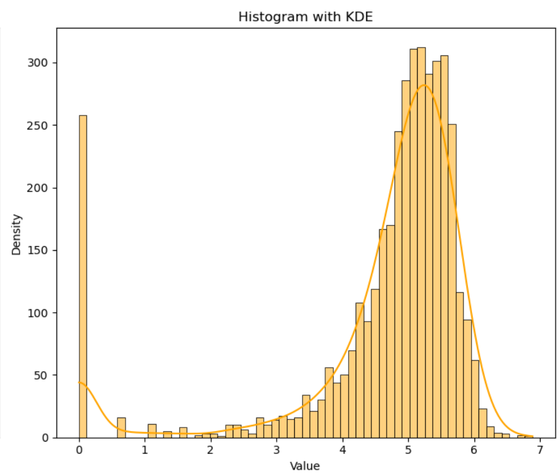
[13]:
```
cars_price_df["Mileage_log"]=np.log(cars_price_df["Mileage"])
cars_price_df["Price_log"]=np.log(cars_price_df["Price"])
cars_price_df["EngineV_log"]=np.log(cars_price_df["EngineV"])
```

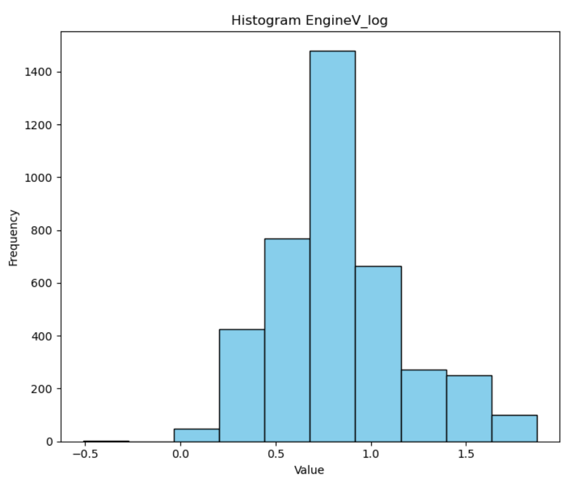
[14]:
```
cars_price_num_df = cars_price_df[['Price_log', 'Mileage_log', 'EngineV_log']].
 ↪copy()
# Fit and transform your data using the pipeline
processed_data = pipeline_num_var.fit_transform(cars_price_num_df)
```

