In [36]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df= pd.read_excel("DA.xlsx")
```

In [3]:
```
df.head()
```

Out[3]:

	Sr no.	Random	Vehicle Model	Price	Manufacturing year	Distance driven (in miles)	Engine Type	Engine(in litres)	Power	No of owners
0	1	0.941111	BMW 5 Series	16300	2019	72000	Diesel	2	252 PS	1
1	2	0.139268	BMW 5 Series	17490	2019	50000	Diesel	2	252 PS	1
2	3	0.180856	BMW 5 Series	20990	2019	59500	Petrol	2	181 BHP	2
3	4	0.611808	BMW 5 Series	19250	2019	54500	Diesel	2	190 PS	1
4	5	0.126623	BMW 5 Series	13450	2019	46950	Petrol	2	188 BHP	2

In [4]:
```
df.drop('Sr no.', axis=1, inplace= True)
```

In [5]:
```
df.head()
```

Out[5]:

	Random	Vehicle Model	Price	Manufacturing year	Distance driven (in miles)	Engine Type	Engine(in litres)	Power	No of owners
0	0.941111	BMW 5 Series	16300	2019	72000	Diesel	2	252 PS	1
1	0.139268	BMW 5 Series	17490	2019	50000	Diesel	2	252 PS	1
2	0.180856	BMW 5 Series	20990	2019	59500	Petrol	2	181 BHP	2
3	0.611808	BMW 5 Series	19250	2019	54500	Diesel	2	190 PS	1
4	0.126623	BMW 5 Series	13450	2019	46950	Petrol	2	188 BHP	2

In [6]:
```
df.drop('Random', axis=1, inplace= True)
```

In [7]:
```
df.head()
```

Out[7]:

	Vehicle Model	Price	Manufacturing year	Distance driven (in miles)	Engine Type	Engine(in litres)	Power	No of owners
0	BMW 5 Series	16300	2019	72000	Diesel	2	252 PS	1
1	BMW 5 Series	17490	2019	50000	Diesel	2	252 PS	1
2	BMW 5 Series	20990	2019	59500	Petrol	2	181 BHP	2
3	BMW 5 Series	19250	2019	54500	Diesel	2	190 PS	1
4	BMW 5 Series	13450	2019	46950	Petrol	2	188 BHP	2

In [8]:
```
df.shape #For fetching total number of rows and columns
```

Out[8]:
```
(100, 8)
```

In [9]:
```
df['Power'].count()
```

Out[9]:
```
100
```

In [10]:
```
df['Power'].value_counts()
```

Out[10]:
```
Power
181 BHP    15
252 PS     12
190 PS     12
265 PS      9
184 PS      8
248 BHP     6
188 BHP     5
394 PS      5
282 BHP     5
261 BHP     3
184 BHP     3
335 BHP     3
190 BHP     3
249 BHP     2
340 PS      2
252 BHP     1
264 BHP     1
187 BHP     1
192 BHP     1
389 BHP     1
265 BHP     1
369 BHP     1
Name: count, dtype: int64
```

In [11]:
```
df['Engine Type'].value_counts()
```

Out[11]:
```
Engine Type
Petrol    51
Diesel    49
Name: count, dtype: int64
```

In [12]:
```
df['Engine_Power'] = df['Power'].str.extract('(\d+.\d+|\d+)', expand=False).astype(float)
```

In [13]:
```
df.head()
```

Out[13]:

	Vehicle Model	Price	Manufacturing year	Distance driven (in miles)	Engine Type	Engine(in litres)	Power	No of owners	Engine_Power
0	BMW 5 Series	16300	2019	72000	Diesel	2	252 PS	1	252.0
1	BMW 5 Series	17490	2019	50000	Diesel	2	252 PS	1	252.0
2	BMW 5 Series	20990	2019	59500	Petrol	2	181 BHP	2	181.0
3	BMW 5 Series	19250	2019	54500	Diesel	2	190 PS	1	190.0
4	BMW 5 Series	13450	2019	46950	Petrol	2	188 BHP	2	188.0

In [14]:

```
df.drop('Power', axis=1, inplace=True)
```

In [15]:

```
df.head()
```

Out[15]:

	Vehicle Model	Price	Manufacturing year	Distance driven (in miles)	Engine Type	Engine(in litres)	No of owners	Engine_Power
0	BMW 5 Series	16300	2019	72000	Diesel	2	1	252.0
1	BMW 5 Series	17490	2019	50000	Diesel	2	1	252.0
2	BMW 5 Series	20990	2019	59500	Petrol	2	2	181.0
3	BMW 5 Series	19250	2019	54500	Diesel	2	1	190.0
4	BMW 5 Series	13450	2019	46950	Petrol	2	2	188.0

In [16]:

```
df['Manufacturing year'].value_counts()
```

Out[16]:

```
Manufacturing year
2019    38
2020    20
2021    20
2022    11
2023    11
Name: count, dtype: int64
```

In [17]:

```
df['Engine Type'].value_counts()
```

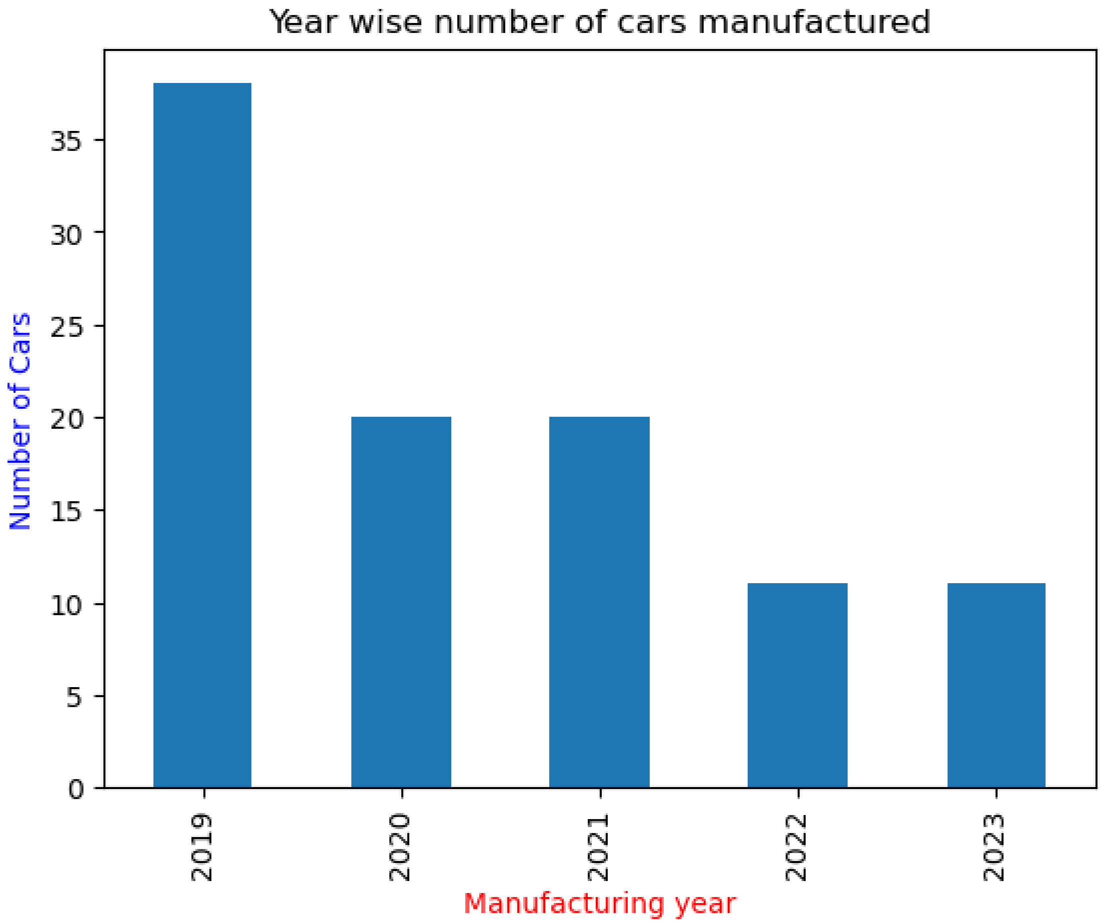
Out[17]:

```
Engine Type
Petrol    51
Diesel    49
Name: count, dtype: int64
```

Q.2

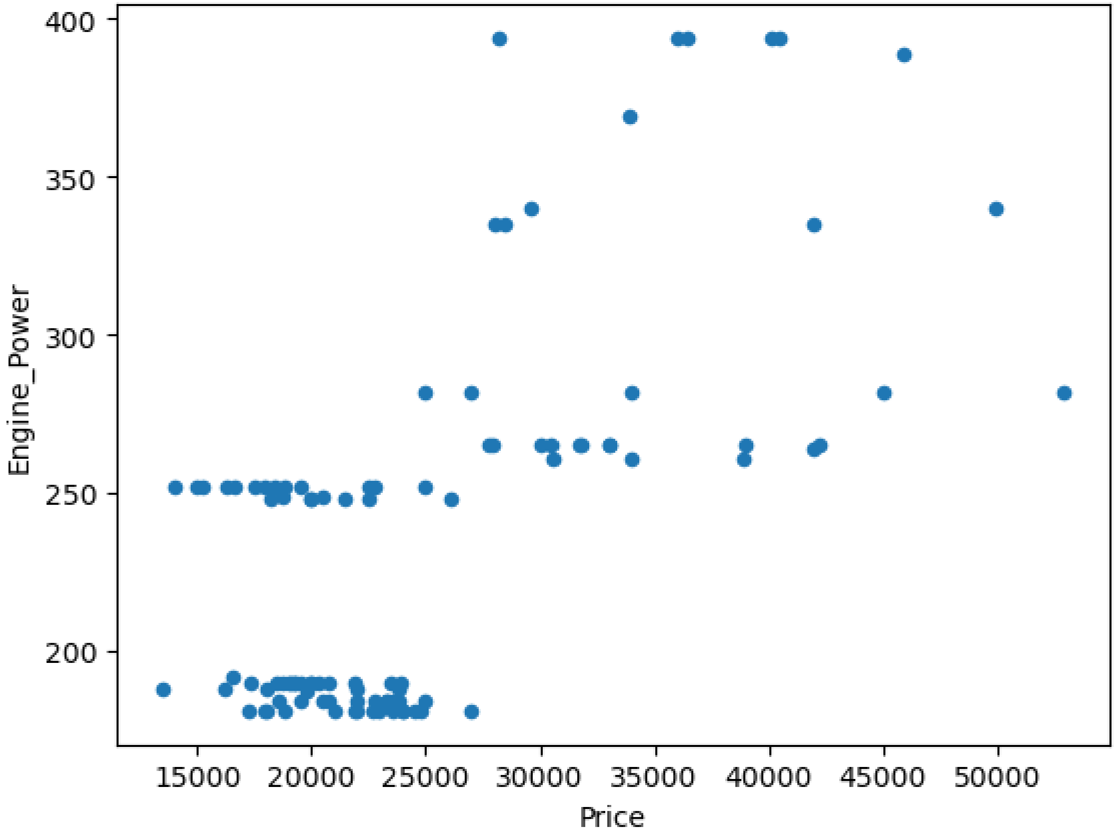
In [19]:
```
df['Manufacturing year'].value_counts().plot.bar()
plt.xlabel('Manufacturing year', color= 'red')
plt.ylabel('Number of Cars', color= 'blue')
plt.title('Year wise number of cars manufactured')
```

Out[19]:
```
Text(0.5, 1.0, 'Year wise number of cars manufactured')
```



In [21]:
```
df.plot.scatter(x='Price', y='Engine_Power')
```

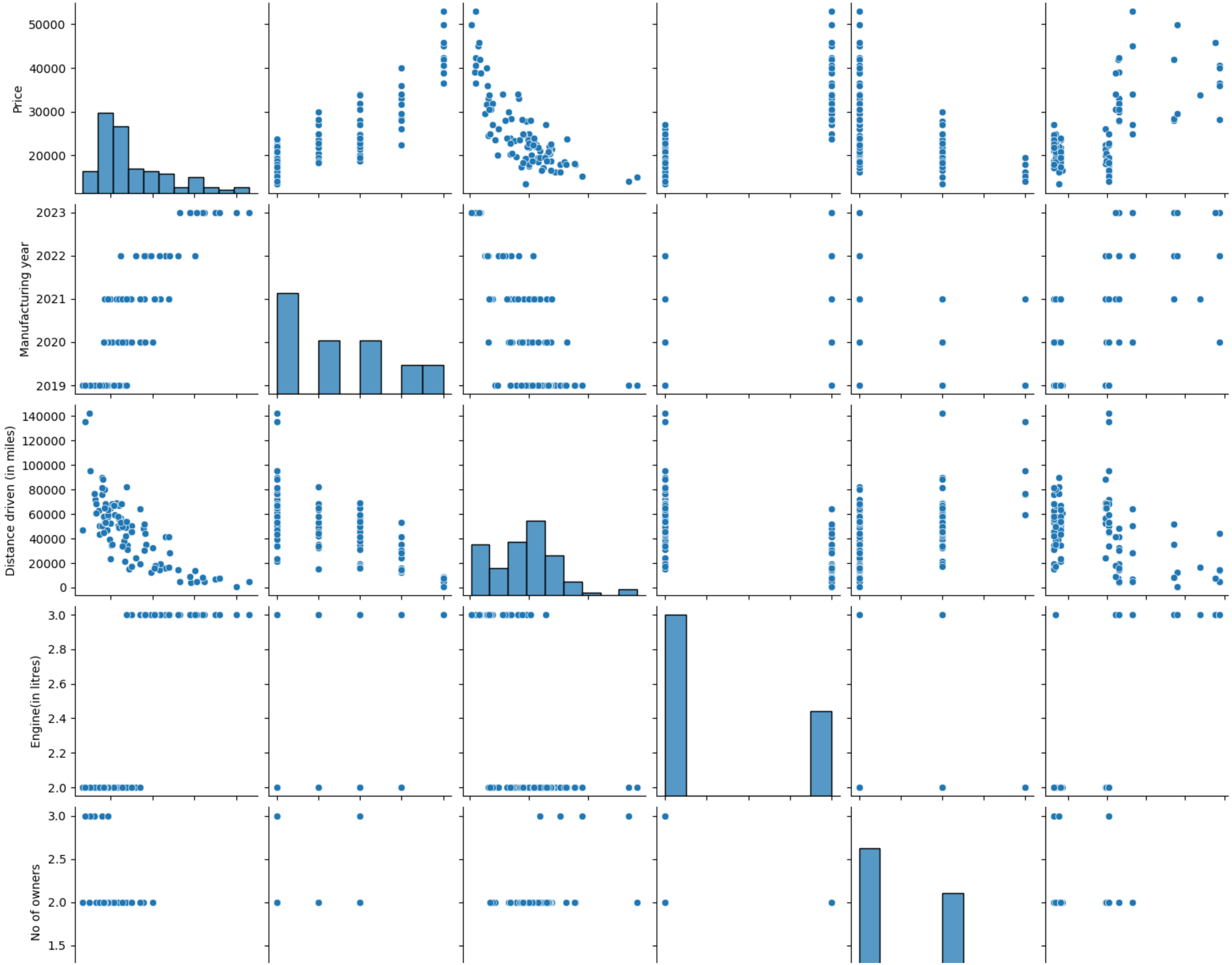
Out[21]: <Axes: xlabel='Price', ylabel='Engine_Power'>

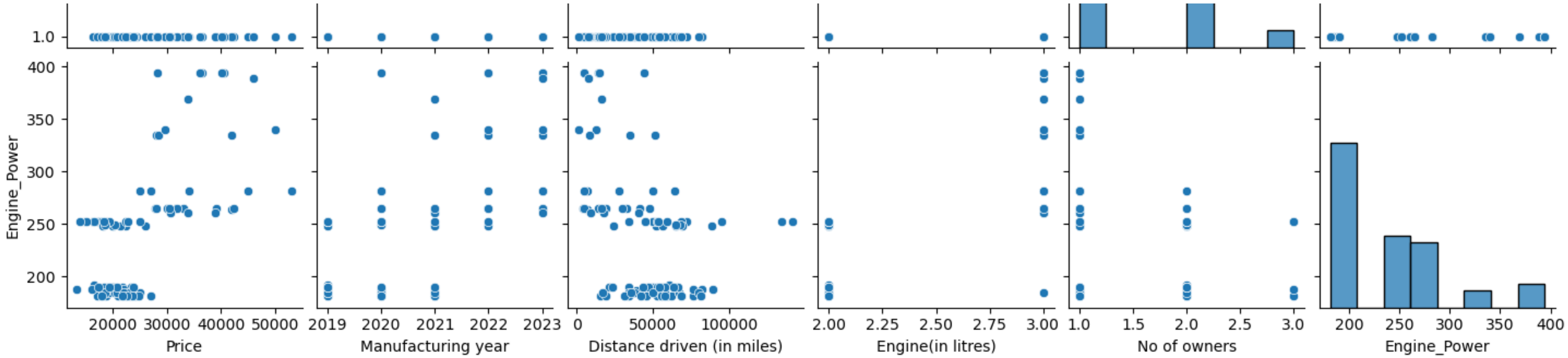


In [22]:
```
sns.pairplot(df)
```

