v-milesstone-car-test-2

September 23, 2024

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
df=pd.read_csv("C:/kgisl class/MILESTONE - 2/car details v4.csv")
df
```

[2]:

	Make	Model	Price	Year	Kilometer	Fuel Type	Transmission	Location	Color	Owner	Seller Type	Engine	Max Power	Max Torque
0	Honda	Amaze 1.2 VX i-VTEC	505000	2017	87150	Petrol	Manual	Pune	Grey	First	Corporate	1198 cc	87 bhp @ 6000 rpm	109 Nm @ 4500 rpm
1	Maruti Suzuki	Swift DZire VDI	450000	2014	75000	Diesel	Manual	Ludhiana	White	Second	Individual	1248 cc	74 bhp @ 4000 rpm	190 Nm @ 2000 rpm
2	Hyundai	i10 Magna 1.2 Kappa2	220000	2011	67000	Petrol	Manual	Lucknow	Maroon	First	Individual	1197 cc	79 bhp @ 6000 rpm	112.7619 Nm @ 4000 rpm
3	Toyota	Glanza G	799000	2019	37500	Petrol	Manual	Mangalore	Red	First				
4	Toyota	Innova 2.4 VX 7 STR [2016-2020]	1950000	2018	69000	Diesel	Manual	Mumbai	Grey	First				
…	…	…	…	…	…	…	…	…	…	…				
2054	Mahindra	XUV500 W8 [2015-2017]	850000	2016	90300	Diesel	Manual	Surat	White	First				
2055	Hyundai	Eon D-Lite +	275000	2014	83000	Petrol	Manual	Ahmedabad	White	Second				
2056	Ford	Figo Duratec Petrol ZXI 1.2	240000	2013	73000	Petrol	Manual	Thane	Silver	First				
2057	BMW	5-Series 520d Luxury Line [2017-2019]	4290000	2018	60474	Diesel	Automatic	Coimbatore	White	First				
2058	Mahindra	Bolero Power Plus ZLX [2016-2019]	670000	2017	72000	Diesel	Manual	Guwahati	White	First				

```
3     Individual  1197 cc   82 bhp @ 6000 rpm       113 Nm @ 4200 rpm
4     Individual  2393 cc  148 bhp @ 3400 rpm       343 Nm @ 1400 rpm
…          …      …                 …                    …
2054  Individual  2179 cc  138 bhp @ 3750 rpm       330 Nm @ 1600 rpm
2055  Individual   814 cc   55 bhp @ 5500 rpm        75 Nm @ 4000 rpm
2056  Individual  1196 cc   70 bhp @ 6250 rpm       102 Nm @ 4000 rpm
2057  Individual  1995 cc  188 bhp @ 4000 rpm       400 Nm @ 1750 rpm
2058  Individual  1493 cc   70 bhp @ 3600 rpm       195 Nm @ 1400 rpm

     Drivetrain  Length   Width  Height  Seating Capacity  Fuel Tank Capacity
0           FWD  3990.0  1680.0  1505.0               5.0                35.0
1           FWD  3995.0  1695.0  1555.0               5.0                42.0
2           FWD  3585.0  1595.0  1550.0               5.0                35.0
3           FWD  3995.0  1745.0  1510.0               5.0                37.0
4           RWD  4735.0  1830.0  1795.0               7.0                55.0
…         …      …       …       …                  …                  …
2054        FWD  4585.0  1890.0  1785.0               7.0                70.0
2055        FWD  3495.0  1550.0  1500.0               5.0                32.0
2056        FWD  3795.0  1680.0  1427.0               5.0                45.0
2057        RWD  4936.0  1868.0  1479.0               5.0                65.0
2058        RWD  3995.0  1745.0  1880.0               7.0                 NaN

[2059 rows x 20 columns]
```

[3]:
```
df.describe()
```

[3]:
```
              Price         Year     Kilometer       Length        Width  \
count  2.059000e+03  2059.000000  2.059000e+03  1995.000000  1995.000000
mean   1.702992e+06  2016.425449  5.422471e+04  4280.860652  1767.991980
std    2.419881e+06     3.363564  5.736172e+04   442.458507   135.265825
min    4.900000e+04  1988.000000  0.000000e+00  3099.000000  1475.000000
25%    4.849990e+05  2014.000000  2.900000e+04  3985.000000  1695.000000
50%    8.250000e+05  2017.000000  5.000000e+04  4370.000000  1770.000000
75%    1.925000e+06  2019.000000  7.200000e+04  4629.000000  1831.500000
max    3.500000e+07  2022.000000  2.000000e+06  5569.000000  2220.000000