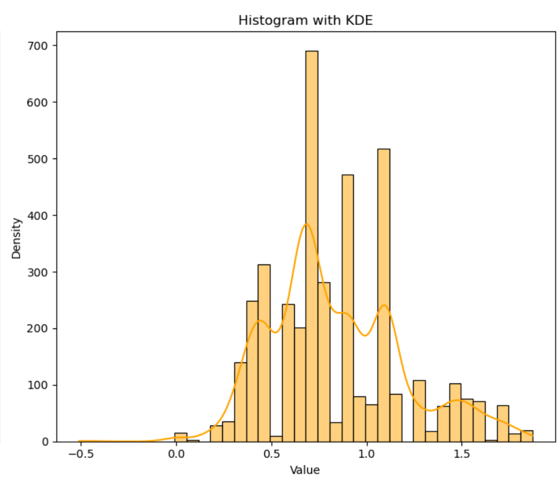

Histogram Price_log
Frequency
Value


Histogram with KDE
Density
Value


Histogram Mileage_log
Frequency
Value

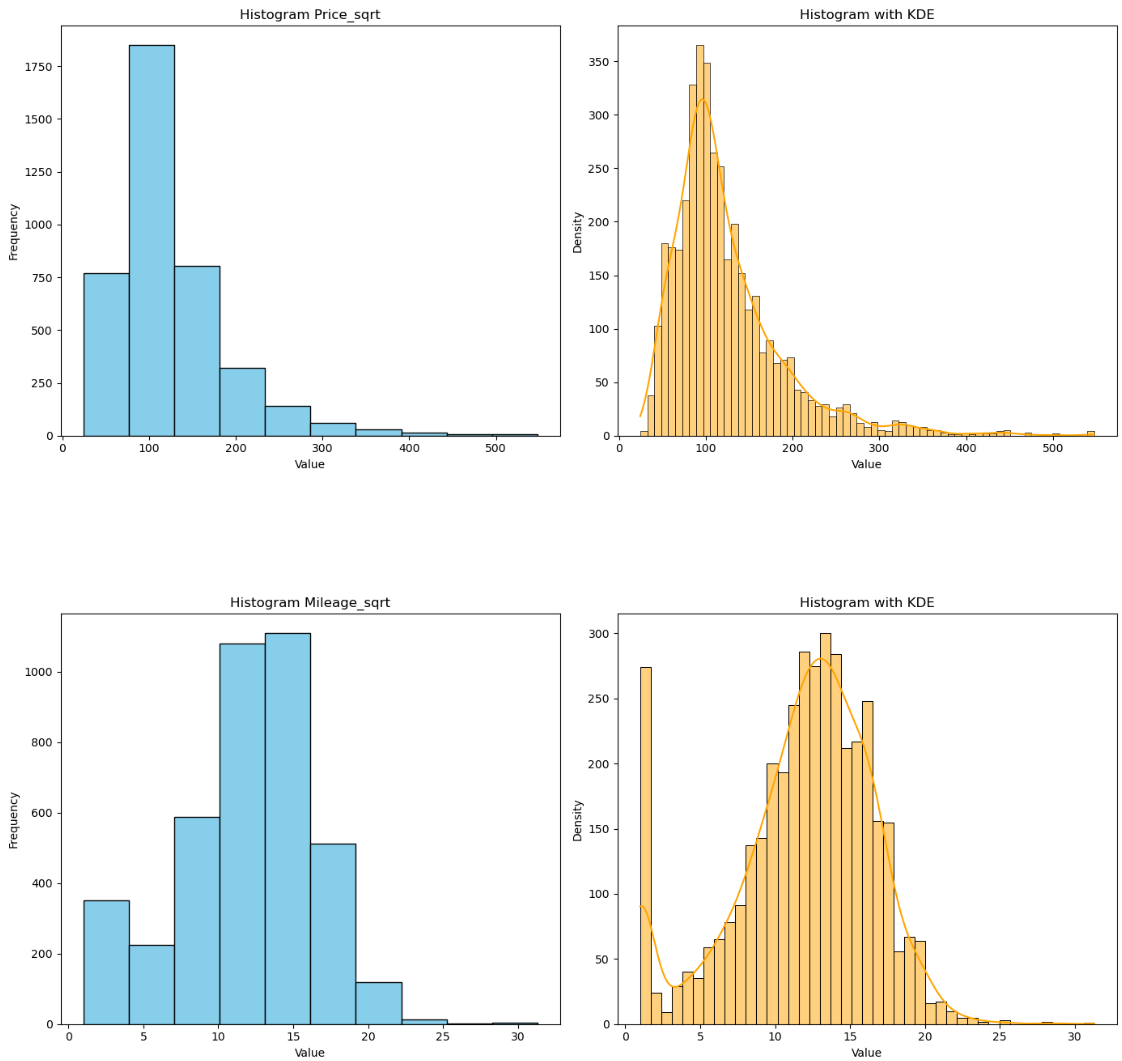

Histogram with KDE
Density
Value

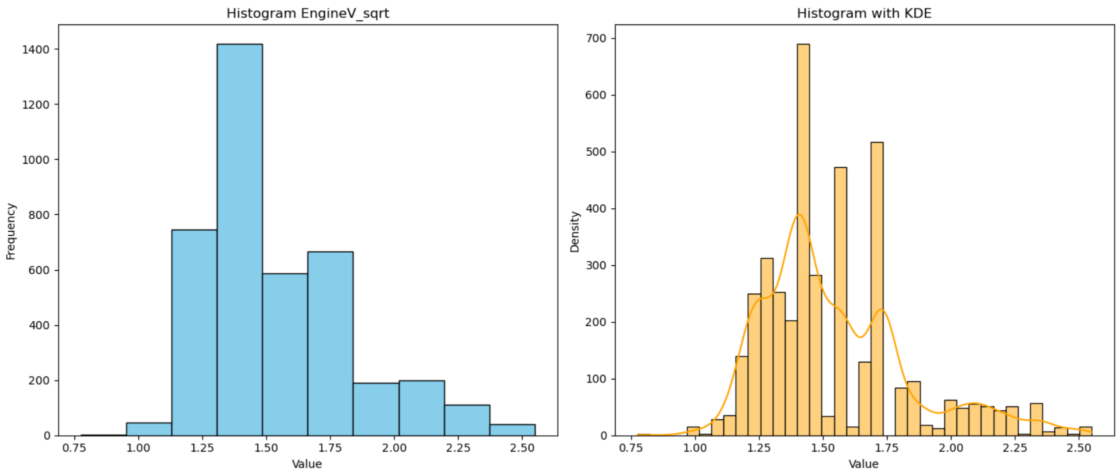

Histogram EngineV_log
Frequency
Value


Histogram with KDE
Density
Value

[15]:
```python
cars_price_df["Mileage_sqrt"]=np.sqrt(cars_price_df["Mileage"])
cars_price_df["Price_sqrt"]=np.sqrt(cars_price_df["Price"])
cars_price_df["EngineV_sqrt"]=np.sqrt(cars_price_df["EngineV"])
```

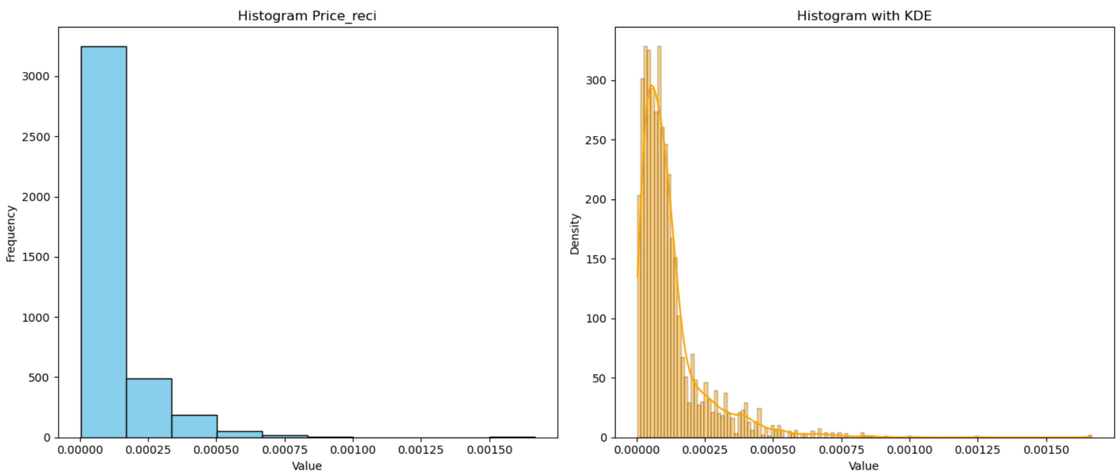
[16]:
```python
cars_price_num_df = cars_price_df[['Price_sqrt', 'Mileage_sqrt', 'EngineV_sqrt']].copy()
# Fit and transform your data using the pipeline
processed_data = pipeline_num_var.fit_transform(cars_price_num_df)
```