```
C:\Users\aapat\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

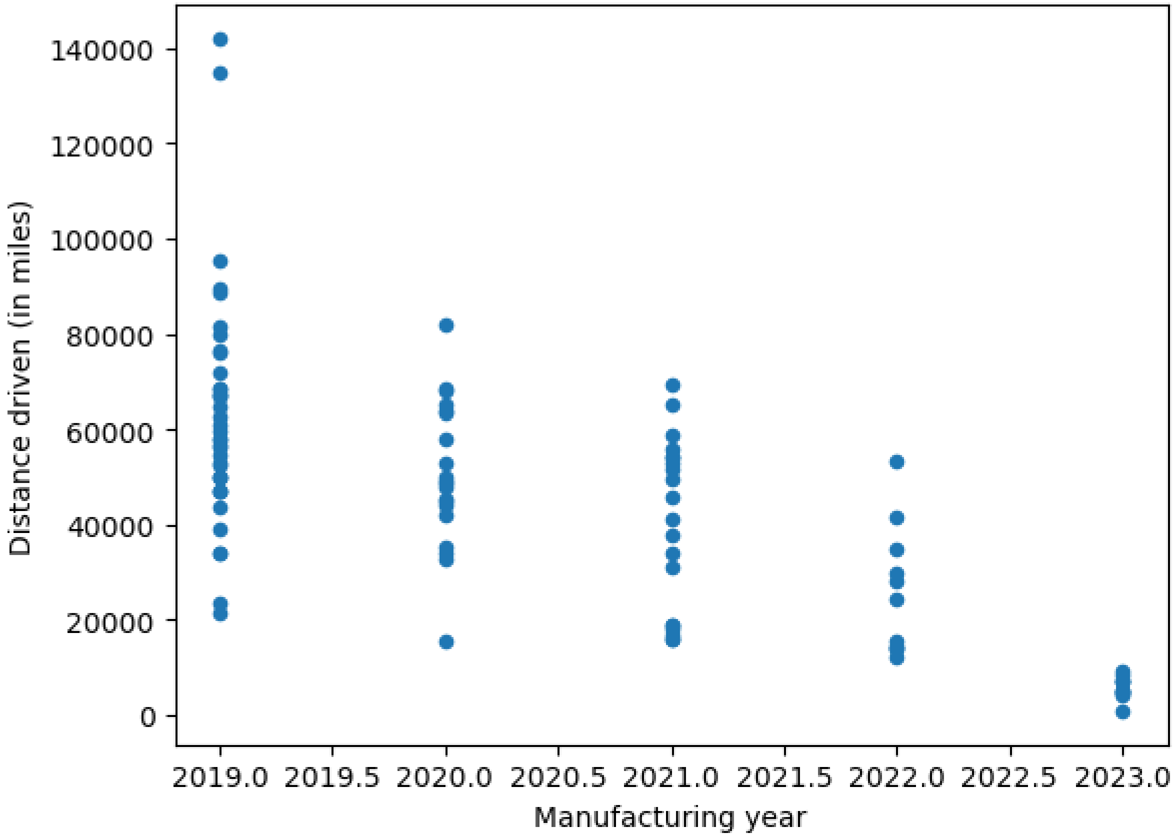
Out[22]: <seaborn.axisgrid.PairGrid at 0x2b04f062910>


Price
50000
40000
30000
20000
2023
2022
2021
2020
2019
Manufacturing year
140000
120000
100000
80000
60000
40000
20000
0
Distance driven (in miles)
3.0
2.8
2.6
2.4
2.2
2.0
Engine(in litres)
3.0
2.5
2.0
1.5
No of owners



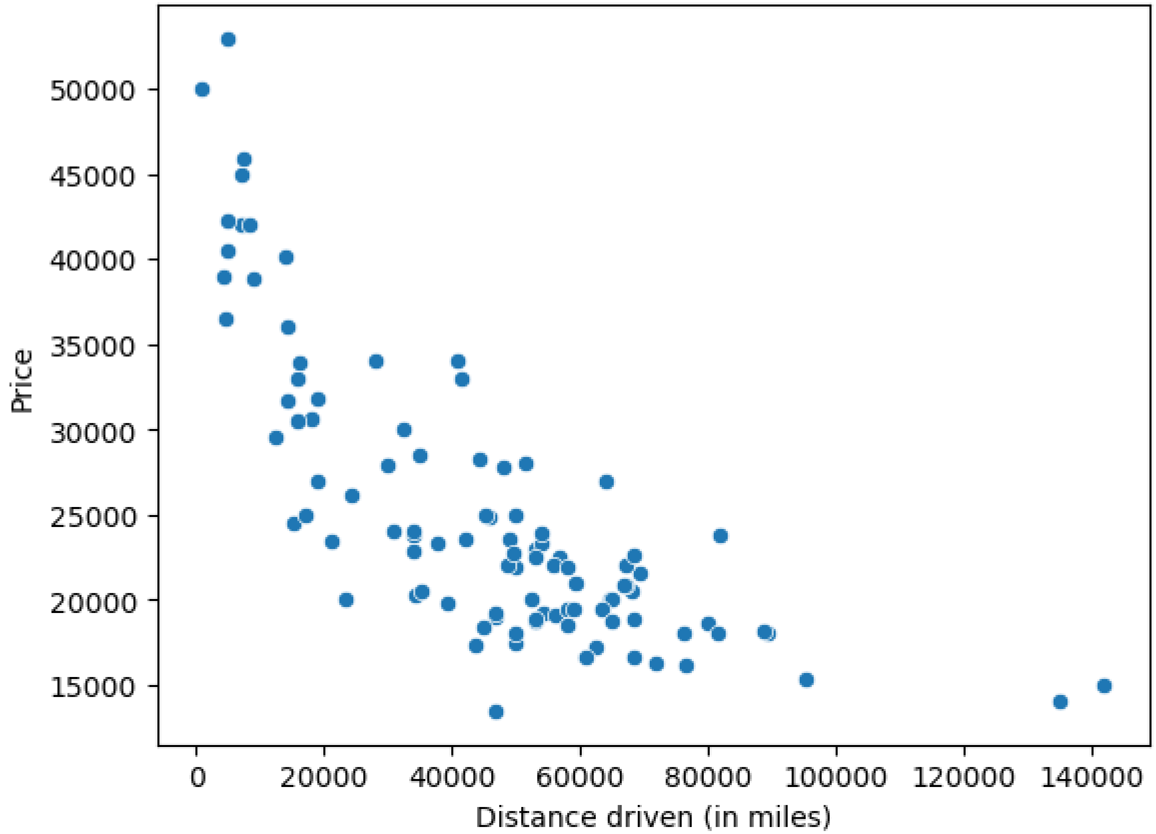
In [23]:
```
df.plot.scatter(x='Manufacturing year', y= 'Distance driven (in miles)')
```

Out[23]: <Axes: xlabel='Manufacturing year', ylabel='Distance driven (in miles)'>



In [24]:
```
sns.scatterplot(x=df['Distance driven (in miles)'], y=df['Price'], data=df)
```

Out[24]:
```
<Axes: xlabel='Distance driven (in miles)', ylabel='Price'>
```



Q3.