            Height  Seating Capacity  Fuel Tank Capacity
count  1995.000000       1995.000000         1946.000000
mean   1591.735338          5.306266           52.002210
std     136.073956          0.822170           15.110198
min    1165.000000          2.000000           15.000000
25%    1485.000000          5.000000           41.250000
50%    1545.000000          5.000000           50.000000
75%    1675.000000          5.000000           60.000000
max    1995.000000          8.000000          105.000000
```

[ ]:
```

```

[4]:
```
df=df.drop(columns=['Length','Width','Height','Fuel Tank Capacity','Location'],␣
↪axis=1)
df
```

[4]:

	Make	Model	Price	Year	\
0	Honda	Amaze 1.2 VX i-VTEC	505000	2017	
1	Maruti Suzuki	Swift DZire VDI	450000	2014	
2	Hyundai	i10 Magna 1.2 Kappa2	220000	2011	
3	Toyota	Glanza G	799000	2019	
4	Toyota	Innova 2.4 VX 7 STR [2016-2020]	1950000	2018	
…	…	…	…	…	
2054	Mahindra	XUV500 W8 [2015-2017]	850000	2016	
2055	Hyundai	Eon D-Lite +	275000	2014	
2056	Ford	Figo Duratec Petrol ZXI 1.2	240000	2013	
2057	BMW	5-Series 520d Luxury Line [2017-2019]	4290000	2018	
2058	Mahindra	Bolero Power Plus ZLX [2016-2019]	670000	2017	

	Kilometer	Fuel Type	Transmission	Color	Owner	Seller Type	Engine	\
0	87150	Petrol	Manual	Grey	First	Corporate	1198 cc	
1	75000	Diesel	Manual	White	Second	Individual	1248 cc	
2	67000	Petrol	Manual	Maroon	First	Individual	1197 cc	
3	37500	Petrol	Manual	Red	First	Individual	1197 cc	
4	69000	Diesel	Manual	Grey	First	Individual	2393 cc	
…	…	…	…	…	…	…		
2054	90300	Diesel	Manual	White	First	Individual	2179 cc	
2055	83000	Petrol	Manual	White	Second	Individual	814 cc	
2056	73000	Petrol	Manual	Silver	First	Individual	1196 cc	
2057	60474	Diesel	Automatic	White	First	Individual	1995 cc	
2058	72000	Diesel	Manual	White	First	Individual	1493 cc	

	Max Power	Max Torque	Drivetrain	Seating Capacity
0	87 bhp @ 6000 rpm	109 Nm @ 4500 rpm	FWD	5.0
1	74 bhp @ 4000 rpm	190 Nm @ 2000 rpm	FWD	5.0
2	79 bhp @ 6000 rpm	112.7619 Nm @ 4000 rpm	FWD	5.0
3	82 bhp @ 6000 rpm	113 Nm @ 4200 rpm	FWD	5.0
4	148 bhp @ 3400 rpm	343 Nm @ 1400 rpm	RWD	7.0
…	…	…	…	…
2054	138 bhp @ 3750 rpm	330 Nm @ 1600 rpm	FWD	7.0
2055	55 bhp @ 5500 rpm	75 Nm @ 4000 rpm	FWD	5.0
2056	70 bhp @ 6250 rpm	102 Nm @ 4000 rpm	FWD	5.0
2057	188 bhp @ 4000 rpm	400 Nm @ 1750 rpm	RWD	5.0
2058	70 bhp @ 3600 rpm	195 Nm @ 1400 rpm	RWD	7.0

[2059 rows x 15 columns]

[5]:
```
df.isnull().sum()
```

[5]:
```
Make                  0
Model                 0
Price                 0
Year                  0
Kilometer             0
Fuel Type             0
Transmission          0
Color                 0
Owner                 0
Seller Type           0
Engine               80
Max Power            80
Max Torque           80
Drivetrain          136
Seating Capacity     64
dtype: int64
```

[6]:
```
categorical=['Engine','Max Power','Max Torque','Drivetrain']
```

[7]:
```
for i in categorical:
    df[i].fillna(df[i].mode()[0], inplace=True)
```

[8]:
```
df['Seating Capacity']=df['Seating Capacity'].fillna(df['Seating Capacity'].
↪median())
```

[9]:
```
df.describe()
```

[9]:

	Price	Year	Kilometer	Seating Capacity
count	2.059000e+03	2059.000000	2.059000e+03	2059.000000
mean	1.702992e+06	2016.425449	5.422471e+04	5.296746
std	2.419881e+06	3.363564	5.736172e+04	0.811029
min	4.900000e+04	1988.000000	0.000000e+00	2.000000
25%	4.849990e+05	2014.000000	2.900000e+04	5.000000
50%	8.250000e+05	2017.000000	5.000000e+04	5.000000
75%	1.925000e+06	2019.000000	7.200000e+04	5.000000
max	3.500000e+07	2022.000000	2.000000e+06	8.000000

[10]:
```
df.isnull().sum()
```

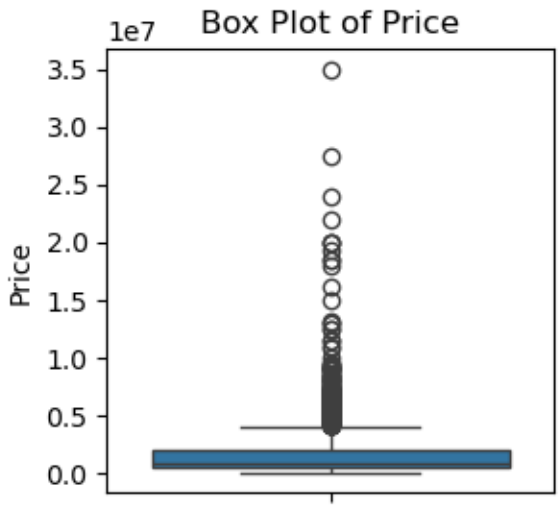
[10]:
```
Make                0
Model               0
Price               0
Year                0
Kilometer           0
Fuel Type           0
Transmission        0
Color               0
```

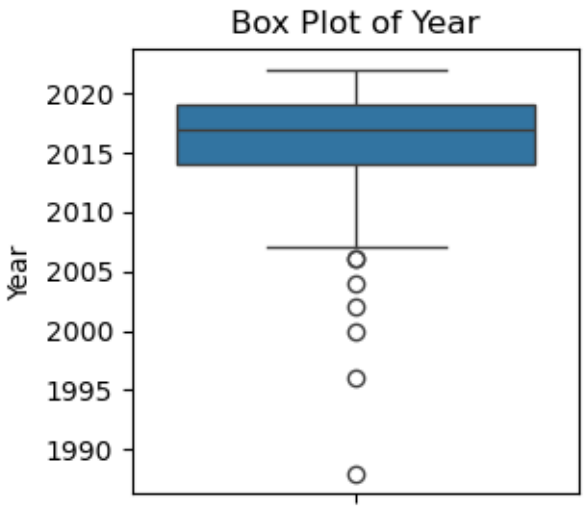
```
Owner                0
Seller Type          0
Engine               0
Max Power            0
Max Torque           0
Drivetrain           0
Seating Capacity     0
dtype: int64
```

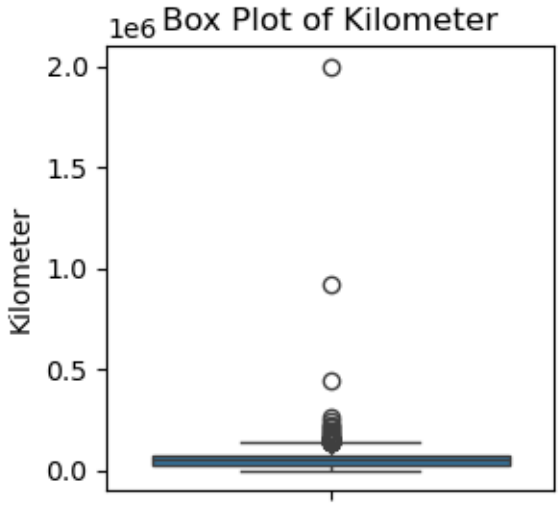
1 Outlier Detection

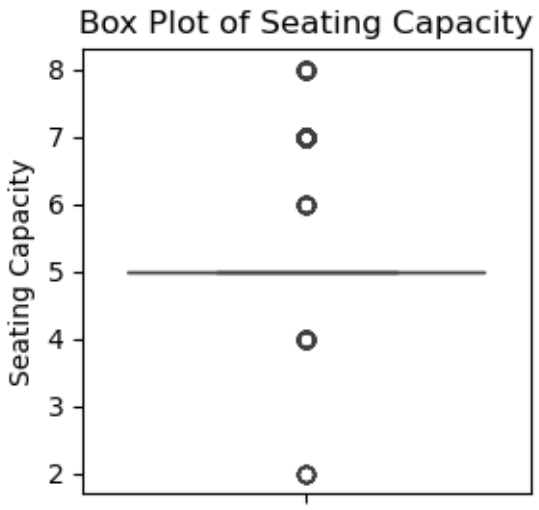
[11]:
```
numerical_columns=['Price','Year','Kilometer','Seating Capacity']
```

[12]:
```
import seaborn as sns
#outlier detection
for col in numerical_columns:
    plt.figure(figsize=(3, 3))
    sns.boxplot(y=df[col])
    plt.title(f'Box Plot of {col}')
    plt.show()
```




Box Plot of Year
Year
2020
2015
2010
2005
2000
1995
1990


Box Plot of Kilometer
1e6
Kilometer
2.0
1.5
1.0
0.5
0.0



[13]:
```
# Calculate Q1 (25th percentile) and Q3 (75th percentile) for numerical columns
Q1 = df[numerical_columns].quantile(0.25)
Q3 = df[numerical_columns].quantile(0.75)
IQR = Q3 - Q1
IQR
```

[13]:
```
Price              1440001.0
Year                     5.0
Kilometer            43000.0
Seating Capacity         0.0
dtype: float64
```

[14]:
```
outlier_mask = ((df[numerical_columns] < (Q1 - 1.5 * IQR)) |␣
 ↪(df[numerical_columns] > (Q3 + 1.5 * IQR)))