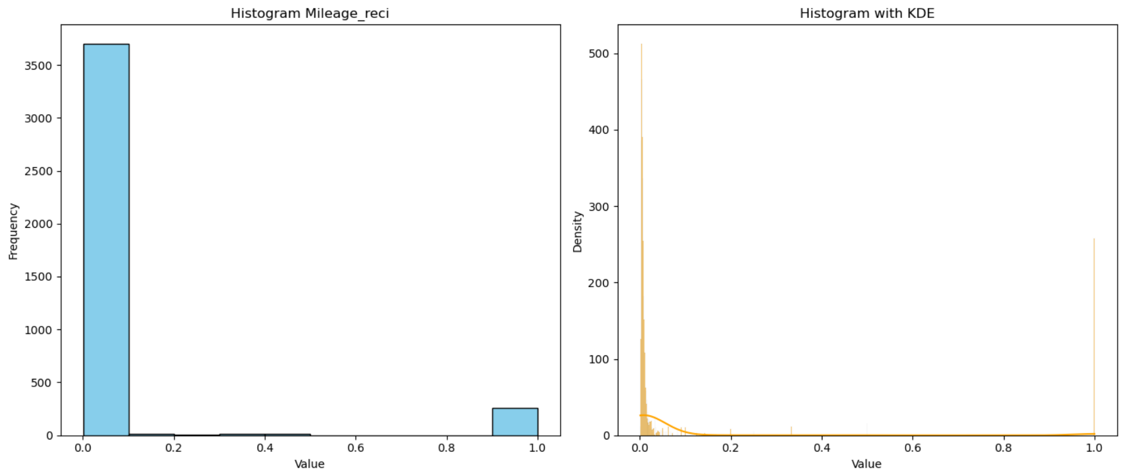


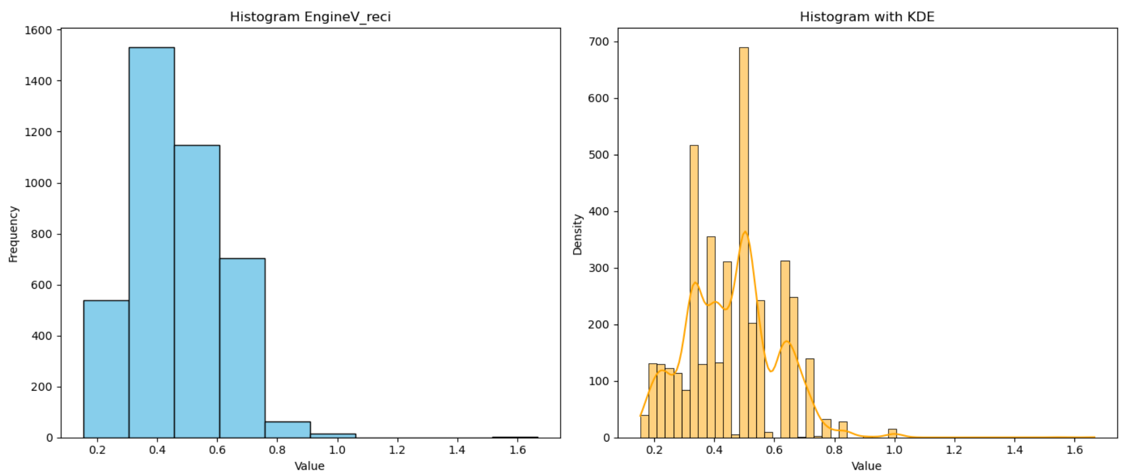


[17]:
```
cars_price_df["Mileage_reci"]=1/(cars_price_df["Mileage"])
cars_price_df["Price_reci"]=1/(cars_price_df["Price"])
cars_price_df["EngineV_reci"]=1/(cars_price_df["EngineV"])
```

[18]:
```
cars_price_num_df = cars_price_df[['Price_reci', 'Mileage_reci', 'EngineV_reci']].copy()
# Fit and transform your data using the pipeline
processed_data = pipeline_num_var.fit_transform(cars_price_num_df)
```







14 11. Standization - Normalization

[19]:
```
cars_price_df[['Price_log','Mileage_sqrt','EngineV_reci']]
```

[19]:

	Price_log	Mileage_sqrt	EngineV_reci
0	12.310433	1.000000	0.158730
1	12.083905	1.000000	0.181818
2	12.088285	1.000000	0.181818
3	12.206068	1.000000	0.181818
4	12.206068	1.000000	0.181818
…	…	…	…
3998	6.396930	3.162278	0.666667
3999	8.003029	17.320508	0.357143
4000	7.740664	16.155494	0.434783

```
4001   8.612503     20.976177      0.500000
4002  10.463075     12.247449      0.357143

[4003 rows x 3 columns]
```

[20]:

```
scaler = StandardScaler()

# Fit and transform the scaler on the selected columns
scaled_columns = scaler.
 ↪fit_transform(cars_price_df[['Price_log','Mileage_sqrt','EngineV_reci']])

# Replace the original columns with the scaled columns
cars_price_df[['Price_log_scaler',␣
 ↪'Mileage_sqrt_scaler','EngineV_reci_scaler']] = scaled_columns

print(cars_price_df)
```

```
              Brand     Price  Mileage  EngineV  Year  Mileage_log  Price_log  \
0     Mercedes-Benz  222000.0        1      6.3  2016     0.000000  12.310433
1     Mercedes-Benz  177000.0        1      5.5  2016     0.000000  12.083905
2     Mercedes-Benz  177777.0        1      5.5  2016     0.000000  12.088285
3     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
4     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
…               …       …        …      …     …          …          …
3998         Toyota     600.0       10      1.5  1979     2.302585   6.396930
3999  Mercedes-Benz    2990.0      300      2.8  1979     5.703782   8.003029
4000  Mercedes-Benz    2300.0      261      2.3  1978     5.564520   7.740664
4001  Mercedes-Benz    5500.0      440      2.0  1978     6.086775   8.612503
4002  Mercedes-Benz   34999.0      150      2.8  1969     5.010635  10.463075

      EngineV_log  Mileage_sqrt  Price_sqrt  EngineV_sqrt  Mileage_reci  \
0        1.840550      1.000000  471.168760      2.509980      1.000000
1        1.704748      1.000000  420.713679      2.345208      1.000000
2        1.704748      1.000000  421.636099      2.345208      1.000000
3        1.704748      1.000000  447.212477      2.345208      1.000000
4        1.704748      1.000000  447.212477      2.345208      1.000000
…             …            …           …            …             …
3998     0.405465      3.162278   24.494897      1.224745      0.100000
3999     1.029619     17.320508   54.680892      1.673320      0.003333
4000     0.832909     16.155494   47.958315      1.516575      0.003831
4001     0.693147     20.976177   74.161985      1.414214      0.002273
4002     1.029619     12.247449  187.080197      1.673320      0.006667

      Price_reci  EngineV_reci  Price_log_scaler  Mileage_sqrt_scaler  \
0       0.000005      0.158730          3.104998            -2.301687
1       0.000006      0.181818          2.862201            -2.301687
2       0.000006      0.181818          2.866896            -2.301687
3       0.000005      0.181818          2.993137            -2.301687
```

```
4       0.000005       0.181818           2.993137             -2.301687
…           …            …                  …                    …
3998    0.001667       0.666667          -3.233208             -1.844164
3999    0.000334       0.357143          -1.511761              1.151619
4000    0.000435       0.434783          -1.792968              0.905111
4001    0.000182       0.500000          -0.858514              1.925134
4002    0.000029       0.357143           1.124964              0.078195

      EngineV_reci_scaler
0               -2.000010
1               -1.845941
2               -1.845941
3               -1.845941
4               -1.845941
…                    …
3998             1.389519
3999            -0.675975
4000            -0.157875
4001             0.277330
4002            -0.675975

[4003 rows x 17 columns]
```