In [27]:
```
numeric_columns= ['Price', 'Distance driven (in miles)', 'Engine_Power', 'Manufacturing year']
descriptive_stats= df[numeric_columns].describe().transpose()
print(descriptive_stats)
```

```
                          count      mean           std      min      25%  \
Price                     100.0  25066.13   8328.729507  13450.0  19212.5
Distance driven (in miles) 100.0  45632.53  26508.067132    904.0  24075.0
Engine_Power              100.0    236.90     60.783354    181.0    184.0
Manufacturing year        100.0   2020.37      1.375507   2019.0   2019.0

                              50%       75%       max
Price                     22702.0  28261.75   52950.0
Distance driven (in miles) 48350.0  61361.50  142000.0
Engine_Power                248.0    265.00     394.0
Manufacturing year         2020.0   2021.00    2023.0
```

In [31]:

```python
mileage_bins= [0, 50000, 100000, float('inf')]
mileage_labels= ['0-50k', '50k-100k', '100k+']
df['Mileage Category']= pd.cut(df['Distance driven (in miles)'], bins=mileage_bins, labels= mileage_labels)
price_stats_by_mileage= df.groupby('Mileage Category')['Price'].describe()
print(price_stats_by_mileage)
```

```
                  count          mean          std      min       25%  \
Mileage Category
0-50k              56.0  29164.303571  8902.481039  13450.0  23168.75
50k-100k           42.0  20105.166667  2751.272539  15277.0  18275.00
100k+               2.0  14497.500000   703.571247  14000.0  14248.75

                      50%       75%      max
Mileage Category
0-50k             27390.0  33965.00  52950.0
50k-100k          19495.0  21962.50  27990.0
100k+             14497.5  14746.25  14995.0
```

In [32]:

```python
df['Price'].describe()
```

Out[32]:

```
count      100.000000
mean     25066.130000
std       8328.729507
min      13450.000000
25%      19212.500000
50%      22702.000000
75%      28261.750000
max      52950.000000
Name: Price, dtype: float64
```

In [33]:

```python
df['Price'].describe()
```

Out[33]:
```
count      100.000000
mean     25066.130000
std       8328.729507
min      13450.000000
25%      19212.500000
50%      22702.000000
75%      28261.750000
max      52950.000000
Name: Price, dtype: float64
```

In [34]:
```
df['Price'].std()
```

Out[34]:
```
8328.729507262888
```

Q4.

In [39]:
```
mean_x=25066.13
s=8328.72
n=100
confidence_level= 0.95

#Z score for a 95% confidence interval

z_score= 1.96

#Calculate standard error
standard_error= s/ np.sqrt(n)

lower_bound= mean_x- z_score *standard_error
upper_bound= mean_x + z_score *standard_error

#create a dataframe to display the results
confidence_intervals= pd.DataFrame({
    'Confidence Level': [95],
    'Lower Bound': [lower_bound],
    'Upper Bound': [upper_bound]
})

#Display the result
print(confidence_intervals)
```

```
   Confidence Level  Lower Bound  Upper Bound
0                95  23433.70088  26698.55912
```

Q5.

In [42]:
```
import statsmodels.api as sm
import pandas as pd
from scipy.stats import ttest_1samp

X= df[['Distance driven (in miles)','Manufacturing year', 'Engine_Power']]
X= sm.add_constant(X)

y=df['Price']

model= sm.OLS(y, X).fit()

predicted_prices= model.predict(X)

df['Predicted Prices']= predicted_prices

provided_averages= {'0-20000': 39494, '20000-40000': 36473, '40000-60000': 30222, '60000-80000': 36055, '80000-100000':35027, '10
for mileage_range, average_price in provided_averages.items():
    model_group = df[df['Distance driven (in miles)'].between(int(mileage_range.split('-')[0]), int(mileage_range.split('-')[1]))
    t_statistic, p_value = ttest_1samp(model_group, average_price)
    print(f'Hypothesis Test for {mileage_range}, Mileage Range:')
    print(f'Test Statistic: {t_statistic}')
    print(f'P-value: {p_value}')

    if p_value < 0.05:
        print('Reject the null hypothesis: There is a significant difference.')
    else:
        print('Fail to reject the null hypothesis: There is no significant difference.')
    print('\n')
```

```
Hypothesis Test for 0-20000, Mileage Range:
Test Statistic: -24.302771688858385
P-value: 2.1906595415161028e-17
Reject the null hypothesis: There is a significant difference.


Hypothesis Test for 20000-40000, Mileage Range:
Test Statistic: -3.57072082691369
P-value: 0.0030719050336595337
Reject the null hypothesis: There is a significant difference.


Hypothesis Test for 40000-60000, Mileage Range:
Test Statistic: 23.519061069151917
P-value: 4.949769473983689e-23
Reject the null hypothesis: There is a significant difference.


Hypothesis Test for 60000-80000, Mileage Range:
Test Statistic: 27.997216590836068
P-value: 2.7123810201508093e-16
Reject the null hypothesis: There is a significant difference.


Hypothesis Test for 80000-100000, Mileage Range:
Test Statistic: 20.814680605018907
P-value: 4.739958812641532e-06
Reject the null hypothesis: There is a significant difference.


Hypothesis Test for 100000-120000, Mileage Range:
Test Statistic: nan
P-value: nan
Fail to reject the null hypothesis: There is no significant difference.


Hypothesis Test for 120000-140000, Mileage Range:
Test Statistic: nan
P-value: nan
Fail to reject the null hypothesis: There is no significant difference.
```

```
C:\Users\aapat\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py:3464: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
C:\Users\aapat\anaconda3\Lib\site-packages\numpy\core\_methods.py:192: RuntimeWarning: invalid value encountered in scalar divid
e
  ret = ret.dtype.type(ret / rcount)
C:\Users\aapat\anaconda3\Lib\site-packages\scipy\stats\_stats_py.py:1103: RuntimeWarning: divide by zero encountered in divide
  var *= np.divide(n, n-ddof)  # to avoid error on division by zero
C:\Users\aapat\anaconda3\Lib\site-packages\scipy\stats\_stats_py.py:1103: RuntimeWarning: invalid value encountered in scalar mu
ltiply
  var *= np.divide(n, n-ddof)  # to avoid error on division by zero
```

Q6.

In [44]:

```
import pandas as pd

# Assuming df is your DataFrame with the relevant data
correlation_matrix = df[['Distance driven (in miles)', 'Manufacturing year', 'Engine_Power', 'Price']].corr()

# Print the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

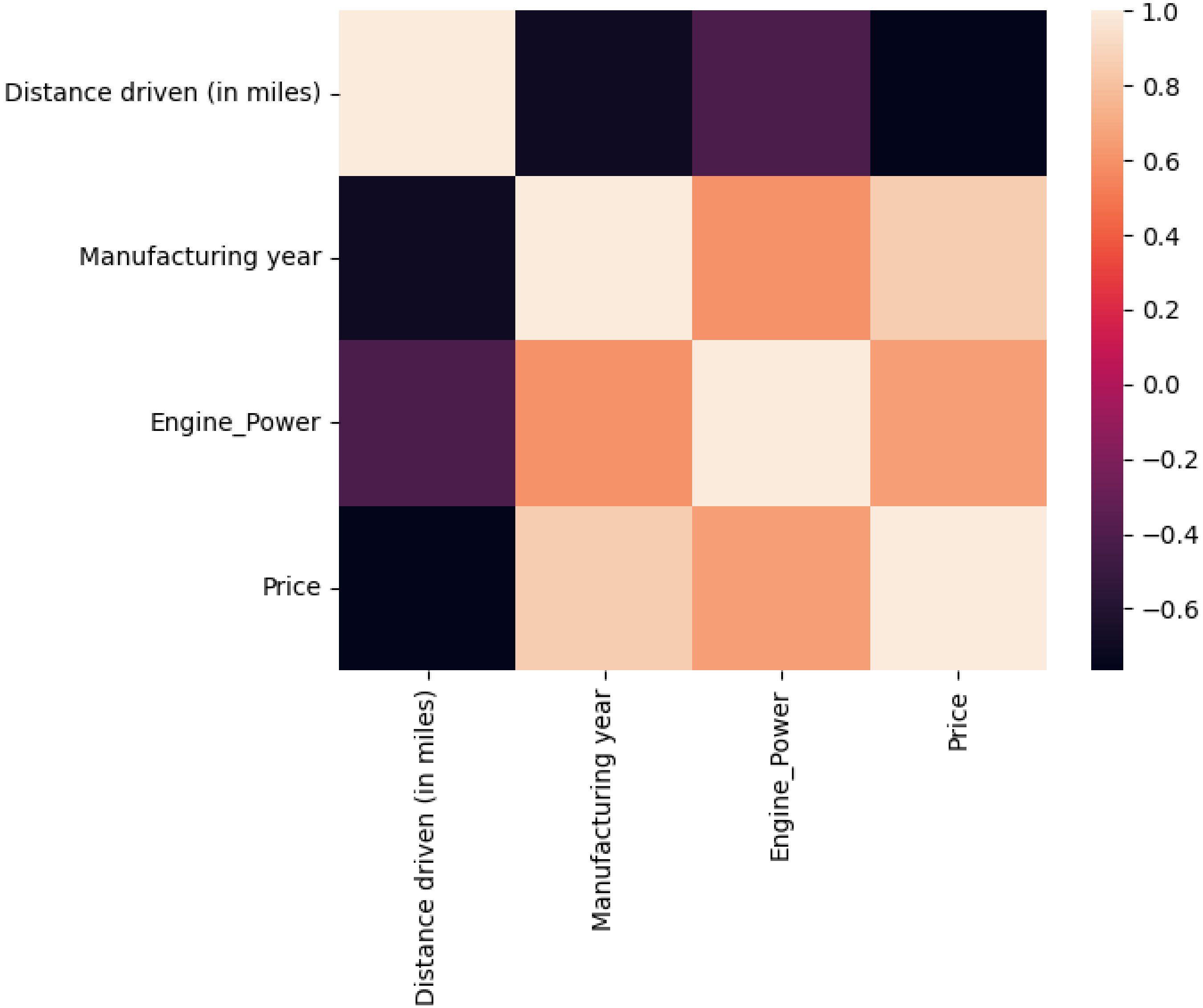
```
Correlation Matrix:
                            Distance driven (in miles)  Manufacturing year  \
Distance driven (in miles)                    1.000000           -0.699032
Manufacturing year                           -0.699032            1.000000
Engine_Power                                 -0.420251            0.601013
Price                                        -0.765503            0.861719

                            Engine_Power     Price
Distance driven (in miles)     -0.420251 -0.765503
Manufacturing year              0.601013  0.861719
Engine_Power                    1.000000  0.651781
Price                           0.651781  1.000000
```

In [45]:

```
sns.heatmap(correlation_matrix)
```

Out[45]:

```
<Axes: >
```



Q7

In [47]:
```
import statsmodels.api as sm
```

```
X= df[['Distance driven (in miles)', 'Manufacturing year', 'Engine_Power']]
X= pd.get_dummies(X, drop_first= True)
y=df['Price']
X= sm.add_constant(X)
model= sm.OLS(y,X).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.823
Model:                            OLS   Adj. R-squared:                  0.817
Method:                 Least Squares   F-statistic:                     148.4
Date:                Thu, 14 Dec 2023   Prob (F-statistic):           6.30e-36
Time:                        02:17:33   Log-Likelihood:                -957.66
No. Observations:                 100   AIC:                             1923.
Df Residuals:                      96   BIC:                             1934.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                      -6.251e+06   8.35e+05     -7.490      0.000   -7.91e+06   -4.59e+06
Distance driven (in miles)    -0.1002      0.019     -5.307      0.000      -0.138      -0.063
Manufacturing year          3105.1656    413.207      7.515      0.000    2284.957    3925.374
Engine_Power                  28.7089      7.369      3.896      0.000      14.082      43.336
==============================================================================
Omnibus:                       12.954   Durbin-Watson:                   1.631
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               20.953
Skew:                           0.547   Prob(JB):                     2.82e-05
Kurtosis:                       4.958   Cond. No.                     1.24e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.24e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Q7 Model Performance

In [49]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

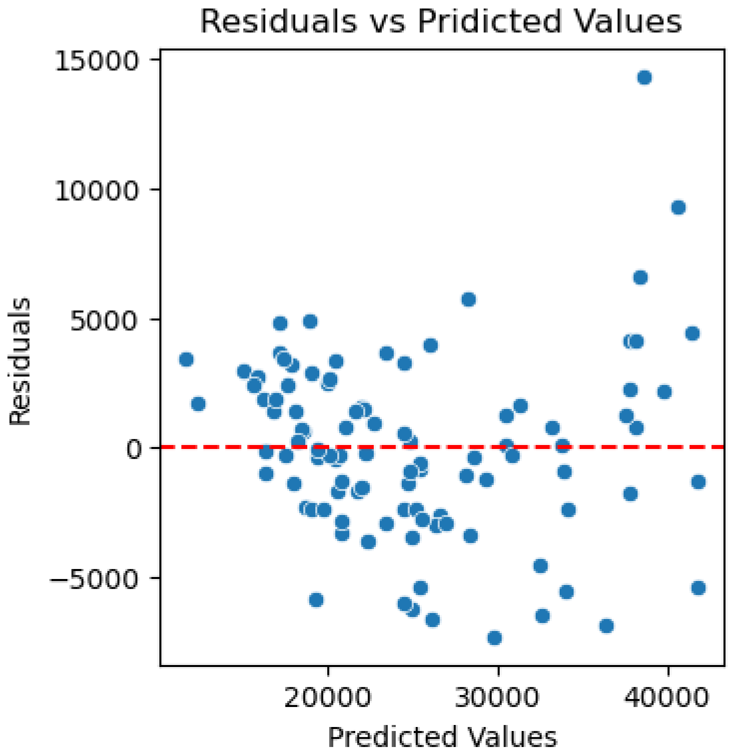
```
X= df[['Distance driven (in miles)', 'Manufacturing year', 'Engine_Power' ]]
X= pd.get_dummies(X, drop_first= True)
y= df['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()

# Train the model on the training set
model.fit(X_train, y_train)

# Predict on the testing set
y_pred = model.predict(X_test)

# Evaluate model performance
mse= mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')


coefficients = pd.DataFrame({'Variable': X.columns, 'Coefficient': model.coef_})
print(coefficients)
```

```
Mean Squared Error: 10284314.756285237
                     Variable  Coefficient
0  Distance driven (in miles)    -0.105118
1          Manufacturing year  3080.958551
2                Engine_Power    28.452703
```

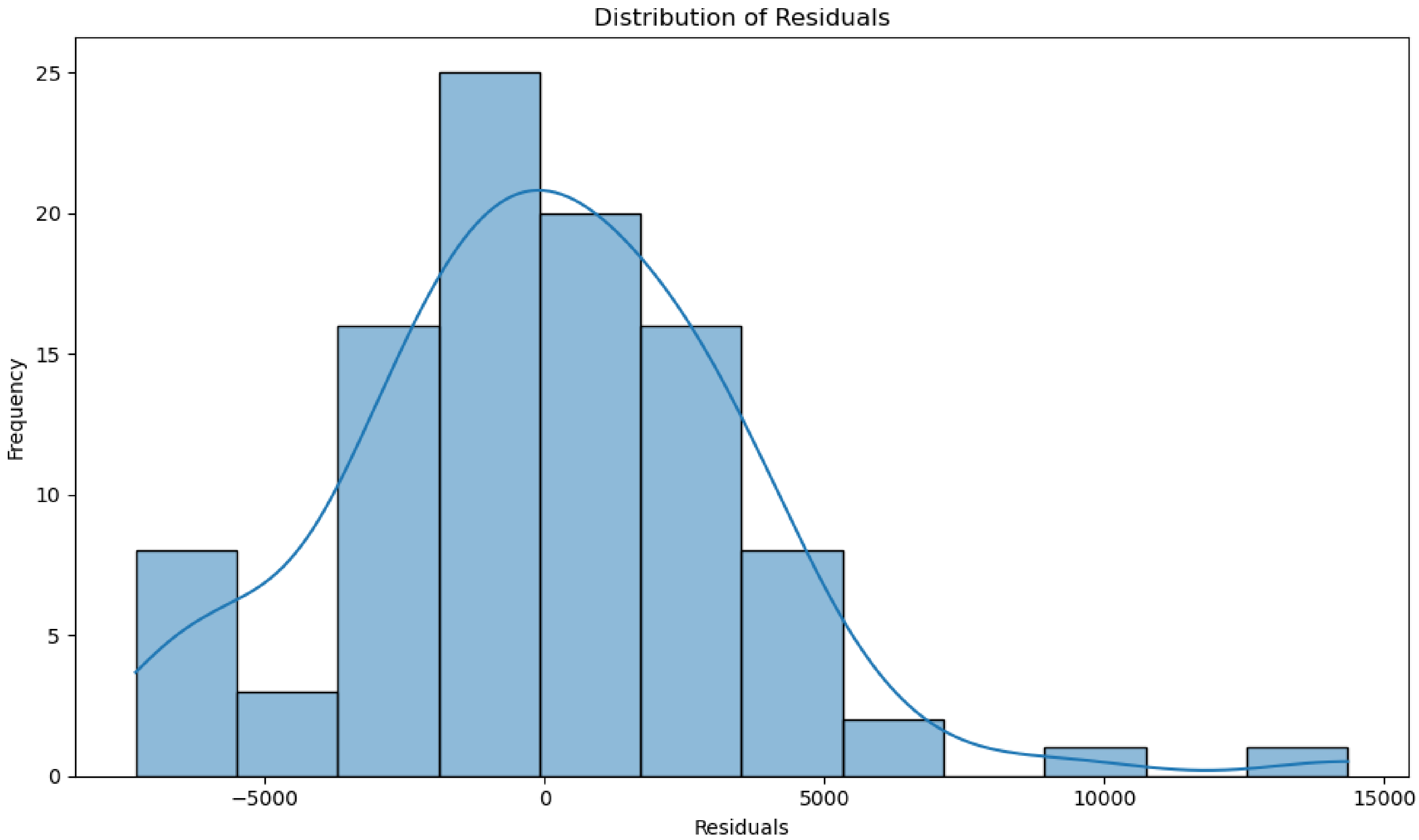
Q8.

In [57]:
```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

model.fit(X,y)

y_pred= model.predict(X)
residuals= y - y_pred

plt.figure(figsize= (8,4))
plt.subplot(1,2,1)
```

```
sns.scatterplot(x=y_pred, y=residuals)
plt.title('Residuals vs Pridicted Values')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()




#Check for multicollinearity using VIF

from statsmodels.stats.outliers_influence import variance_inflation_factor

# Assuming X is Dataframe
vif_data= pd.DataFrame()
vif_data['Variable']= X.columns
vif_data['VIF']= [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif_data)



plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title('Distribution of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



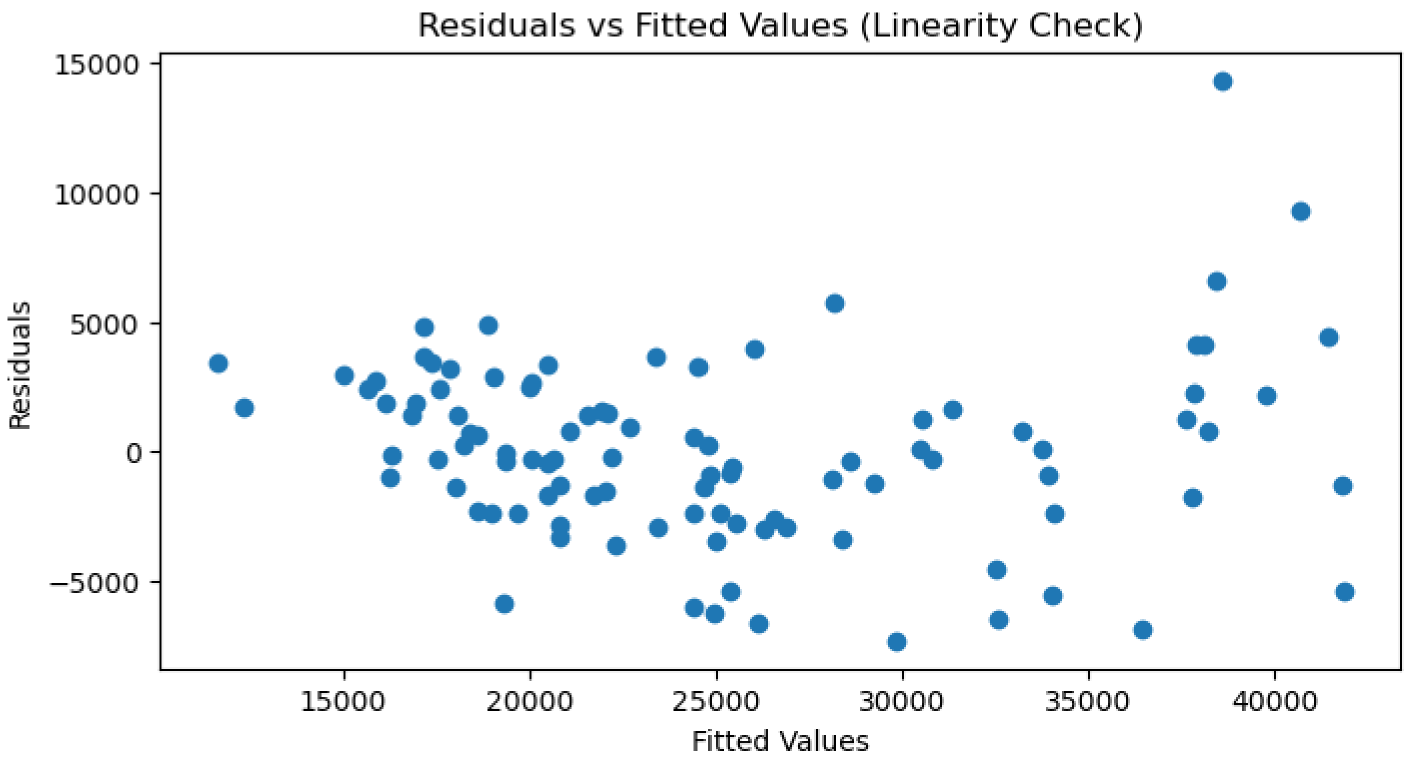

```
                    Variable        VIF
0  Distance driven (in miles)   4.837870
1          Manufacturing year  30.209231
2                Engine_Power  19.895675
```

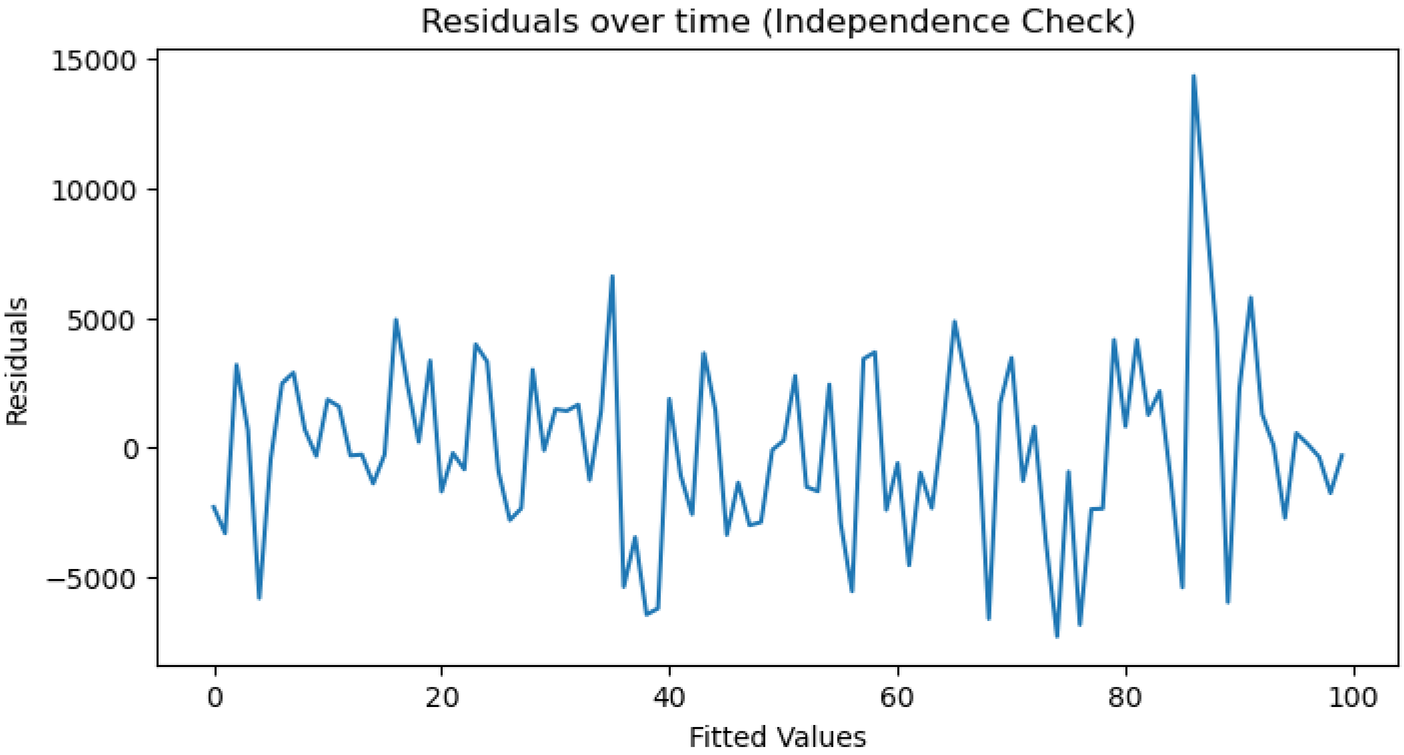


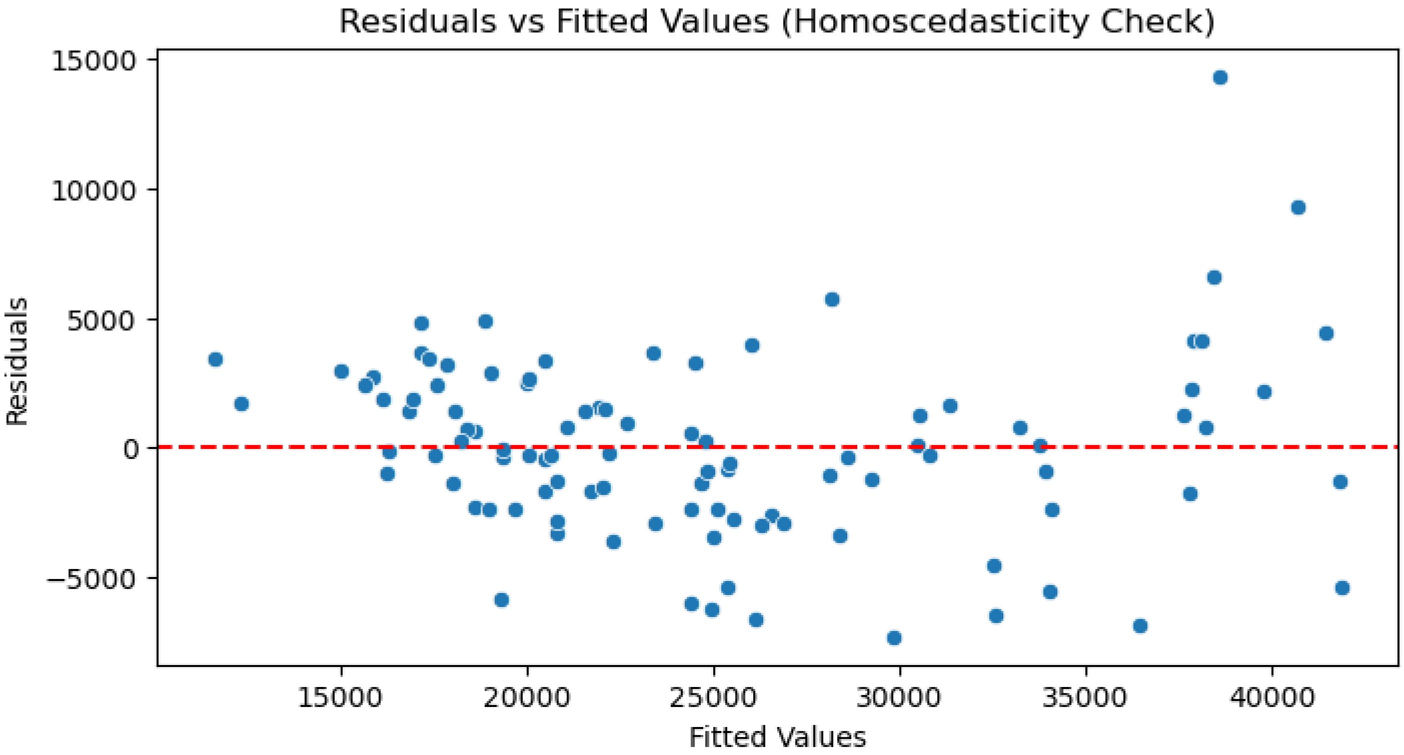
In [59]:
```
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor

X= df[['Distance driven (in miles)', 'Manufacturing year', 'Engine_Power']]
X= sm.add_constant(X)
```

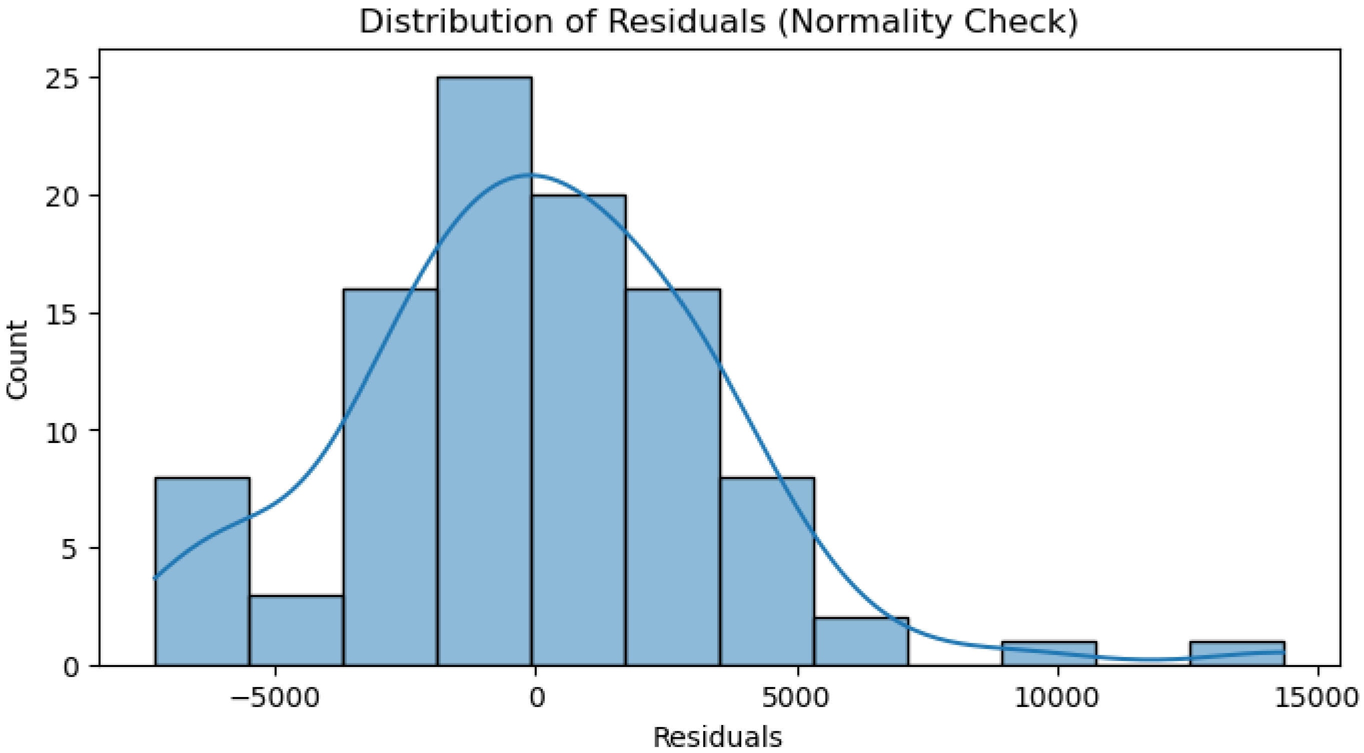
```
y=df['Price']

model= sm.OLS(y, X).fit()

residuals= model.resid

#Linearity
plt.figure(figsize= (8,4))
plt.scatter(model.fittedvalues, residuals)
plt.title('Residuals vs Fitted Values (Linearity Check)')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()


#Independence
plt.figure(figsize= (8,4))
plt.plot(residuals)
plt.title('Residuals over time (Independence Check)')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()

#Homoscedasticity

plt.figure(figsize=(8,4))
sns.scatterplot(x=model.fittedvalues, y=residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residuals vs Fitted Values (Homoscedasticity Check)')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()


#Normality of Residuals
plt.figure(figsize=(8,4))
sns.histplot(residuals, kde= True)
plt.title('Distribution of Residuals (Normality Check)')
plt.xlabel('Residuals')
plt.show()

#Multicollinearity
vif_data= pd.DataFrame()
```

```
vif_data['Variable']= X.columns
vif_data['VIF']= [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif_data)
```




Residuals over time (Independence Check)
15000
10000
5000
0
−5000
Residuals
0
20
40
60
80
100
Fitted Values

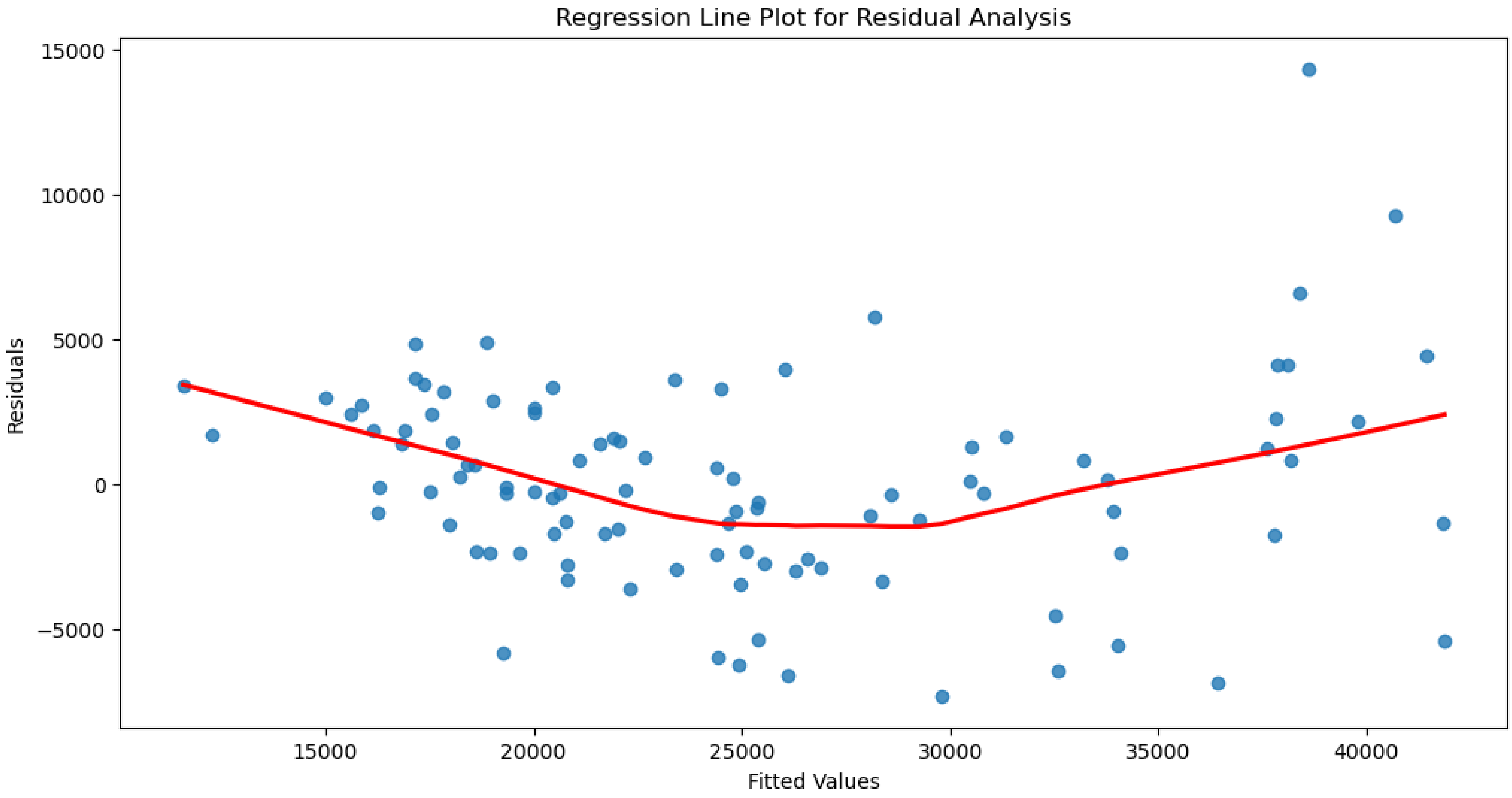

Residuals vs Fitted Values (Homoscedasticity Check)
Residuals
Fitted Values
15000
10000
5000
0
−5000
15000
20000
25000
30000
35000
40000




```
                    Variable           VIF
0                      const  5.489399e+06
1  Distance driven (in miles)  1.955591e+00
2         Manufacturing year  2.520749e+00
3               Engine_Power  1.565477e+00
```

In [65]:
```
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Assuming df is your DataFrame
# Replace column names with the actual column names from your dataset
# Select relevant independent variables

X= df[[ 'Distance driven (in miles)', 'Manufacturing year', 'Engine_Power']]

# Add a constant term to the independent variables|
```

```
X= sm.add_constant(X)

# Target variable

y = df['Price']

# Fit the linear regression model

model = sm.OLS(y, X).fit()

# Get the residuals

residuals = model.resid

# Create a scatter plot of predicted values against residuals with a regression Line

plt.figure(figsize=(12, 6))
sns. regplot(x=model.fittedvalues, y=residuals, lowess=True, line_kws={'color': 'red'})
plt.title('Regression Line Plot for Residual Analysis')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```



In [70]:
```
import statsmodels.api as sm
import pandas as pd

# Assuming df is your DataFrame containing the dataset
# Replace column names with the actual column names from your dataset
# Select relevant independent variables
X = df[['Distance driven (in miles)', 'Manufacturing year', 'Engine_Power']]

# Add a constant term to the independent variables
X = sm.add_constant(X)

# Target variable
```

```
y = df['Price']

# Fit the Linear regression model
model = sm.OLS(y, X).fit()

# Now, let's use the model for prediction
# Example input data for prediction
new_data = {'Distance driven (in miles)': 50000, 'Manufacturing year': 2016, 'Engine_Power': 200}

# Create a DataFrame for the new data with the same columns as X
new_data_df = pd.DataFrame([new_data], columns=X.columns)

# Make the prediction without adding a constant term
predicted_price = model.predict(new_data_df)
print(f'Predicted Price: ${predicted_price.iloc[0]:,.2f}')
```

```
Predicted Price: $nan
```

In [71]:

```
import statsmodels.api as sm
# Repuie coum nomes with the actual column names from your dataset
# Select relevant independent variables

X= df[['Distance driven (in miles)', 'Manufacturing year', 'Engine_Power']]

# Add a constant term to the independent variables
X= sm.add_constant (X)

# Target variable
y = df['Price']

# Print the summary

print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.823
Model:                            OLS   Adj. R-squared:                  0.817
Method:                 Least Squares   F-statistic:                     148.4
Date:                Thu, 14 Dec 2023   Prob (F-statistic):           6.30e-36
Time:                        03:47:43   Log-Likelihood:                -957.66
No. Observations:                 100   AIC:                             1923.
Df Residuals:                      96   BIC:                             1934.
Df Model:                           3
Covariance Type:            nonrobust
================================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                      -6.251e+06   8.35e+05     -7.490      0.000   -7.91e+06   -4.59e+06
Distance driven (in miles)    -0.1002      0.019     -5.307      0.000      -0.138      -0.063
Manufacturing year          3105.1656    413.207      7.515      0.000    2284.957    3925.374
Engine_Power                  28.7089      7.369      3.896      0.000      14.082      43.336
==============================================================================
Omnibus:                       12.954   Durbin-Watson:                   1.631
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               20.953
Skew:                           0.547   Prob(JB):                     2.82e-05
Kurtosis:                       4.958   Cond. No.                     1.24e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.24e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```