# Check which rows contain outliers
outliers = outlier_mask.any(axis=1)
print(f"Number of rows with outliers: {outliers.sum()}")
```

```
Number of rows with outliers: 563
```

[15]:
```
df_cleaned = df[~outliers]

# Check the shape of the cleaned data
print(f"Original data shape: {df.shape}")
print(f"Cleaned data shape: {df_cleaned.shape}")
```

```
Original data shape: (2059, 15)
Cleaned data shape: (1496, 15)
```

[16]:
```
df_cleaned
```

[16]:
```
                Make                          Model   Price  Year  Kilometer  \
0              Honda           Amaze 1.2 VX i-VTEC  505000  2017      87150
1      Maruti Suzuki               Swift DZire VDI  450000  2014      75000
2            Hyundai          i10 Magna 1.2 Kappa2  220000  2011      67000
3             Toyota                       Glanza G  799000  2019      37500
5      Maruti Suzuki                       Ciaz ZXi  675000  2017      73315
…                …                             …     …   …        …
2051   Maruti Suzuki             Vitara Brezza VXi  925000  2021      48000
2052         Hyundai           i20 Sportz 1.4 CRDI  409999  2014      68000
2053   Maruti Suzuki          Ritz Vxi (ABS) BS-IV  245000  2014      79000
2055         Hyundai                  Eon D-Lite +  275000  2014      83000
2056            Ford  Figo Duratec Petrol ZXI 1.2  240000  2013      73000

     Fuel Type Transmission   Color   Owner Seller Type   Engine  \
0       Petrol       Manual    Grey   First   Corporate  1198 cc
1       Diesel       Manual   White  Second  Individual  1248 cc
2       Petrol       Manual  Maroon   First  Individual  1197 cc
3       Petrol       Manual     Red   First  Individual  1197 cc
5       Petrol       Manual    Grey   First  Individual  1373 cc
…         …           …     …     …        …      …
2051    Petrol       Manual   White   First  Individual  1462 cc
2052    Diesel       Manual  Silver   First  Individual  1396 cc
2053    Petrol       Manual   White  Second  Individual  1197 cc
2055    Petrol       Manual   White  Second  Individual   814 cc
2056    Petrol       Manual  Silver   First  Individual  1196 cc

                Max Power              Max Torque Drivetrain  Seating Capacity
0       87 bhp @ 6000 rpm       109 Nm @ 4500 rpm        FWD               5.0
1       74 bhp @ 4000 rpm       190 Nm @ 2000 rpm        FWD               5.0
2       79 bhp @ 6000 rpm  112.7619 Nm @ 4000 rpm        FWD               5.0
3       82 bhp @ 6000 rpm       113 Nm @ 4200 rpm        FWD               5.0
5       91 bhp @ 6000 rpm       130 Nm @ 4000 rpm        FWD               5.0
…                     …                       …        …                 …
2051   103 bhp @ 6000 rpm       138 Nm @ 4400 rpm        FWD               5.0
2052              90@4000                 220@1750        FWD               5.0
2053    85 bhp @ 6000 rpm       113 Nm @ 4500 rpm        FWD               5.0
2055    55 bhp @ 5500 rpm        75 Nm @ 4000 rpm        FWD               5.0
2056    70 bhp @ 6250 rpm       102 Nm @ 4000 rpm        FWD               5.0

[1496 rows x 15 columns]
```

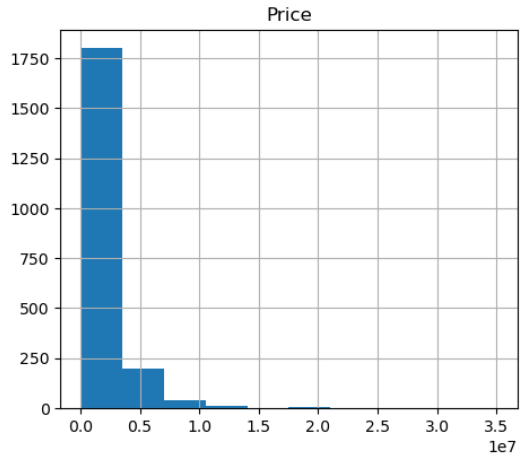
[17]:
```
df_cleaned.isnull().sum()
```

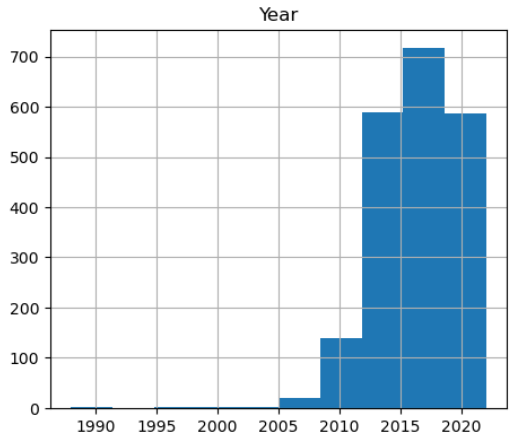
[17]:
```
Make              0
Model             0
```

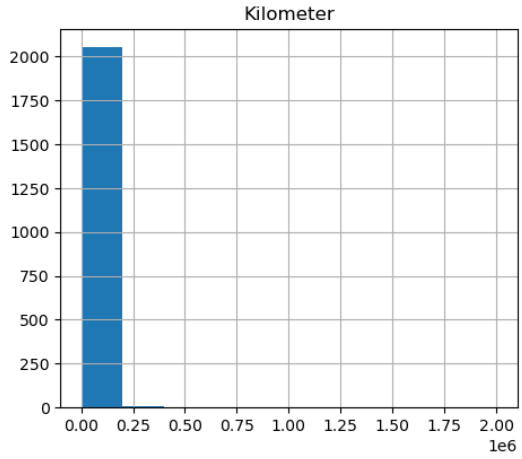
```
Price               0
Year                0
Kilometer           0
Fuel Type           0
Transmission        0
Color               0
Owner               0
Seller Type         0
Engine              0
Max Power           0
Max Torque          0
Drivetrain          0
Seating Capacity    0
dtype: int64
```

2 Univariate Analysis

[18]:
```
# Histograms
df[['Price', 'Year', 'Kilometer', 'Engine', 'Max Power', 'Max Torque']].
 ↪hist(figsize=(12, 10))
plt.show()
```







3 Bivariate analysis

[ ]:

[19]:
```
# Function to extract numeric values from strings
def extract_numeric(value):
    try:
        return float(value.split()[0].replace(',', ''))
    except:
        return None