[21]:
```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the scaler on the selected column
scaled_column = scaler.fit_transform(cars_price_df[['Year']])

# Replace the original column with the scaled column
cars_price_df['Year_MinMax'] = scaled_column
print(cars_price_df)
```

```
              Brand     Price  Mileage  EngineV  Year  Mileage_log  Price_log  \
0     Mercedes-Benz  222000.0        1      6.3  2016     0.000000  12.310433
1     Mercedes-Benz  177000.0        1      5.5  2016     0.000000  12.083905
2     Mercedes-Benz  177777.0        1      5.5  2016     0.000000  12.088285
3     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
4     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
…               …        …       …        …     …          …          …
3998         Toyota     600.0       10      1.5  1979     2.302585   6.396930
3999  Mercedes-Benz    2990.0      300      2.8  1979     5.703782   8.003029
4000  Mercedes-Benz    2300.0      261      2.3  1978     5.564520   7.740664
4001  Mercedes-Benz    5500.0      440      2.0  1978     6.086775   8.612503
4002  Mercedes-Benz   34999.0      150      2.8  1969     5.010635  10.463075

      EngineV_log  Mileage_sqrt  Price_sqrt  EngineV_sqrt  Mileage_reci  \
0        1.840550      1.000000  471.168760      2.509980      1.000000
```

```
1       1.704748      1.000000  420.713679      2.345208      1.000000
2       1.704748      1.000000  421.636099      2.345208      1.000000
3       1.704748      1.000000  447.212477      2.345208      1.000000
4       1.704748      1.000000  447.212477      2.345208      1.000000
…            …             …         …            …            …
3998    0.405465      3.162278   24.494897      1.224745      0.100000
3999    1.029619     17.320508   54.680892      1.673320      0.003333
4000    0.832909     16.155494   47.958315      1.516575      0.003831
4001    0.693147     20.976177   74.161985      1.414214      0.002273
4002    1.029619     12.247449  187.080197      1.673320      0.006667

      Price_reci  EngineV_reci  Price_log_scaler  Mileage_sqrt_scaler  \
0       0.000005      0.158730          3.104998            -2.301687
1       0.000006      0.181818          2.862201            -2.301687
2       0.000006      0.181818          2.866896            -2.301687
3       0.000005      0.181818          2.993137            -2.301687
4       0.000005      0.181818          2.993137            -2.301687
…          …             …                …                    …
3998    0.001667      0.666667         -3.233208            -1.844164
3999    0.000334      0.357143         -1.511761             1.151619
4000    0.000435      0.434783         -1.792968             0.905111
4001    0.000182      0.500000         -0.858514             1.925134
4002    0.000029      0.357143          1.124964             0.078195

      EngineV_reci_scaler  Year_MinMax
0               -2.000010     1.000000
1               -1.845941     1.000000
2               -1.845941     1.000000
3               -1.845941     1.000000
4               -1.845941     1.000000
…                    …           …
3998             1.389519     0.212766
3999            -0.675975     0.212766
4000            -0.157875     0.191489
4001             0.277330     0.191489
4002            -0.675975     0.000000

[4003 rows x 18 columns]
```

[22]:
```
cars_price_df[['Price_log_scaler',
 ↪'Mileage_sqrt_scaler','EngineV_reci_scaler','Year_MinMax']].describe()
```

[22]:

	Price_log_scaler	Mileage_sqrt_scaler	EngineV_reci_scaler	Year_MinMax
count	4.003000e+03	4.003000e+03	4.003000e+03	4003.000000
mean	5.680082e-17	-5.680082e-17	-1.278018e-16	0.795653
std	1.000125e+00	1.000125e+00	1.000125e+00	0.142453
min	-3.233208e+00	-2.301687e+00	-2.032602e+00	0.000000

```
25%       -6.000318e-01      -5.059319e-01      -8.348595e-01    0.723404
50%       -6.794118e-02       1.464036e-01      -2.599454e-02    0.829787
75%        6.224600e-01       6.956878e-01       6.480596e-01    0.914894
max        3.427728e+00       4.110631e+00       8.062655e+00    1.000000
```

15 12. Convert Cat - to - Numerical Columns

[23]:
```python
cat_onehot_df = pd.get_dummies(cars_price_df['Brand'], prefix='Category',
    drop_first=False)

# Concatenate the dummy variables with the original DataFrame
df = pd.concat([cars_price_df, cat_onehot_df], axis=1)
df
```

[23]:
```
              Brand     Price  Mileage  EngineV  Year  Mileage_log  Price_log  \
0     Mercedes-Benz  222000.0        1      6.3  2016     0.000000  12.310433
1     Mercedes-Benz  177000.0        1      5.5  2016     0.000000  12.083905
2     Mercedes-Benz  177777.0        1      5.5  2016     0.000000  12.088285
3     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
4     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
…               …         …       …        …    …          …          …
3998         Toyota     600.0       10      1.5  1979     2.302585   6.396930
3999  Mercedes-Benz    2990.0      300      2.8  1979     5.703782   8.003029
4000  Mercedes-Benz    2300.0      261      2.3  1978     5.564520   7.740664
4001  Mercedes-Benz    5500.0      440      2.0  1978     6.086775   8.612503
4002  Mercedes-Benz   34999.0      150      2.8  1969     5.010635  10.463075

      EngineV_log  Mileage_sqrt  Price_sqrt  …  Mileage_sqrt_scaler  \
0        1.840550      1.000000  471.168760  …            -2.301687
1        1.704748      1.000000  420.713679  …            -2.301687
2        1.704748      1.000000  421.636099  …            -2.301687
3        1.704748      1.000000  447.212477  …            -2.301687
4        1.704748      1.000000  447.212477  …            -2.301687
…             …           …         …   …             …
3998     0.405465      3.162278   24.494897  …            -1.844164
3999     1.029619     17.320508   54.680892  …             1.151619
4000     0.832909     16.155494   47.958315  …             0.905111
4001     0.693147     20.976177   74.161985  …             1.925134
4002     1.029619     12.247449  187.080197  …             0.078195

      EngineV_reci_scaler  Year_MinMax  Category_Audi  Category_BMW  \
0               -2.000010     1.000000          False         False
1               -1.845941     1.000000          False         False
2               -1.845941     1.000000          False         False
3               -1.845941     1.000000          False         False
4               -1.845941     1.000000          False         False
```

```
…                …          …          …          …
3998          1.389519   0.212766      False        False
3999         -0.675975   0.212766      False        False
4000         -0.157875   0.191489      False        False
4001          0.277330   0.191489      False        False
4002         -0.675975   0.000000      False        False

      Category_Mercedes-Benz  Category_Mitsubishi  Category_Renault  \
0                       True                False             False
1                       True                False             False
2                       True                False             False
3                       True                False             False
4                       True                False             False
…                        …                    …                 …
3998                   False                False             False
3999                    True                False             False
4000                    True                False             False
4001                    True                False             False
4002                    True                False             False

      Category_Toyota  Category_Volkswagen
0               False                False
1               False                False
2               False                False
3               False                False
4               False                False
…                 …                    …
3998             True                False
3999            False                False
4000            False                False
4001            False                False
4002            False                False

[4003 rows x 25 columns]
```

15.1 VIF

[24]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
# VIF dataframe
# the independent variables set
X = cars_price_df[['Mileage_sqrt_scaler','EngineV_reci_scaler','Year_MinMax']]

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
```

```
                          for i in range(len(X.columns))]

print(vif_data)
```

```
             feature       VIF
0  Mileage_sqrt_scaler  1.013955
1  EngineV_reci_scaler  1.001218
2          Year_MinMax  1.012732
```

[25]:
```
df.columns
```

[25]:
```
Index(['Brand', 'Price', 'Mileage', 'EngineV', 'Year', 'Mileage_log',
       'Price_log', 'EngineV_log', 'Mileage_sqrt', 'Price_sqrt',
       'EngineV_sqrt', 'Mileage_reci', 'Price_reci', 'EngineV_reci',
       'Price_log_scaler', 'Mileage_sqrt_scaler', 'EngineV_reci_scaler',
       'Year_MinMax', 'Category_Audi', 'Category_BMW',
       'Category_Mercedes-Benz', 'Category_Mitsubishi', 'Category_Renault',
       'Category_Toyota', 'Category_Volkswagen'],
      dtype='object')
```

[26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4003 entries, 0 to 4002
Data columns (total 25 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Brand                   4003 non-null   object
 1   Price                   4003 non-null   float64
 2   Mileage                 4003 non-null   int64
 3   EngineV                 4003 non-null   float64
 4   Year                    4003 non-null   int64
 5   Mileage_log             4003 non-null   float64
 6   Price_log               4003 non-null   float64
 7   EngineV_log             4003 non-null   float64
 8   Mileage_sqrt            4003 non-null   float64
 9   Price_sqrt              4003 non-null   float64
 10  EngineV_sqrt            4003 non-null   float64
 11  Mileage_reci            4003 non-null   float64
 12  Price_reci              4003 non-null   float64
 13  EngineV_reci            4003 non-null   float64
 14  Price_log_scaler        4003 non-null   float64
 15  Mileage_sqrt_scaler     4003 non-null   float64
 16  EngineV_reci_scaler     4003 non-null   float64
 17  Year_MinMax             4003 non-null   float64
 18  Category_Audi           4003 non-null   bool
 19  Category_BMW            4003 non-null   bool
 20  Category_Mercedes-Benz  4003 non-null   bool
```

```
 21   Category_Mitsubishi     4003 non-null   bool
 22   Category_Renault        4003 non-null   bool
 23   Category_Toyota         4003 non-null   bool
 24   Category_Volkswagen     4003 non-null   bool
dtypes: bool(7), float64(15), int64(2), object(1)
memory usage: 590.4+ KB
```

16 13. Inferential statistics test

[27]:
```python
for i in ['Category_Audi','Category_BMW','Category_Mercedes-Benz',␣
↪'Category_Mitsubishi', 'Category_Renault',
        'Category_Toyota', 'Category_Volkswagen']:
    df[i] = df[i].astype(int)
```

[28]:
```python
import statsmodels.api as sm
X = sm.add_constant(df[␣
↪['Mileage_sqrt_scaler','EngineV_reci_scaler','Year_MinMax','Category_Audi',␣
↪'Category_BMW',       'Category_Mercedes-Benz', 'Category_Mitsubishi',␣
↪'Category_Renault',      'Category_Toyota','Category_Volkswagen']])

# Fit a linear regression model
model = sm.OLS(df['Price_log_scaler'], X)
results = model.fit()

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

# Calculate the R2 score
print(f"R2 score: {results.rsquared}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       Price_log_scaler   R-squared:                       0.802
Model:                            OLS   Adj. R-squared:                  0.801
Method:                 Least Squares   F-statistic:                     1792.
Date:                Sat, 01 Jun 2024   Prob (F-statistic):               0.00
Time:                        16:37:36   Log-Likelihood:                -2443.4
No. Observations:                4003   AIC:                             4907.
Df Residuals:                    3993   BIC:                             4970.
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
==========
                             coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----------
```

```
const                      -3.1326      0.046    -68.103      0.000      -3.223
-3.042
Mileage_sqrt_scaler        -0.2231      0.009    -24.054      0.000      -0.241
-0.205
EngineV_reci_scaler        -0.3421      0.008    -40.651      0.000      -0.359
-0.326
Year_MinMax                 4.4511      0.065     68.088      0.000       4.323
4.579
Category_Audi              -0.3800      0.021    -18.270      0.000      -0.421
-0.339
Category_BMW               -0.2507      0.018    -14.088      0.000      -0.286
-0.216
Category_Mercedes-Benz     -0.1987      0.017    -11.815      0.000      -0.232
-0.166
Category_Mitsubishi        -0.6544      0.024    -27.500      0.000      -0.701
-0.608
Category_Renault           -0.7840      0.024    -33.161      0.000      -0.830
-0.738
Category_Toyota            -0.3957      0.020    -19.948      0.000      -0.435
-0.357
Category_Volkswagen        -0.4691      0.017    -27.776      0.000      -0.502
-0.436
==============================================================================
Omnibus:                      375.806   Durbin-Watson:                   1.560
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2464.651
Skew:                          -0.161   Prob(JB):                         0.00
Kurtosis:                       6.830   Cond. No.                     3.84e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 4.9e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
R2 score: 0.8015206758411855
```