# Apply the function to convert columns
df_cleaned['Engine'] = df_cleaned['Engine'].apply(extract_numeric)
df_cleaned['Max Power'] = df_cleaned['Max Power'].apply(extract_numeric)
df_cleaned['Max Torque'] = df_cleaned['Max Torque'].apply(extract_numeric)
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_8852\1821585746.py:9:
```

```
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Engine'] = df_cleaned['Engine'].apply(extract_numeric)
C:\Users\DELL\AppData\Local\Temp\ipykernel_8852\1821585746.py:10:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Max Power'] = df_cleaned['Max Power'].apply(extract_numeric)
C:\Users\DELL\AppData\Local\Temp\ipykernel_8852\1821585746.py:11:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Max Torque'] = df_cleaned['Max Torque'].apply(extract_numeric)
```

[20]:
```
df_cleaned
```

[20]:

	Make	Model	Price	Year	Kilometer	\
0	Honda	Amaze 1.2 VX i-VTEC	505000	2017	87150	
1	Maruti Suzuki	Swift DZire VDI	450000	2014	75000	
2	Hyundai	i10 Magna 1.2 Kappa2	220000	2011	67000	
3	Toyota	Glanza G	799000	2019	37500	
5	Maruti Suzuki	Ciaz ZXi	675000	2017	73315	
…	…	…	…	…	…	
2051	Maruti Suzuki	Vitara Brezza VXi	925000	2021	48000	
2052	Hyundai	i20 Sportz 1.4 CRDI	409999	2014	68000	
2053	Maruti Suzuki	Ritz Vxi (ABS) BS-IV	245000	2014	79000	
2055	Hyundai	Eon D-Lite +	275000	2014	83000	
2056	Ford	Figo Duratec Petrol ZXI 1.2	240000	2013	73000	

	Fuel Type	Transmission	Color	Owner	Seller Type	Engine	Max Power	\
0	Petrol	Manual	Grey	First	Corporate	1198.0	87.0	
1	Diesel	Manual	White	Second	Individual	1248.0	74.0	
2	Petrol	Manual	Maroon	First	Individual	1197.0	79.0	
3	Petrol	Manual	Red	First	Individual	1197.0	82.0	
5	Petrol	Manual	Grey	First	Individual	1373.0	91.0	
…	…	…	…	…	…	…		
2051	Petrol	Manual	White	First	Individual	1462.0	103.0	

```
2052    Diesel       Manual  Silver   First  Individual  1396.0        NaN
2053    Petrol       Manual   White  Second  Individual  1197.0       85.0
2055    Petrol       Manual   White  Second  Individual   814.0       55.0
2056    Petrol       Manual  Silver   First  Individual  1196.0       70.0

      Max Torque Drivetrain  Seating Capacity
0       109.0000        FWD               5.0
1       190.0000        FWD               5.0
2       112.7619        FWD               5.0
3       113.0000        FWD               5.0
5       130.0000        FWD               5.0
…          …        …                 …
2051    138.0000        FWD               5.0
2052         NaN        FWD               5.0
2053    113.0000        FWD               5.0
2055     75.0000        FWD               5.0
2056    102.0000        FWD               5.0