Observations - Durbin-Watson (1.560) - Since this is within the range of 1.5 and 2.5, we would consider autocorrelation not to be problematic in this regression model.

17 14. ML - Linear Regression Model

[29]:
```
df
```

[29]:

	Brand	Price	Mileage	EngineV	Year	Mileage_log	Price_log	\
0	Mercedes-Benz	222000.0	1	6.3	2016	0.000000	12.310433	
1	Mercedes-Benz	177000.0	1	5.5	2016	0.000000	12.083905	
2	Mercedes-Benz	177777.0	1	5.5	2016	0.000000	12.088285	

```
3     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
4     Mercedes-Benz  199999.0        1      5.5  2016     0.000000  12.206068
…               …         …        …        …    …         …         …
3998         Toyota     600.0       10      1.5  1979     2.302585   6.396930
3999  Mercedes-Benz    2990.0      300      2.8  1979     5.703782   8.003029
4000  Mercedes-Benz    2300.0      261      2.3  1978     5.564520   7.740664
4001  Mercedes-Benz    5500.0      440      2.0  1978     6.086775   8.612503
4002  Mercedes-Benz   34999.0      150      2.8  1969     5.010635  10.463075

      EngineV_log  Mileage_sqrt  Price_sqrt  …  Mileage_sqrt_scaler  \
0        1.840550      1.000000  471.168760  …            -2.301687
1        1.704748      1.000000  420.713679  …            -2.301687
2        1.704748      1.000000  421.636099  …            -2.301687
3        1.704748      1.000000  447.212477  …            -2.301687
4        1.704748      1.000000  447.212477  …            -2.301687
…             …             …          …  …                  …
3998     0.405465      3.162278   24.494897  …            -1.844164
3999     1.029619     17.320508   54.680892  …             1.151619
4000     0.832909     16.155494   47.958315  …             0.905111
4001     0.693147     20.976177   74.161985  …             1.925134
4002     1.029619     12.247449  187.080197  …             0.078195

      EngineV_reci_scaler  Year_MinMax  Category_Audi  Category_BMW  \
0               -2.000010     1.000000              0             0
1               -1.845941     1.000000              0             0
2               -1.845941     1.000000              0             0
3               -1.845941     1.000000              0             0
4               -1.845941     1.000000              0             0
…                     …           …             …             …
3998             1.389519     0.212766              0             0
3999            -0.675975     0.212766              0             0
4000            -0.157875     0.191489              0             0
4001             0.277330     0.191489              0             0
4002            -0.675975     0.000000              0             0

      Category_Mercedes-Benz  Category_Mitsubishi  Category_Renault  \
0                          1                    0                 0
1                          1                    0                 0
2                          1                    0                 0
3                          1                    0                 0
4                          1                    0                 0
…                        …                    …                 …
3998                       0                    0                 0
3999                       1                    0                 0
4000                       1                    0                 0
4001                       1                    0                 0
4002                       1                    0                 0
```

```
      Category_Toyota  Category_Volkswagen
0                   0                    0
1                   0                    0
2                   0                    0
3                   0                    0
4                   0                    0
…                 …                  …
3998                1                    0
3999                0                    0
4000                0                    0
4001                0                    0
4002                0                    0

[4003 rows x 25 columns]
```

[30]:
```
df.columns
```

[30]:
```
Index(['Brand', 'Price', 'Mileage', 'EngineV', 'Year', 'Mileage_log',
       'Price_log', 'EngineV_log', 'Mileage_sqrt', 'Price_sqrt',
       'EngineV_sqrt', 'Mileage_reci', 'Price_reci', 'EngineV_reci',
       'Price_log_scaler', 'Mileage_sqrt_scaler', 'EngineV_reci_scaler',
       'Year_MinMax', 'Category_Audi', 'Category_BMW',
       'Category_Mercedes-Benz', 'Category_Mitsubishi', 'Category_Renault',
       'Category_Toyota', 'Category_Volkswagen'],
      dtype='object')
```

[31]:
```
df_final=df[['Price_log_scaler', 'Mileage_sqrt_scaler',␣
 ↪'EngineV_reci_scaler','Year_MinMax','Category_Audi',
        'Category_BMW', 'Category_Mercedes-Benz',
       'Category_Mitsubishi', 'Category_Renault', 'Category_Toyota',
       'Category_Volkswagen']].copy()
```

[32]:
```
X=df_final.drop(['Price_log_scaler'],axis='columns')
Y=df_final['Price_log_scaler']
```

[33]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.30)
print("train data length:",len(X_train))
print("test data length:",len(X_test))
X.columns
```

```
train data length: 2802
test data length: 1201
```

[33]:
```
Index(['Mileage_sqrt_scaler', 'EngineV_reci_scaler', 'Year_MinMax',
       'Category_Audi', 'Category_BMW', 'Category_Mercedes-Benz',
       'Category_Mitsubishi', 'Category_Renault', 'Category_Toyota',
       'Category_Volkswagen'],
```

```
dtype='object')
```

17.1 14.1 Linear Regression

[34]:
```python
def adjusted_r_squared(y_true, y_pred, n_samples, n_features):
    """
    Calculate the adjusted R-squared score.

    Parameters:
    - y_true: array-like, true target values
    - y_pred: array-like, predicted target values
    - n_samples: int, number of samples (observations)
    - n_features: int, number of features (predictors)

    Returns:
    - adjusted R-squared score
    """
    from sklearn.metrics import r2_score

    r_squared = r2_score(y_true, y_pred)
    adjusted_r_squared = 1 - (1 - r_squared) * ((n_samples - 1) / (n_samples -␣
 ↪n_features - 1))

    return adjusted_r_squared
```

[35]:
```python
model_results = {}
def regression_matrix(model ,X_train,X_test,y_test, model_name):
    print("Model Name ",model_name)
    y_pred = model.predict(X_test)
    train_r2_score=round(model.score(X_train,y_train),3)
    print("train R2 Score:",train_r2_score)
    test_r2_score=round(model.score(X_test,y_test),3)
    print("Test R2 Score:",test_r2_score)
    print("Test R2 score:",r2_score(y_test,y_pred))

    mse = round(mean_squared_error(y_test,y_pred),3)
    print("MSE:",mse)
    #rmse=round(root_mean_squared_error(y_test,y_pred),3)
    rmse=round(np.sqrt(mse),3)
    print("RMSE:",rmse)
    adj_r2_score=round(adjusted_r_squared(y_test,y_pred,len(y_test),len(X_train.
 ↪columns)),3)
    print("Adj-R Score",adj_r2_score)

    if abs(train_r2_score - test_r2_score) > .10:
        print("model :" , model_name ,"is overfitting")
    if train_r2_score < 0.50:
```

```
        print("model :" , model_name ,"is underfitting")

    ␣
  ↪model_results[model_name]=[train_r2_score,test_r2_score,adj_r2_score,mse,rmse]
```

[36]:
```
lr = LinearRegression()
lr.fit(X_train,y_train)


print("columns:",X_train.columns)
print('Coefficients: ', lr.coef_)
print('Intercept:',lr.intercept_)

regression_matrix(lr ,X_train,X_test,y_test, "LinearReg")
```

```
columns: Index(['Mileage_sqrt_scaler', 'EngineV_reci_scaler', 'Year_MinMax',
       'Category_Audi', 'Category_BMW', 'Category_Mercedes-Benz',
       'Category_Mitsubishi', 'Category_Renault', 'Category_Toyota',
       'Category_Volkswagen'],
      dtype='object')
Coefficients:  [-2.26901611e-01 -3.34514321e-01  4.54020890e+00 -1.46260246e+13
 -1.46260246e+13 -1.46260246e+13 -1.46260246e+13 -1.46260246e+13
 -1.46260246e+13 -1.46260246e+13]
Intercept: 14626024585324.809
Model Name  LinearReg
train R2 Score: 0.807
Test R2 Score: 0.785
Test R2 score: 0.7846599620798789
MSE: 0.197
RMSE: 0.444
Adj-R Score 0.783
```

17.1.1 Lasso Regression - L1

[37]:
```
lasso_reg = Lasso(alpha=0.1)  # Regularization strength (alpha) is set to 0.1
lasso_reg.fit(X_train,y_train)


print("columns:",X_train.columns)
print('Coefficients: ', lasso_reg.coef_)
print('Intercept:',lasso_reg.intercept_)

regression_matrix(lasso_reg ,X_train,X_test,y_test, "Lasso")
```

```
columns: Index(['Mileage_sqrt_scaler', 'EngineV_reci_scaler', 'Year_MinMax',
       'Category_Audi', 'Category_BMW', 'Category_Mercedes-Benz',
       'Category_Mitsubishi', 'Category_Renault', 'Category_Toyota',
```

```
       'Category_Volkswagen'],
      dtype='object')
Coefficients:  [-0.52324899 -0.33562024  0.          -0.          0.          0.
 -0.         -0.          0.         -0.        ]
Intercept: -0.014651228818764964
Model Name  Lasso
train R2 Score: 0.535
Test R2 Score: 0.546
Test R2 score: 0.5464730454365776
MSE: 0.416
RMSE: 0.645
Adj-R Score 0.543
```

17.1.2 Ridge Regression -L2

[38]:
```python
ridge_reg = Ridge(alpha=0.1)  # Regularization strength (alpha) is set to 0.1
ridge_reg.fit(X_train,y_train)


print("columns:",X_train.columns)
print('Coefficients: ', ridge_reg.coef_)
print('Intercept:',ridge_reg.intercept_)

regression_matrix(ridge_reg ,X_train,X_test,y_test, "Ridge")
```

```
columns: Index(['Mileage_sqrt_scaler', 'EngineV_reci_scaler', 'Year_MinMax',
       'Category_Audi', 'Category_BMW', 'Category_Mercedes-Benz',
       'Category_Mitsubishi', 'Category_Renault', 'Category_Toyota',
       'Category_Volkswagen'],
      dtype='object')
Coefficients:  [-0.22962844 -0.33578971  4.50788691  0.06821927  0.2005005
0.2554526
 -0.20342715 -0.34470256  0.04436584 -0.0204085 ]
Intercept: -3.6321460198717666
Model Name  Ridge
train R2 Score: 0.807
Test R2 Score: 0.785
Test R2 score: 0.784952991018684
MSE: 0.197
RMSE: 0.444
Adj-R Score 0.783
```

17.1.3 Elastic Net

[39]:
```python
elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)  # l1_ratio controls the␣
 ↪balance between L1 and L2 penalties
elastic_net.fit(X_train,y_train)
```