[1496 rows x 15 columns]
```

[21]:
```
df_cleaned.isnull().sum()
```

[21]:
```
Make                0
Model               0
Price               0
Year                0
Kilometer           0
Fuel Type           0
Transmission        0
Color               0
Owner               0
Seller Type         0
Engine              0
Max Power          99
Max Torque         99
Drivetrain          0
Seating Capacity    0
dtype: int64
```

[22]:
```
df_cleaned['Max Power'].fillna(df_cleaned['Max Power'].median(), inplace=True)
df_cleaned['Max Torque'].fillna(df_cleaned['Max Torque'].median(), inplace=True)
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_8852\2398879905.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Max Power'].fillna(df_cleaned['Max Power'].median(), inplace=True)
C:\Users\DELL\AppData\Local\Temp\ipykernel_8852\2398879905.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_cleaned['Max Torque'].fillna(df_cleaned['Max Torque'].median(),
inplace=True)
```

[23]:
```
df_cleaned = df_cleaned.dropna()
```

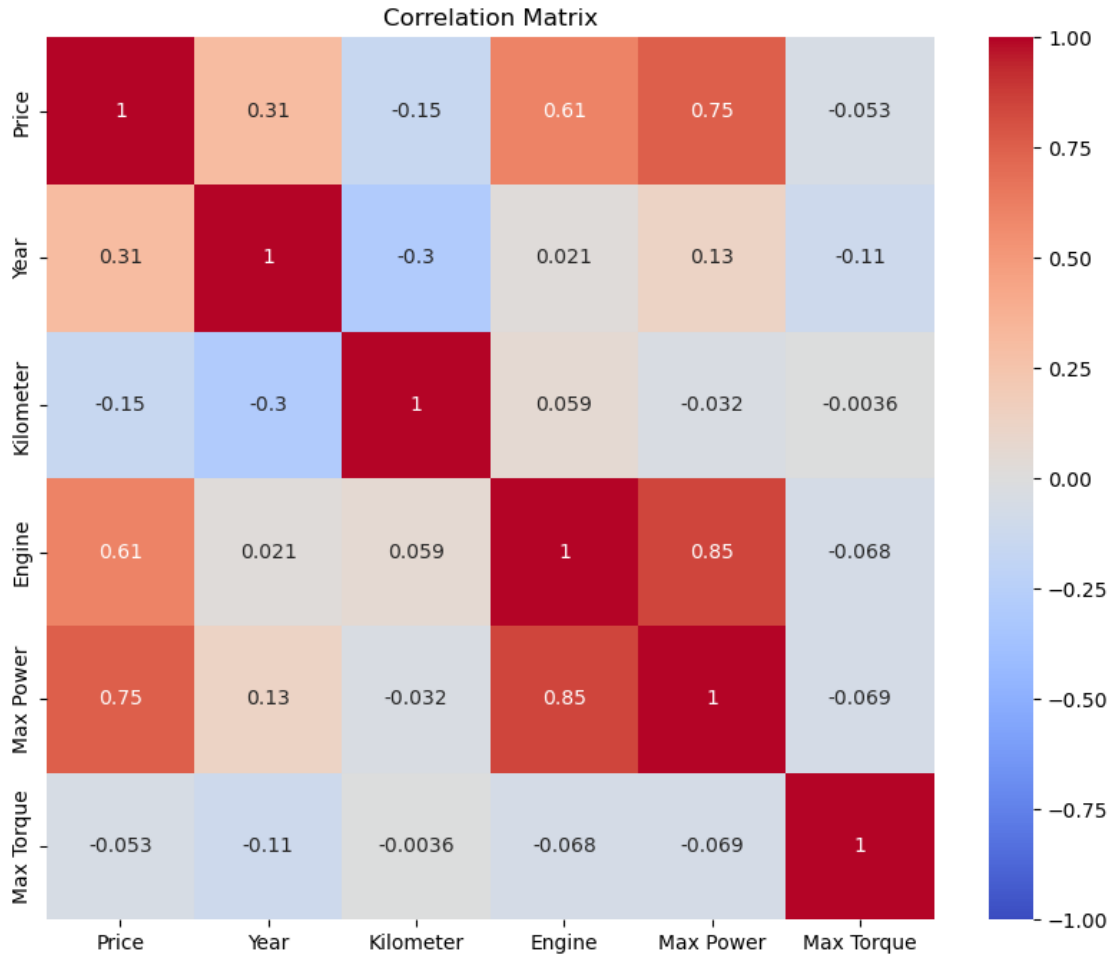
[24]:
```
# Remove non-numeric characters and convert to numeric
df['Engine'] = df['Engine'].replace('[^\d]', '', regex=True).astype(float)
df['Max Power'] = df['Max Power'].replace('[^\d]', '', regex=True).astype(float)
df['Max Torque'] = df['Max Torque'].replace('[^\d]', '', regex=True).
 ↪astype(float)
```

[25]:
```
corr_matrix = df[['Price', 'Year', 'Kilometer', 'Engine', 'Max Power', 'Max␣
 ↪Torque']].corr()
corr_matrix
```

[25]:

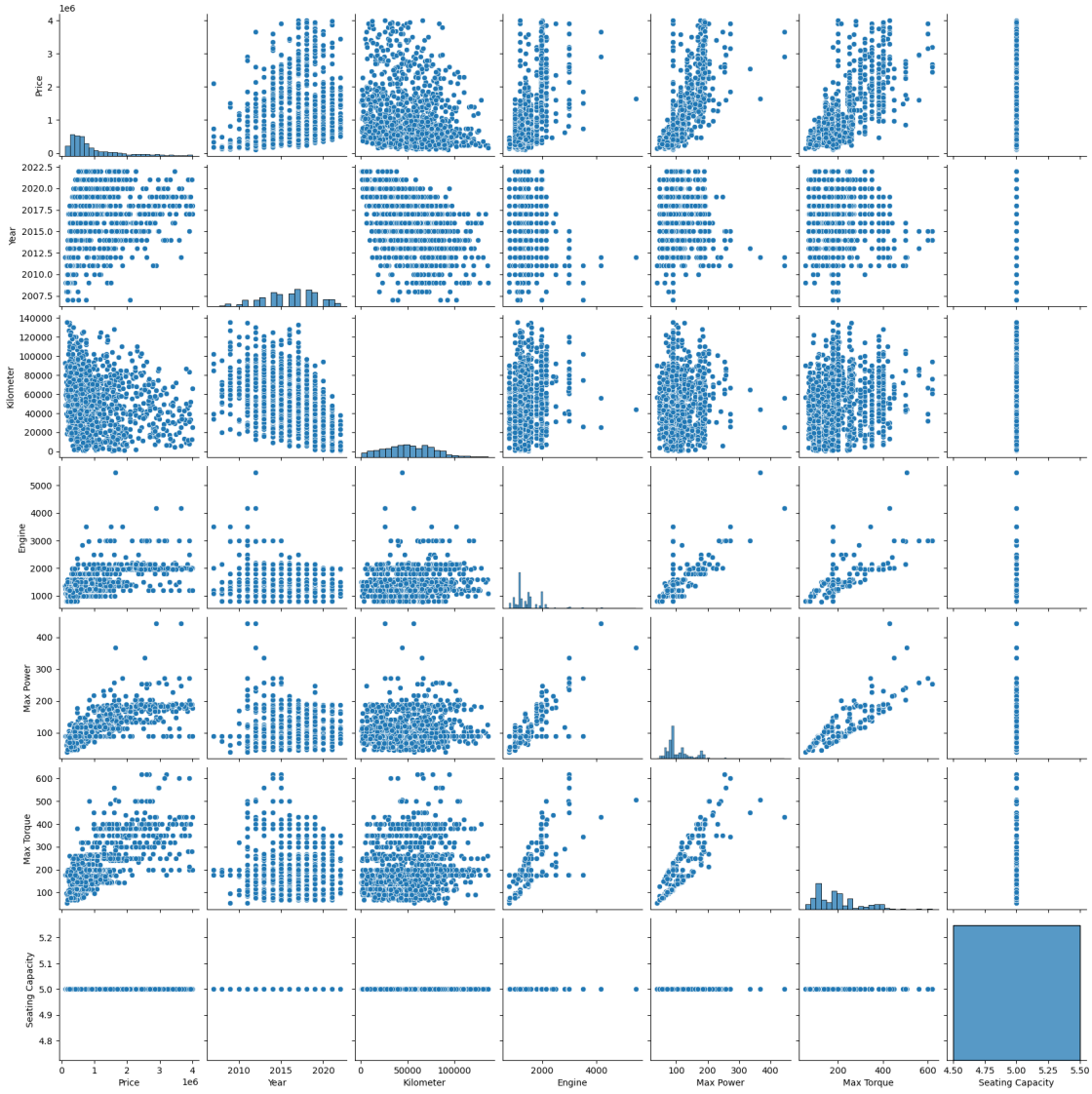
	Price	Year	Kilometer	Engine	Max Power	Max Torque
Price	1.000000	0.311400	-0.150825	0.608255	0.753338	-0.053103
Year	0.311400	1.000000	-0.296547	0.021308	0.126709	-0.112548
Kilometer	-0.150825	-0.296547	1.000000	0.058900	-0.032393	-0.003623
Engine	0.608255	0.021308	0.058900	1.000000	0.848248	-0.068478
Max Power	0.753338	0.126709	-0.032393	0.848248	1.000000	-0.068556
Max Torque	-0.053103	-0.112548	-0.003623	-0.068478	-0.068556	1.000000

[26]:
```
# Heatmap of correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



[27]:
```
import seaborn as sns
sns.pairplot(df_cleaned)
```

[27]: <seaborn.axisgrid.PairGrid at 0x22f2ad103d0>



[28]:
```
categorical_col=['Make','Model','Fuel␣
↪Type','Transmission','Color','Owner','Seller Type','Drivetrain']
```

[29]:
```
from sklearn.preprocessing import LabelEncoder
```

[30]:
```
le = LabelEncoder()
for col in categorical_col:
    df_cleaned[col] = le.fit_transform(df_cleaned[col])
```

[31]:
```
df_cleaned
```

[31]:

	Make	Model	Price	Year	Kilometer	Fuel Type	Transmission	Color	\
0	6	77	505000	2017	87150	5	1	7	

```
1        16    563  450000  2014      75000           2              1     15
2         7    727  220000  2011      67000           5              1      8
3        24    369  799000  2019      37500           5              1     13
5        16    166  675000  2017      73315           5              1      7
…    …    …    …    …       …         …           …    …
2051     16    656  925000  2021      48000           5              1     15
2052      7    746  409999  2014      68000           2              1     14
2053     16    506  245000  2014      79000           5              1     15
2055      7    330  275000  2014      83000           5              1     15
2056      5    355  240000  2013      73000           5              1     14

      Owner  Seller Type  Engine  Max Power  Max Torque  Drivetrain  \
0         1            1  1198.0       87.0    109.0000           1
1         3            2  1248.0       74.0    190.0000           1
2         1            2  1197.0       79.0    112.7619           1
3         1            2  1197.0       82.0    113.0000           1
5         1            2  1373.0       91.0    130.0000           1
…     …           …     …        …          …         …
2051      1            2  1462.0      103.0    138.0000           1
2052      1            2  1396.0       89.0    177.0000           1
2053      3            2  1197.0       85.0    113.0000           1
2055      3            2   814.0       55.0     75.0000           1
2056      1            2  1196.0       70.0    102.0000           1

      Seating Capacity
0                  5.0
1                  5.0
2                  5.0
3                  5.0
5                  5.0
…                  …
2051               5.0
2052               5.0
2053               5.0
2055               5.0
2056               5.0

[1496 rows x 15 columns]
```

[32]:
```
df_cleaned['Seating Capacity'].unique()
```

[32]:
```
array([5.])
```

[33]:
```
df_cleaned = df_cleaned.drop(columns=['Seating Capacity'])
df_cleaned
```

[33]:

	Make	Model	Price	Year	Kilometer	Fuel Type	Transmission	Color	Owner	Seller Type	Engine	Max Power	Max Torque	Drivetrain
0	6	77	505000	2017	87150	5	1	7	1	1	1198.0	87.0	109.0000	1
1	16	563	450000	2014	75000	2	1	15	3	2	1248.0	74.0	190.0000	1
2	7	727	220000	2011	67000	5	1	8	1	2	1197.0	79.0	112.7619	1
3	24	369	799000	2019	37500	5	1	13	1	2	1197.0	82.0	113.0000	1
5	16	166	675000	2017	73315	5	1	7	1	2	1373.0	91.0	130.0000	1
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
2051	16	656	925000	2021	48000	5	1	15	1	2	1462.0	103.0	138.0000	1
2052	7	746	409999	2014	68000	2	1	14	1	2	1396.0	89.0	177.0000	1
2053	16	506	245000	2014	79000	5	1	15	3	2	1197.0	85.0	113.0000	1
2055	7	330	275000	2014	83000	5	1	15	3	2	814.0	55.0	75.0000	1
2056	5	355	240000	2013	73000	5	1	14	1	2	1196.0	70.0	102.0000	1

[1496 rows x 14 columns]

[34]:
```
df_cleaned.describe()
```

[34]:

	Make	Model	Price	Year	Kilometer	Fuel Type	Transmission	Color	Owner	Seller Type
count	1496.000000	1496.000000	1.496000e+03	1496.000000	1496.000000	1496.000000	1496.000000	1496.000000	1496.000000	1496.000000
mean	11.758021	378.377005	9.807479e+05	2016.078877	51675.606952	3.610294	0.617647	10.680481	1.432487	1.970588
std	7.151696	222.313704	8.343264e+05	3.164465	26285.560975	1.586271	0.486125	5.144681	0.872156	0.184166
min	0.000000	0.000000	1.149990e+05	2007.000000	600.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	7.000000	181.000000	4.250000e+05	2014.000000	32000.000000	2.000000	0.000000	7.000000	1.000000	2.000000
50%	10.000000	381.000000	6.500000e+05	2016.000000	50733.500000	5.000000	1.000000	14.000000	1.000000	2.000000
75%	16.000000	577.000000	1.250000e+06	2018.000000	70000.000000					
max	26.000000	746.000000	4.000000e+06	2022.000000	135797.000000					

```
75%       5.000000      1.000000    15.000000     1.000000     2.000000
max       6.000000      1.000000    16.000000     5.000000     2.000000

            Engine    Max Power   Max Torque   Drivetrain
count  1496.000000  1496.000000  1496.000000  1496.000000
mean   1444.264706   108.309291   198.272425     1.018717
std     431.472406    43.667120   102.180693     0.338673
min     793.000000    39.000000    54.000000     0.000000
25%    1197.000000    82.000000   114.000000     1.000000
50%    1317.000000    89.000000   177.000000     1.000000
75%    1591.000000   126.000000   250.000000     1.000000
max    5461.000000   444.000000   619.000000     2.000000
```

4 Regression Analysis

[35]:
```
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Features to standardize
numerical_features = ['Price', 'Year', 'Kilometer', 'Engine', 'Max Power', 'Max␣
 ↪Torque']

# Initialize the StandardScaler
scaler = StandardScaler()

# Apply standardization
df_cleaned[numerical_features] = scaler.
 ↪fit_transform(df_cleaned[numerical_features])

# Verify the result
df_cleaned[numerical_features].describe()
```

[35]:
```
              Price          Year     Kilometer        Engine     Max Power  \
count  1.496000e+03  1.496000e+03  1.496000e+03  1.496000e+03  1.496000e+03
mean   8.074349e-17  2.263786e-14 -1.288334e-16  3.324732e-17 -7.599388e-17
std    1.000334e+00  1.000334e+00  1.000334e+00  1.000334e+00  1.000334e+00
min   -1.038009e+00 -2.869968e+00 -1.943755e+00 -1.509905e+00 -1.587750e+00
25%   -6.663265e-01 -6.571638e-01 -7.487832e-01 -5.732635e-01 -6.026981e-01
50%   -3.965577e-01 -2.493419e-02 -3.585322e-02 -2.950531e-01 -4.423408e-01
75%    3.228258e-01  6.072954e-01  6.973608e-01  3.401941e-01  4.052620e-01
max    3.620000e+00  1.871755e+00  3.201359e+00  9.312481e+00  7.690064e+00

         Max Torque
count  1.496000e+03
mean   2.683534e-16
std    1.000334e+00
```

```
min   -1.412406e+00
25%   -8.250150e-01
50%   -2.082540e-01
75%    5.064056e-01
max    4.118863e+00
```

[36]:
```
df_cleaned
```

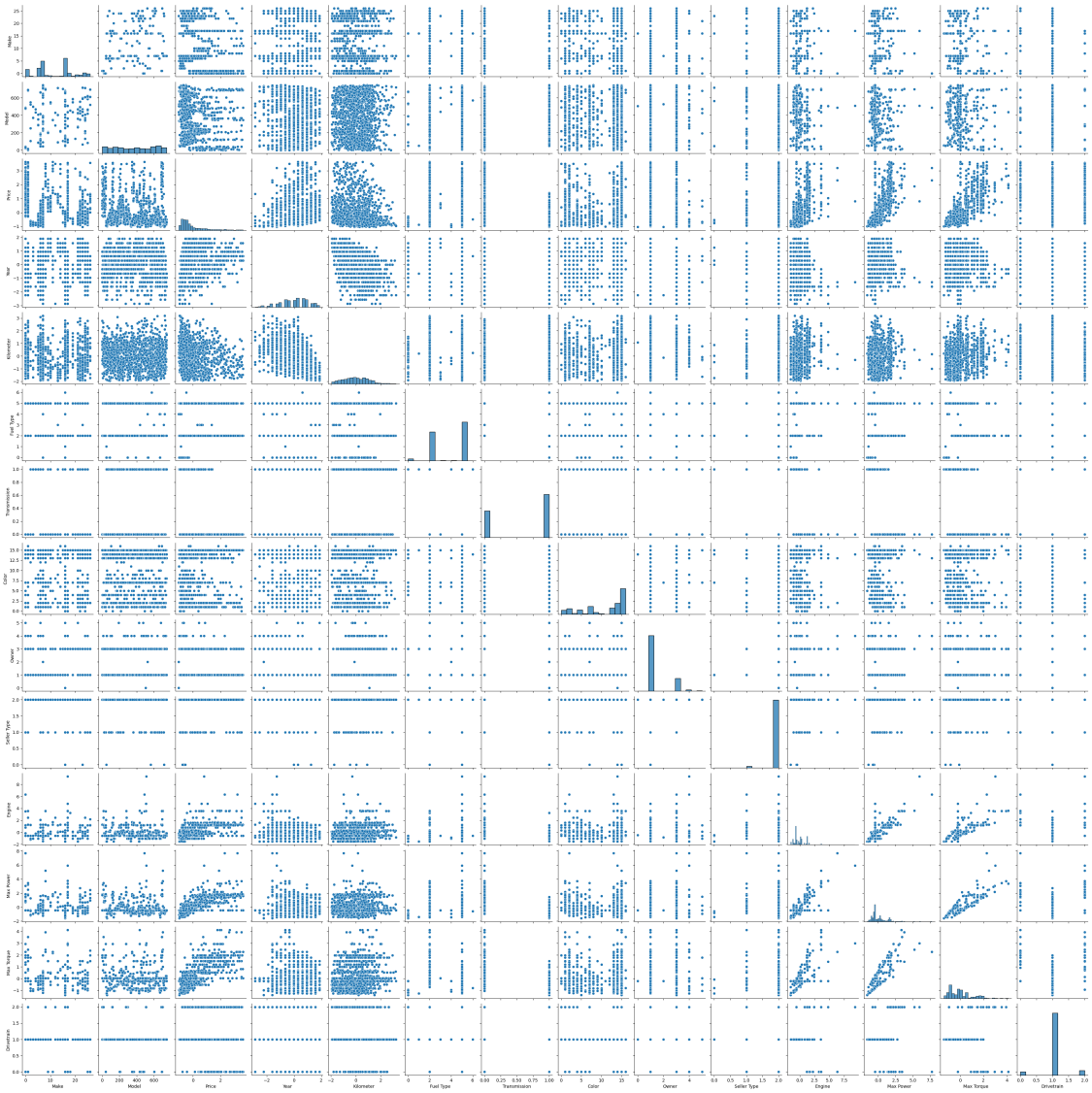
[36]:
```
       Make  Model     Price      Year  Kilometer  Fuel Type  Transmission  \
0         6     77 -0.570409  0.291181   1.350028          5             1
1        16    563 -0.636352 -0.657164   0.887643          2             1
2         7    727 -0.912116 -1.605508   0.583192          5             1
3        24    369 -0.217911  0.923410  -0.539473          5             1
5        16    166 -0.366583  0.291181   0.823518          5             1
…     …    …       …        …         …        …           …
2051     16    656 -0.066840  1.555640  -0.139880          5             1
2052      7    746 -0.684312 -0.657164   0.621248          2             1
2053     16    506 -0.882142 -0.657164   1.039869          5             1
2055      7    330 -0.846172 -0.657164   1.192094          5             1
2056      5    355 -0.888136 -0.973279   0.811530          5             1

       Color  Owner  Seller Type    Engine  Max Power  Max Torque  Drivetrain
0          7      1            1 -0.570945  -0.488157   -0.873964           1
1         15      3            2 -0.455024  -0.785964   -0.080986           1
2          8      1            2 -0.573264  -0.671423   -0.837136           1
3         13      1            2 -0.573264  -0.602698   -0.834805           1
5          7      1            2 -0.165222  -0.396524   -0.668377           1
…      …    …           …        …         …          …          …
2051      15      1            2  0.041118  -0.121626   -0.590058           1
2052      14      1            2 -0.111898  -0.442341   -0.208254           1
2053      15      3            2 -0.573264  -0.533974   -0.834805           1
2055      15      3            2 -1.461219  -1.221219   -1.206819           1
2056      14      1            2 -0.575582  -0.877596   -0.942493           1

[1496 rows x 14 columns]
```

[37]:
```
sns.pairplot(df_cleaned)
```

[37]:
```
<seaborn.axisgrid.PairGrid at 0x22f2a746e10>
```



[38]:
```
x = df_cleaned.drop(columns=['Price'])
y = df_cleaned['Price']
```

[39]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=42)
from sklearn.metrics import classification_report
```

[ ]:
```

```

[40]:
```
linear=LinearRegression()
linear.fit(x_train,y_train)
y_predict=linear.predict(x_test)
mse1 = mean_squared_error(y_test,y_predict)
r2_sq = np.sqrt(mse1)

print("MSE of LR: ",mse1)
print('R2 of LR: ', r2_sq)
```

```
MSE of LR:  0.24322150109243054
R2 of LR:  0.4931749193667806
```

[41]:
```
from sklearn.linear_model import Ridge,Lasso
from sklearn.model_selection import GridSearchCV

# Define parameter grid for Ridge regression
param_grid_ridge = {'alpha': [0.1, 1, 10, 100]}

# Create a Ridge regression model
ridge = Ridge()

# Perform grid search with cross-validation
grid_search_ridge = GridSearchCV(ridge, param_grid_ridge, cv=5,␣
 ↪scoring='neg_mean_squared_error')
grid_search_ridge.fit(x_train, y_train)

# Best parameters and score
best_params_ridge = grid_search_ridge.best_params_
best_score_ridge = -grid_search_ridge.best_score_

print("Best parameters for Ridge:", best_params_ridge)
print("Best score (MSE) for Ridge:", best_score_ridge)
```

```
Best parameters for Ridge: {'alpha': 10}
Best score (MSE) for Ridge: 0.24550850761586768
```

[42]:
```
best_ridge_model = grid_search_ridge.best_estimator_
y_pred_ridge = best_ridge_model.predict(x_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print("Test MSE for Ridge:", mse_ridge)
```

```
Test MSE for Ridge: 0.2427443602402604
```

[43]:
```
# Define parameter grid for Lasso regression
param_grid_lasso = {'alpha': [0.1, 1, 10, 100]}

# Create a Lasso regression model
```

```
lasso = Lasso()

# Perform grid search with cross-validation
grid_search_lasso = GridSearchCV(lasso, param_grid_lasso, cv=5,␣
 ↪scoring='neg_mean_squared_error')
grid_search_lasso.fit(x_train, y_train)

# Best parameters and score
best_params_lasso = grid_search_lasso.best_params_
best_score_lasso = -grid_search_lasso.best_score_

print("Best parameters for Lasso:", best_params_lasso)
print("Best score (MSE) for Lasso:", best_score_lasso)

# Evaluate on test set
best_lasso_model = grid_search_lasso.best_estimator_
y_pred_lasso = best_lasso_model.predict(x_test)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
print("Test MSE for Lasso:", mse_lasso)
```

```
Best parameters for Lasso: {'alpha': 0.1}
Best score (MSE) for Lasso: 0.30246442065761636
Test MSE for Lasso: 0.2725312880614805
```

[ ]:

5 Applying Random Forest Regressor

[44]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

[45]:
```
rfr=RandomForestRegressor(random_state=42)
```

[46]:
```
rfr.fit(x_train, y_train)
```

[46]:
```
RandomForestRegressor(random_state=42)
```

[47]:
```
y_pred = rfr.predict(x_test)

mse= mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Without Hyperparameter Tuning:")
print("Mean Squared Error:", mse)
```

```
print("R-squared:", r2)
```

```
Without Hyperparameter Tuning:
Mean Squared Error: 0.08856394845372201
R-squared: 0.912306620755473
```

[48]:
```
# with hyperparametric tuning
```

[49]:
```
param_grid={
    'n_estimators':[50,100,200],
    'max_features':['auto','sqrt'],
    'max_depth':[10,20,30],
    'min_samples_split':[2,5,10],
    'min_samples_leaf':[1,2,4]
}
```

[50]:
```
grid_search =GridSearchCV(estimator=rfr,param_grid=param_grid , cv=5,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1,verbose=2)
```

[51]:
```
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
```

[51]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [10, 20, 30],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]},
             scoring='neg_mean_squared_error', verbose=2)
```

[52]:
```
best_params = grid_search.best_params_
print(best_params)
best_rfr= grid_search.best_estimator_
print(best_rfr)
```

```
{'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 200}
RandomForestRegressor(max_depth=20, max_features='sqrt', n_estimators=200,
                      random_state=42)
```

[53]:
```
y_pred=best_rfr.predict(x_test)
```

[54]:
```
mse=mean_squared_error(y_test,y_pred)
mse
```

[54]:
```
0.0860003557006667
```

[55]:
```
r2_score= r2_score(y_test,y_pred)
r2_score
```

[55]:
```
0.9148450138087103
```

[56]:
```
from sklearn.svm import SVR

# SVR without tuning
svr_model = SVR()
svr_model.fit(x_train, y_train)
y_pred_svr = svr_model.predict(x_test)
```

[57]:
```
from sklearn.metrics import r2_score

# Evaluate SVR
r2_svr = r2_score(y_test, y_pred_svr)
mse_svr = mean_squared_error(y_test, y_pred_svr)
print(f"SVR R2: {r2_svr}")
print(f"SVR MSE: {mse_svr}")
```

```
SVR R2: -0.09972620190197734
SVR MSE: 1.1106436483291606
```

[58]:
```
# SVR with tuning
param_grid_svr = {'C': [0.1, 1, 10], 'epsilon': [0.1, 0.01]}
svr_cv = GridSearchCV(SVR(), param_grid_svr, cv=5)
svr_cv.fit(x_train, y_train)
y_pred_svr_tuned = svr_cv.predict(x_test)
```

[59]:
```
# Evaluate tuned SVR
r2_svr_tuned = r2_score(y_test, y_pred_svr_tuned)
mse_svr_tuned = mean_squared_error(y_test, y_pred_svr_tuned)
print(f"Tuned SVR R2: {r2_svr_tuned}")
print(f"Tuned SVR MSE: {mse_svr_tuned}")
```

```
Tuned SVR R2: 0.17398036676004547
Tuned SVR MSE: 0.8342198789721214
```

Based on the results from applying the **Random Forest Regressor** to the **Car Details v4** dataset, the model achieved a **Mean Squared Error (MSE) of 0.0875**, indicating a relatively low prediction error. Additionally, the **R-squared value of 0.9134** suggests that the model explains approximately **91.34% of the variance** in the data, making it a strong predictive model for this dataset.