```python
print("columns:",X_train.columns)
print('Coefficients: ', elastic_net.coef_)
print('Intercept:',elastic_net.intercept_)

regression_matrix(elastic_net ,X_train,X_test,y_test, "ElasticNet")
```

```
columns: Index(['Mileage_sqrt_scaler', 'EngineV_reci_scaler', 'Year_MinMax',
       'Category_Audi', 'Category_BMW', 'Category_Mercedes-Benz',
       'Category_Mitsubishi', 'Category_Renault', 'Category_Toyota',
       'Category_Volkswagen'],
      dtype='object')
Coefficients:  [-0.53140875 -0.36702749  0.17807115 -0.          0.          0.
 -0.         -0.          0.         -0.        ]
Intercept: -0.15596468992061246
Model Name  ElasticNet
train R2 Score: 0.562
Test R2 Score: 0.572
Test R2 score: 0.5723694410669489
MSE: 0.392
RMSE: 0.626
Adj-R Score 0.569
```

17.2 14.2 Decision Tree Regression

[40]:
```python
# Create and fit the model
model = DecisionTreeRegressor()
model.fit(X_train,y_train)
print("Model - Decision Tree Regression")

regression_matrix(model ,X_train,X_test,y_test, "DT")
```

```
Model - Decision Tree Regression
Model Name  DT
train R2 Score: 0.996
Test R2 Score: 0.765
Test R2 score: 0.765061007871335
MSE: 0.215
RMSE: 0.464
Adj-R Score 0.763
model : DT is overfitting
```

17.3 14.3 Random Forest Regression

[41]:
```python
# Create and fit the model
model = RandomForestRegressor()
model.fit(X_train, y_train)
print("Model - Random Forest Regression")
```

```
regression_matrix(model ,X_train,X_test,y_test, "RandomForest")
```

```
Model - Random Forest Regression
Model Name  RandomForest
train R2 Score: 0.978
Test R2 Score: 0.855
Test R2 score: 0.8550559993534799
MSE: 0.133
RMSE: 0.365
Adj-R Score 0.854
model : RandomForest is overfitting
```

17.4 14.4 Support Vector Regression (SVR)

[42]:
```
# Create and fit the model
model = SVR(kernel='linear')
model.fit(X_train, y_train)
print("Model - Support Vector Regression ")

regression_matrix(model ,X_train,X_test,y_test, "SVR")
```

```
Model - Support Vector Regression
Model Name  SVR
train R2 Score: 0.803
Test R2 Score: 0.784
Test R2 score: 0.7840000160244213
MSE: 0.198
RMSE: 0.445
Adj-R Score 0.782
```

17.5 14.5 AdaBoost Regression

[43]:
```
# Create and fit the model
ada_boost = AdaBoostRegressor()
ada_boost.fit(X_train, y_train)
print("Model - AdaBoost Regression ")

regression_matrix(ada_boost ,X_train,X_test,y_test, "AdaBoost")
```

```
Model - AdaBoost Regression
Model Name  AdaBoost
train R2 Score: 0.812
Test R2 Score: 0.774
Test R2 score: 0.7742414400136288
MSE: 0.207
RMSE: 0.455
Adj-R Score 0.772
```

17.6 14.6 Gradient Boosting Regression

[44]:
```
# Create and fit the model
gradient_boost = GradientBoostingRegressor()
gradient_boost.fit(X_train, y_train)
print("Model - Gradient Boosting Regression")

regression_matrix(gradient_boost ,X_train,X_test,y_test, "GradientBoost")
```

```
Model - Gradient Boosting Regression
Model Name  GradientBoost
train R2 Score: 0.891
Test R2 Score: 0.858
Test R2 score: 0.857631467748983
MSE: 0.13
RMSE: 0.361
Adj-R Score 0.856
```

17.7 14.7 XGBoost Regression

[45]:
```
# Create and fit the model
xg_boost = xgb.XGBRegressor()
xg_boost.fit(X_train, y_train)
print("Model-XGBoost Regression")

regression_matrix(xg_boost ,X_train,X_test,y_test, "XGB")
```

```
Model-XGBoost Regression
Model Name  XGB
train R2 Score: 0.971
Test R2 Score: 0.85
Test R2 score: 0.8499927128937329
MSE: 0.137
RMSE: 0.37
Adj-R Score 0.849
model : XGB is overfitting
```

17.8 18. Summary

[46]:
```
print("\n\n")
result=pd.DataFrame(model_results,index=["Train R2","Test R2" ,"Adj␣
↪R2","MSE","RMSE"])
print(result)
print("\n\n")

print(tabulate(result, headers='keys', tablefmt='pretty'))
```

```
          LinearReg  Lasso  Ridge  ElasticNet     DT  RandomForest    SVR  \
Train R2      0.807  0.535  0.807       0.562  0.996         0.978  0.803
Test R2       0.785  0.546  0.785       0.572  0.765         0.855  0.784
Adj R2        0.783  0.543  0.783       0.569  0.763         0.854  0.782
MSE           0.197  0.416  0.197       0.392  0.215         0.133  0.198
RMSE          0.444  0.645  0.444       0.626  0.464         0.365  0.445

          AdaBoost  GradientBoost    XGB
Train R2     0.812          0.891  0.971
Test R2      0.774          0.858  0.850
Adj R2       0.772          0.856  0.849
MSE          0.207          0.130  0.137
RMSE         0.455          0.361  0.370
```

```
+----------+-----------+-------+-------+------------+-------+--------------+----
---+----------+---------------+-------+
|          | LinearReg | Lasso | Ridge | ElasticNet |  DT   | RandomForest |
SVR  | AdaBoost | GradientBoost |  XGB  |
+----------+-----------+-------+-------+------------+-------+--------------+----
---+----------+---------------+-------+
| Train R2 |   0.807   | 0.535 | 0.807 |   0.562    | 0.996 |    0.978     |
0.803 |  0.812   |     0.891     | 0.971 |
| Test R2  |   0.785   | 0.546 | 0.785 |   0.572    | 0.765 |    0.855     |
0.784 |  0.774   |     0.858     | 0.85  |
|  Adj R2  |   0.783   | 0.543 | 0.783 |   0.569    | 0.763 |    0.854     |
0.782 |  0.772   |     0.856     | 0.849 |
|   MSE    |   0.197   | 0.416 | 0.197 |   0.392    | 0.215 |    0.133     |
0.198 |  0.207   |     0.13      | 0.137 |
|   RMSE   |   0.444   | 0.645 | 0.444 |   0.626    | 0.464 |    0.365     |
0.445 |  0.455   |     0.361     | 0.37  |
+----------+-----------+-------+-------+------------+-------+--------------+----
---+----------+---------------+-------+
```

[ ]: