The project will require a multiple linear regression model

In [318]:

```
# Importing Packages
!pip install statsmodels
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as stats
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
import statsmodels.formula.api as smf
import statsmodels.api as sm
from sklearn.metrics import r2_score
from sklearn.model_selection import KFold
```

```
Requirement already satisfied: statsmodels in
c:\users\msafw\anaconda3\lib\site-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in
c:\users\msafw\anaconda3\lib\site-packages (from statsmodels) (1.24.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in
c:\users\msafw\anaconda3\lib\site-packages (from statsmodels) (1.15.2)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in
c:\users\msafw\anaconda3\lib\site-packages (from statsmodels) (2.2.3)
Requirement already satisfied: patsy>=0.5.6 in
c:\users\msafw\anaconda3\lib\site-packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in
c:\users\msafw\anaconda3\lib\site-packages (from statsmodels) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\msafw\anaconda3\lib\site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\msafw\anaconda3\lib\site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\msafw\anaconda3\lib\site-packages (from
pandas!=2.1.0,>=1.4->statsmodels) (2023.3)
Requirement already satisfied: six>=1.5 in
c:\users\msafw\anaconda3\lib\site-packages (from
python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.17.0)
```

In [319]:

```
car = pd.read_csv("usedcars.csv")
car.head(5)
```

Out[319]:

Price	Age	Odometer	Inspection	ABS	Sunroof	Annualkms

0	7.30	73	10.0	12	1	1	1.643836
1	3.85	115	30.0	20	1	0	3.130435
2	2.95	127	43.0	6	0	1	4.062992
3	4.80	104	54.0	25	1	1	6.230769
4	6.20	86	57.0	23	0	0	7.953488

In [320]:

```
car.var() # to check for scalability of the variables
```

Out[320]:

```
Price            1.541861
Age            362.853939
Odometer      2009.675408
Inspection      48.952391
ABS              0.215136
Sunroof          0.189336
Annualkms       21.515985
dtype: float64
```

In [321]:

```
car.shape
```

Out[321]:

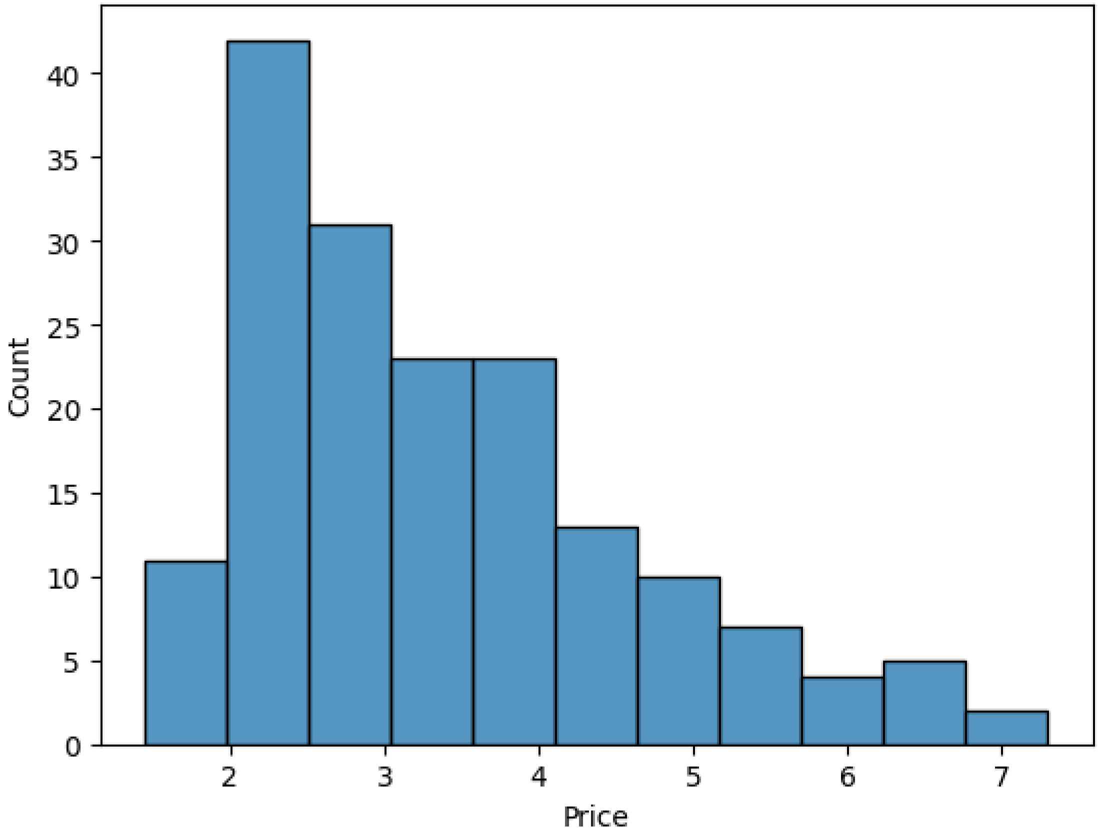
```
(171, 7)
```

In [322]:

```
sns.histplot(car['Price'])
```

Out[322]:

```
<Axes: xlabel='Price', ylabel='Count'>
```

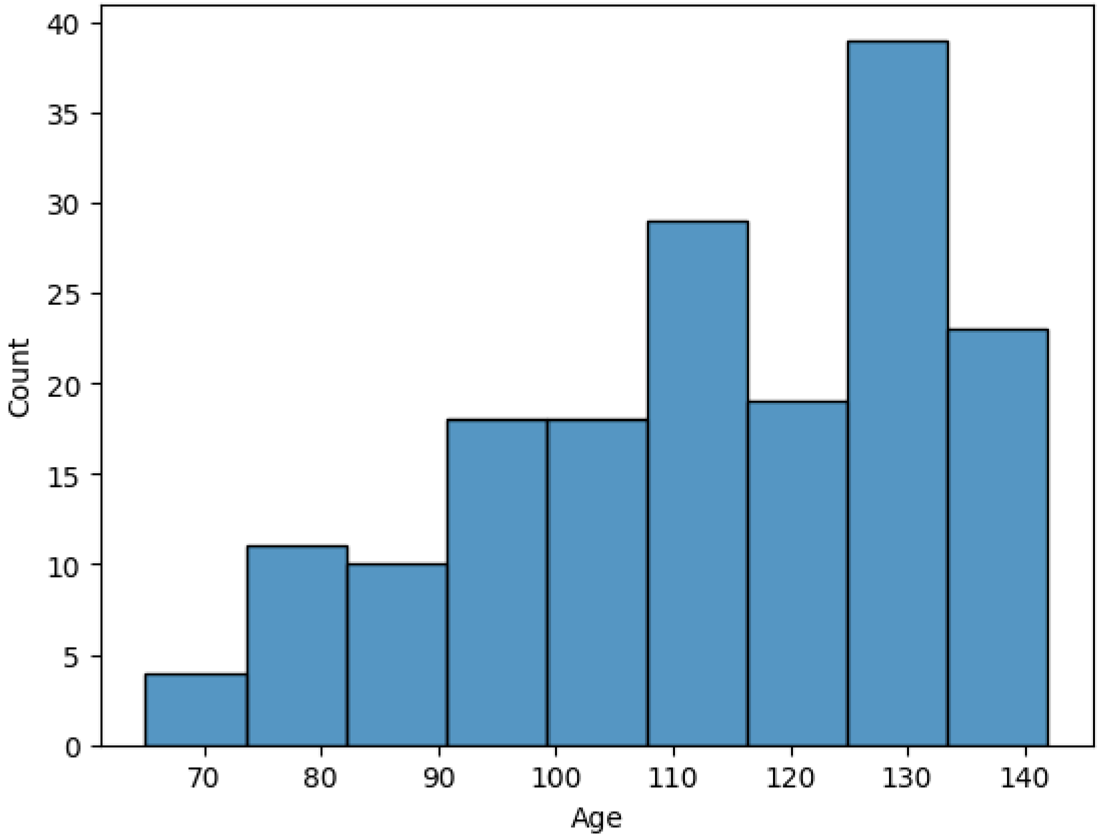


In [323]:

```
sns.histplot(car['Age'])
```

Out[323]:

<Axes: xlabel='Age', ylabel='Count'>

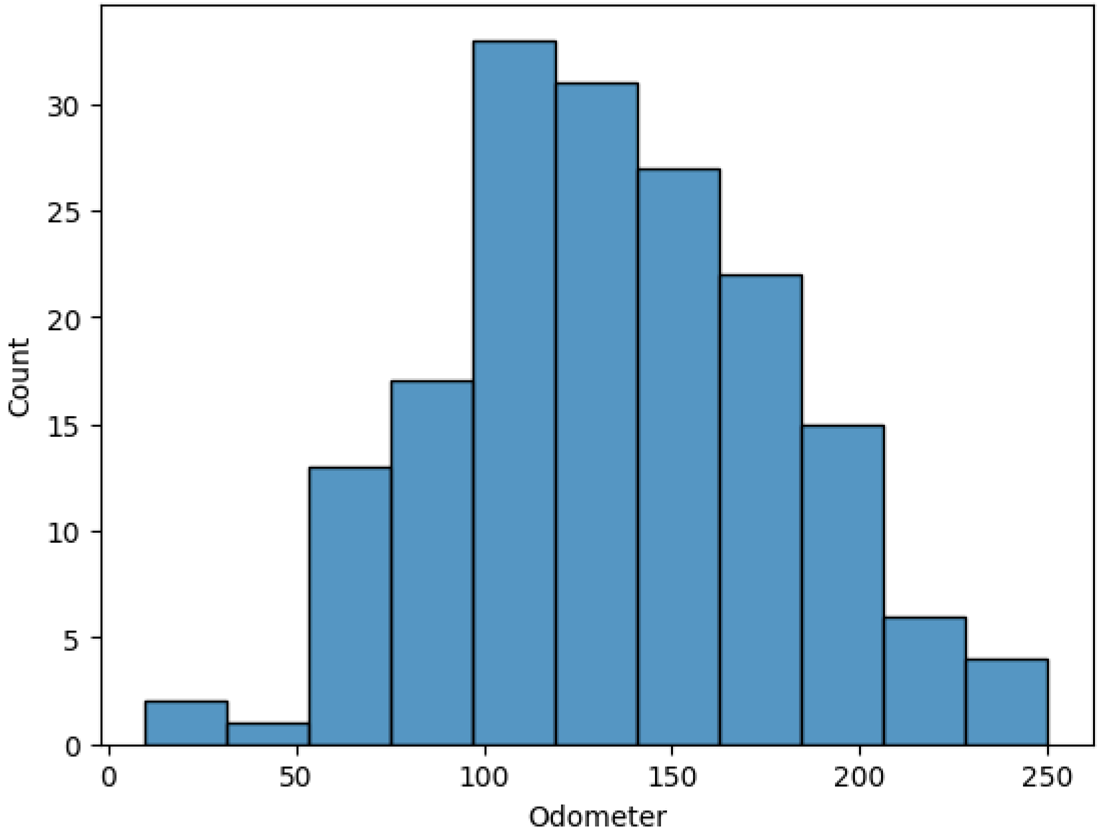


In [324]:

```
sns.histplot(car['Odometer'])
```

Out[324]:

```
<Axes: xlabel='Odometer', ylabel='Count'>
```

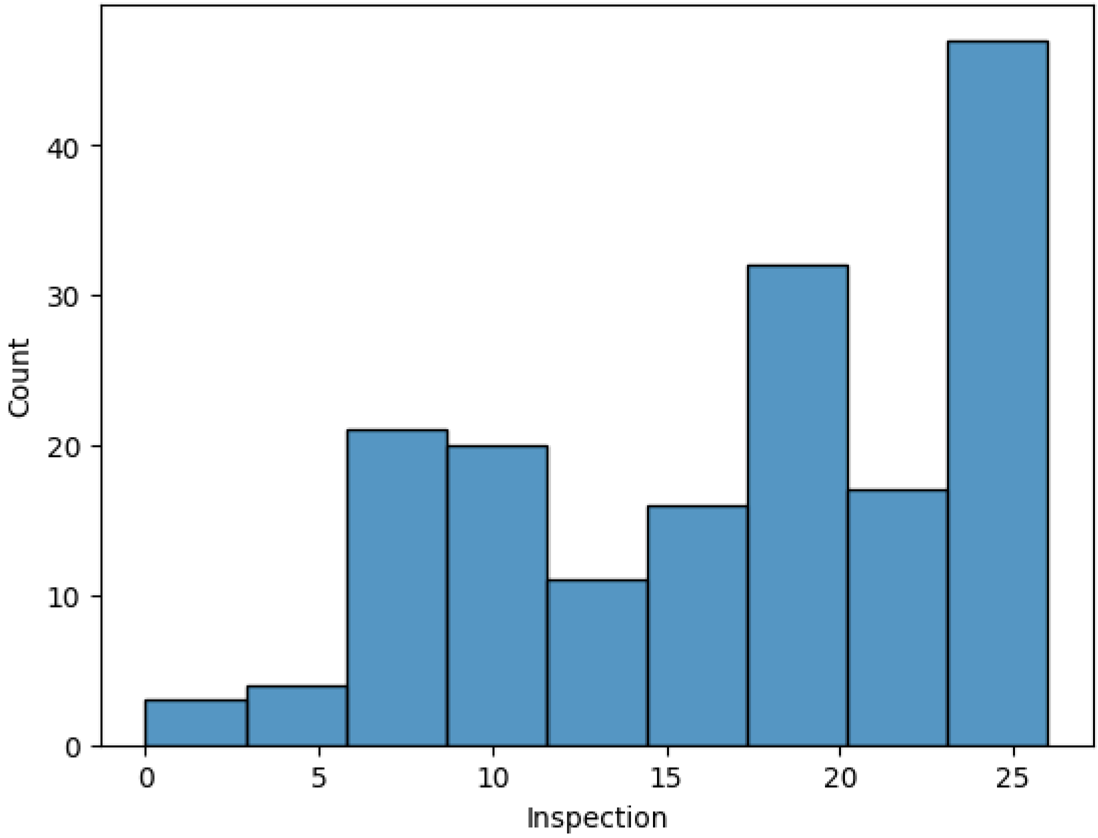


In [325]:

```
sns.histplot(car['Inspection'])
```

Out[325]:

```
<Axes: xlabel='Inspection', ylabel='Count'>
```

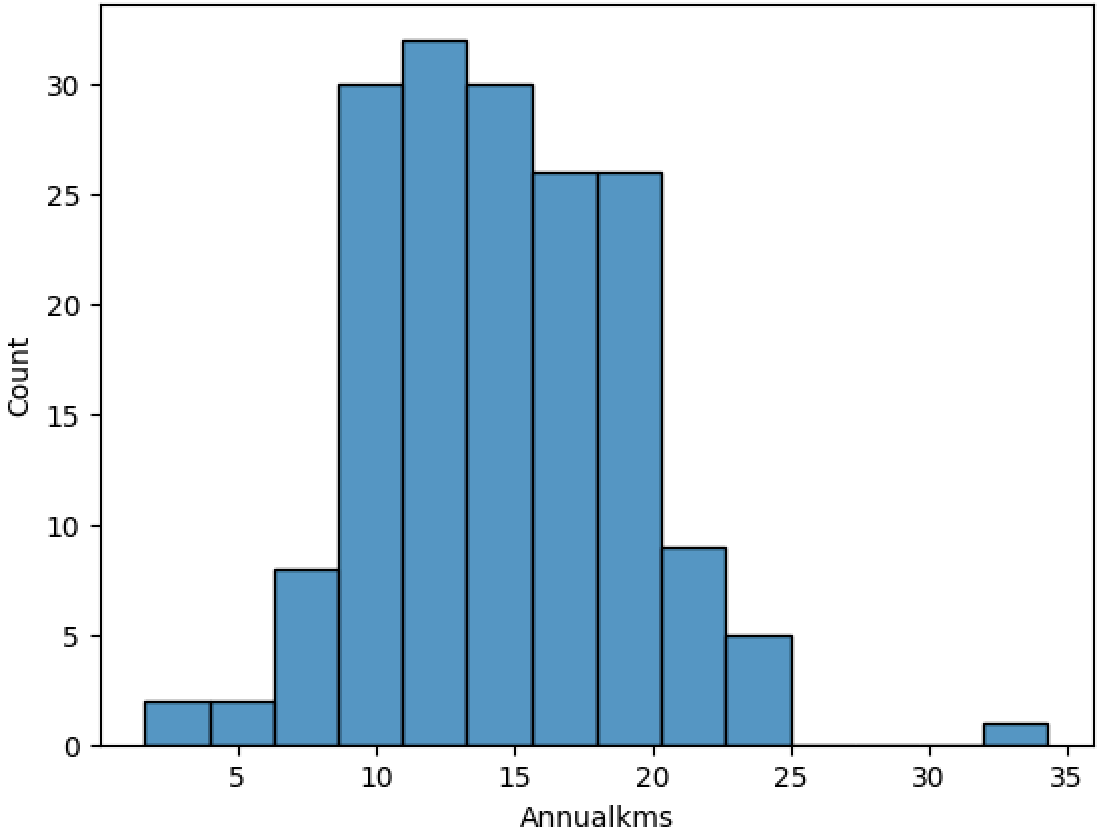


In [326]:

```
sns.histplot(car['Annualkms'])
```

Out[326]:

```
<Axes: xlabel='Annualkms', ylabel='Count'>
```



In [327]:

```
# Linear Regression without transformation of the variables
model = smf.ols('Price ~ Age + Odometer + Inspection + ABS + Sunroof +
Annualkms', data=car).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  Price   R-squared:
0.641
Model:                            OLS   Adj. R-squared:
0.628
Method:                 Least Squares   F-statistic:
48.78
Date:                Sun, 04 May 2025   Prob (F-statistic):
4.90e-34
Time:                        19:52:26   Log-Likelihood:
-191.59
No. Observations:                 171   AIC:
397.2
Df Residuals:                     164   BIC:
419.2
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
=
```

```
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept     11.4162      1.044     10.931      0.000       9.354
13.478
Age           -0.0567      0.009     -6.066      0.000      -0.075
-0.038
Odometer       0.0053      0.007      0.735      0.464      -0.009
0.020
Inspection    -0.0074      0.008     -0.878      0.381      -0.024
0.009
ABS           -0.2955      0.128     -2.314      0.022      -0.548
-0.043
Sunroof        0.0358      0.136      0.263      0.793      -0.233
0.305
Annualkms     -0.1399      0.065     -2.142      0.034      -0.269
-0.011
==============================================================================
=
Omnibus:                        4.617   Durbin-Watson:
2.324
Prob(Omnibus):                  0.099   Jarque-Bera (JB):
6.273
Skew:                          -0.006   Prob(JB):
0.0434
Kurtosis:                       3.938   Cond. No.
3.29e+03
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 3.29e+03. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

In [328]:

```
car['Age'] = np.log(car['Age'])
car['Inspection'] = np.log(car['Inspection'] +1)
car['Price'] = np.log(car['Price'])
car['Odometer'] = np.log(car['Odometer'])
car['Annualkms'] = np.log(car['Annualkms'])
# there is 1 value in the inspection of 0 which log does not work on so we
add the +1
```

In [329]:

```
car.corr()
```

Out[329]:

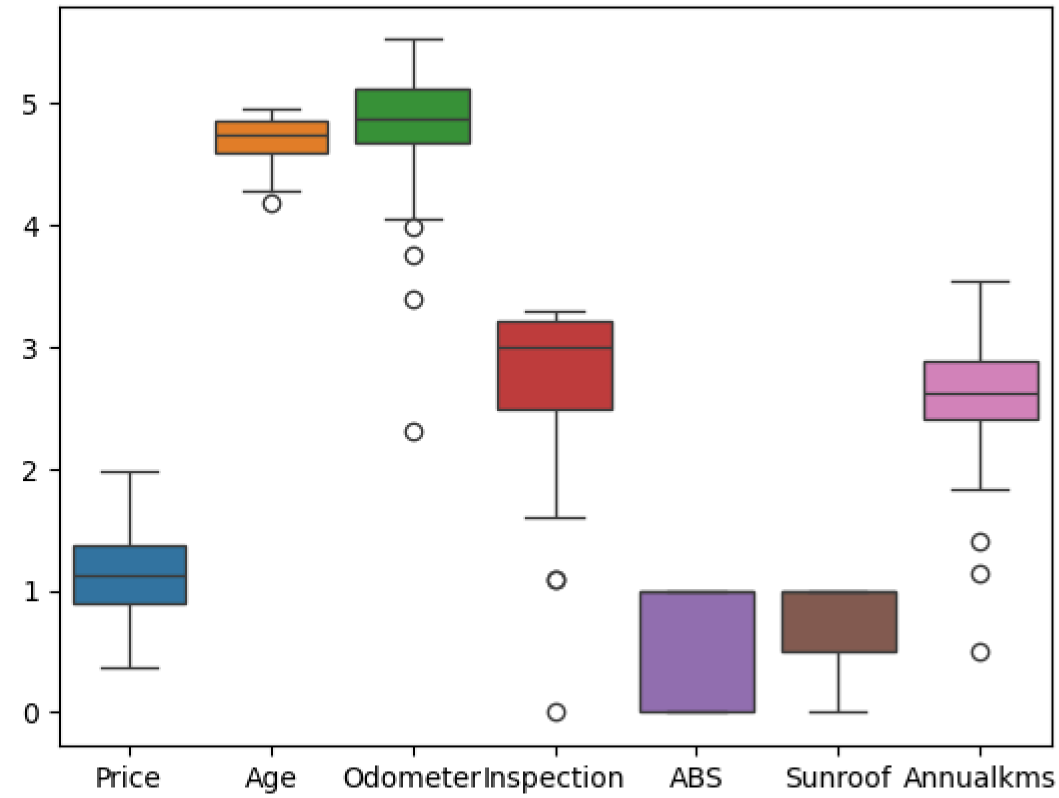
	Price	Age	Odometer	Inspection	ABS	Sunroof	Annualkms
Price	1.000000	-0.707616	-0.568057	-0.044606	0.007920	-0.103568	-0.277357
Age	-0.707616	1.000000	0.395550	-0.010695	-0.047623	0.098166	-0.050854
Odometer	-0.568057	0.395550	1.000000	0.007959	-0.025866	0.123071	0.897141
Inspection	-0.044606	-0.010695	0.007959	1.000000	-0.111970	0.054090	0.013799
ABS	0.007920	-0.047623	-0.025866	-0.111970	1.000000	0.048747	-0.005221
Sunroof	-0.103568	0.098166	0.123071	0.054090	0.048747	1.000000	0.086611
Annualkms	-0.277357	-0.050854	0.897141	0.013799	-0.005221	0.086611	1.000000

In [330]:

```
sns.boxplot(data = car)
```

Out[330]:

```
<Axes: >
```



In [331]:

```
car.head()
```

Out[331]:

	Price	Age	Odometer	Inspection	ABS	Sunroof	Annualkms
0	1.987874	4.290459	2.302585	2.564949	1	1	0.497032
1	1.348073	4.744932	3.401197	3.044522	1	0	1.141172
2	1.081805	4.844187	3.761200	1.945910	0	1	1.401920
3	1.568616	4.644391	3.988984	3.258097	1	1	1.829500

4	1.824549	4.454347	4.043051	3.178054	0	0	2.073611

In [332]:

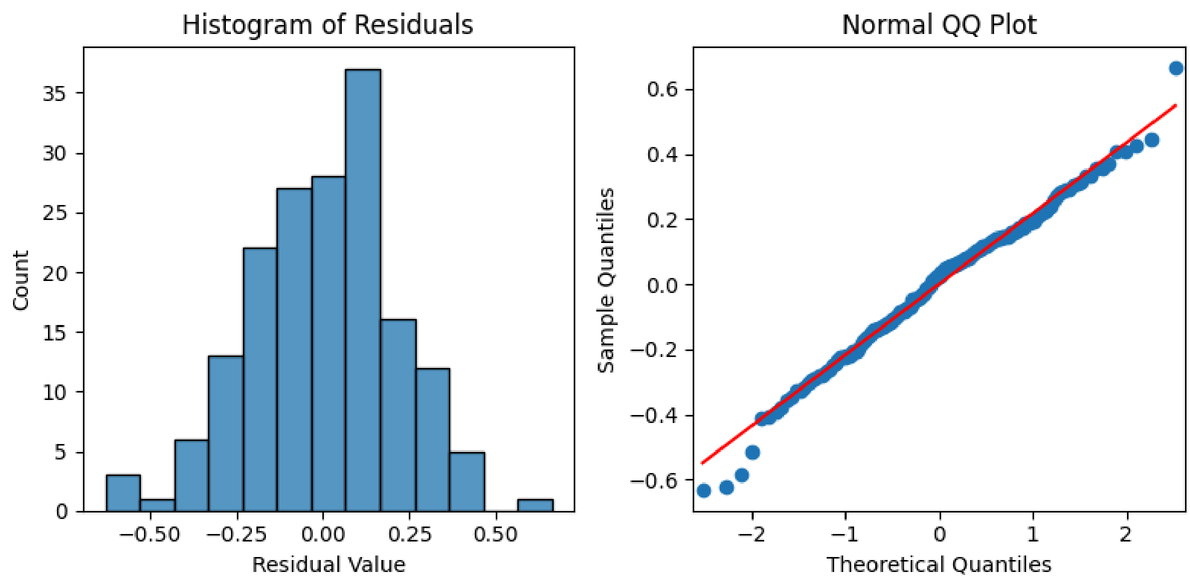
```
# Linear Regression transformations
model_transformation = smf.ols('Price ~ Age + Inspection + C(ABS) +
C(Sunroof) + Odometer', data=car).fit()
# C is statsmodel way of using a hot-encoding method in the model
print(model_transformation.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  Price   R-squared:
0.603
Model:                            OLS   Adj. R-squared:
0.591
Method:                 Least Squares   F-statistic:
50.04
Date:                Sun, 04 May 2025   Prob (F-statistic):
2.35e-31
Time:                        19:52:27   Log-Likelihood:
18.533
No. Observations:                 171   AIC:
-25.07
Df Residuals:                     165   BIC:
-6.215
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
======
                      coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------
Intercept           7.8754      0.459     17.156      0.000       6.969
8.782
C(ABS)[T.1]        -0.0254      0.037     -0.687      0.493      -0.098
0.047
C(Sunroof)[T.1]    -0.0006      0.039     -0.014      0.989      -0.078
0.077
Age                -1.1030      0.103    -10.728      0.000      -1.306
-0.900
Inspection         -0.0337      0.032     -1.047      0.297      -0.097
0.030
Odometer           -0.2895      0.046     -6.355      0.000      -0.379
-0.200
==============================================================================
=
Omnibus:                        2.695   Durbin-Watson:
2.302
```

```
Prob(Omnibus):                  0.260   Jarque-Bera (JB):
2.245
Skew:                          -0.243   Prob(JB):
0.326
Kurtosis:                       3.280   Cond. No.
206.
==============================================================================
=

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

In [333]:

```
# Testing for Normality
# Plotting of Residuals
residuals = model_transformation.resid

# Histogram of Residuals
fig, axes = plt.subplots(1, 2, figsize = (8,4))
sns.histplot(residuals, ax=axes[0])
axes[0].set_xlabel("Residual Value")
axes[0].set_title("Histogram of Residuals")

# Q-Q plot of the residuals.
sm.qqplot(residuals, line='s',ax = axes[1])

axes[1].set_title("Normal QQ Plot")

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



In [334]:

```
# Testing independent observations
'''
Durbin-Watson: 2.3 is good score
The residuals are independent.
'''
```

Out[334]:

```
'\nDurbin-Watson: 2.3 is good score\nThe residuals are independent.\n'
```

In [335]:

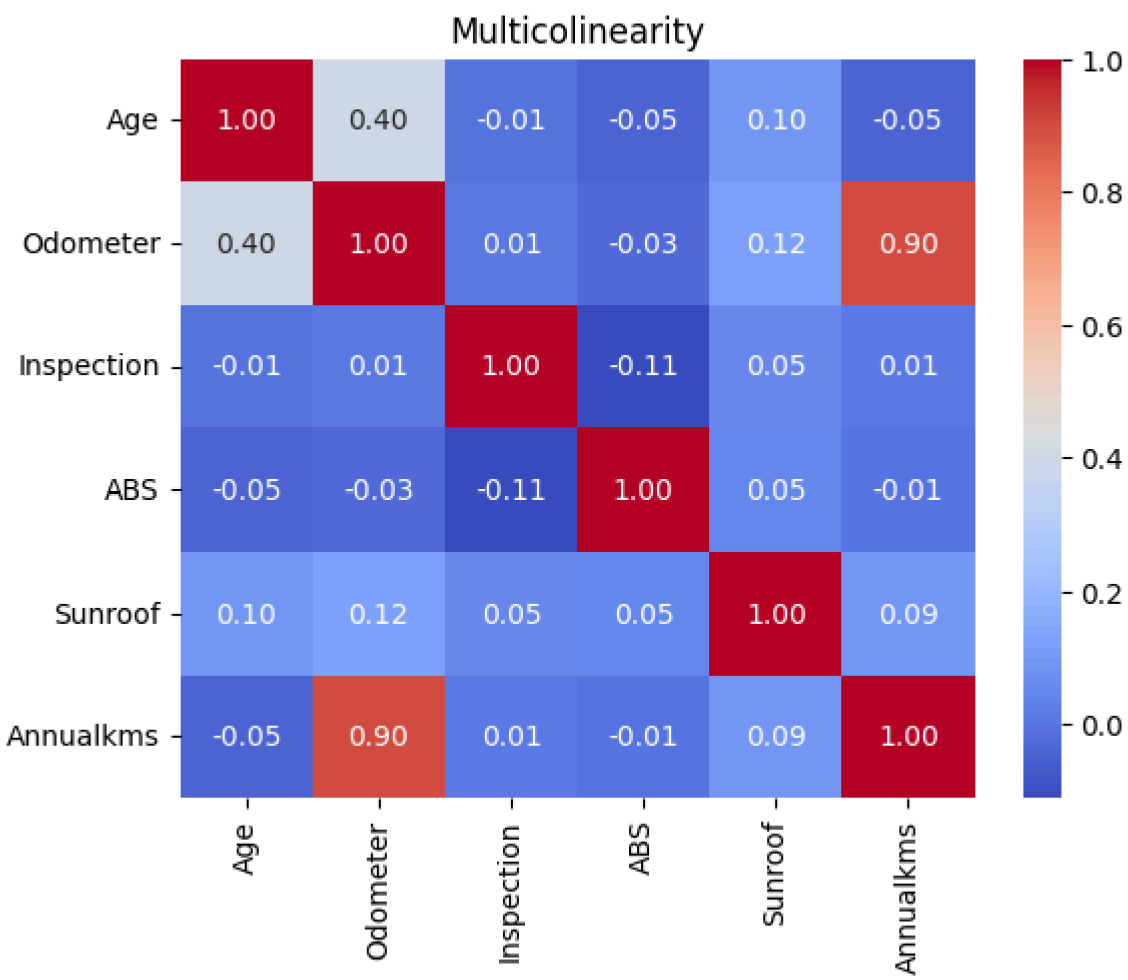
```
y = car['Price']
x = car.drop(columns='Price')
```

In [336]:

```
fig = sns.heatmap(x.corr(), annot=True, cmap="coolwarm", fmt=".2f")
fig.set_title("Multicolinearity")
```

Out[336]:

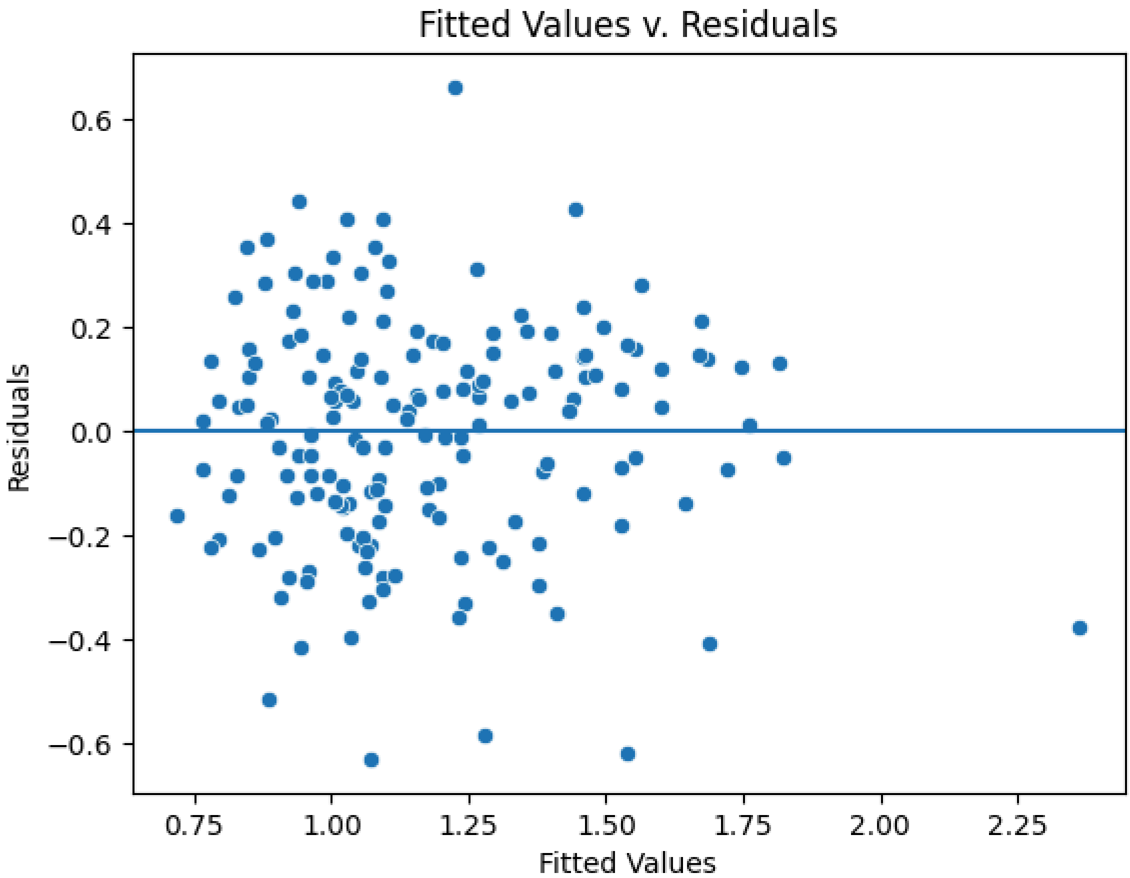
```
Text(0.5, 1.0, 'Multicolinearity')
```



In [337]:

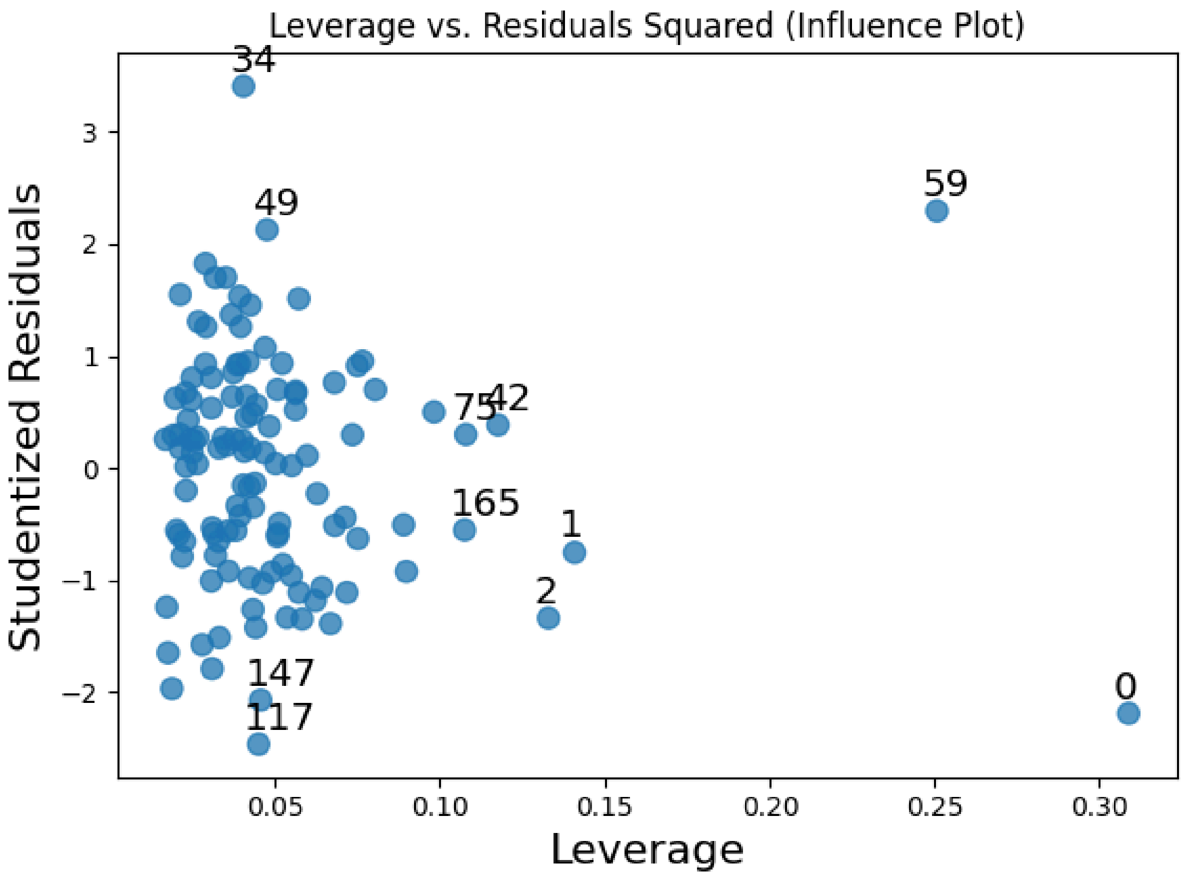
```
# Checking for Homoscedasticity
```

```
fig = sns.scatterplot(x = model_transformation.fittedvalues, y =
model_transformation.resid)


fig.set_xlabel("Fitted Values")

fig.set_ylabel("Residuals")

fig.set_title("Fitted Values v. Residuals")
fig.axhline(0)
plt.show()
```



In [346]:

```
sm.graphics.influence_plot(model_transformation, criterion="cooks", size=8)
plt.title("Leverage vs. Residuals Squared (Influence Plot)")
plt.tight_layout()
plt.show()
```



In [339]:

```
rows_to_drop_of_car = [24,62,78] # Delete extreme outliers
car = car.drop(rows_to_drop_of_car, axis=0)
car.shape
```

Out[339]:

```
(168, 7)
```

In [340]:

```
# Linear Regression transformation of the variable age and inspection
model_transformation = smf.ols('Price ~ Age + Inspection + C(ABS) +
C(Sunroof) + Annualkms', data=car).fit()
# C is statsmodel way of using a hot-encoding method in the model
print(model_transformation.summary())
```

```
                            OLS Regression Results
==============================================================================
=
Dep. Variable:                  Price   R-squared:
0.645
Model:                            OLS   Adj. R-squared:
0.634
Method:                 Least Squares   F-statistic:
58.96
Date:                Sun, 04 May 2025   Prob (F-statistic):
1.00e-34
```

```
Time:                        19:52:30   Log-Likelihood:
29.751
No. Observations:                 168   AIC:
-47.50
Df Residuals:                     162   BIC:
-28.76
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
======
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------
Intercept          8.7678      0.451     19.420      0.000       7.876
9.659
C(ABS)[T.1]       -0.0282      0.035     -0.812      0.418      -0.097
0.040
C(Sunroof)[T.1]    0.0162      0.037      0.438      0.662      -0.057
0.089
Age               -1.4256      0.089    -15.940      0.000      -1.602
-1.249
Inspection        -0.0323      0.030     -1.070      0.286      -0.092
0.027
Annualkms         -0.2974      0.043     -6.977      0.000      -0.382
-0.213
==============================================================================
=
Omnibus:                        0.013   Durbin-Watson:
2.123
Prob(Omnibus):                  0.994   Jarque-Bera (JB):
0.110
Skew:                           0.004   Prob(JB):
0.946
Kurtosis:                       2.875   Cond. No.
180.
==============================================================================
=

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

In [341]:

```
new_car = pd.DataFrame({
    'Age': np.log(124),
    'Annualkms': np.log(12),
    'Inspection':np.log(9),
    'Sunroof': [1],
    'ABS': [0]
})

predicted_price = model_transformation.predict(new_car)
```

```
print(f"Predicted price: {np.exp(predicted_price[0]):.3f} euros")
```

```
Predicted price: 3.012 euros
```

In [342]:

```
new_car = pd.DataFrame({
    'Age': np.log(148),
    'Annualkms': np.log(40),
    'Inspection':np.log(9),
    'Sunroof': [1],
    'ABS': [0]
})

predicted_price = model_transformation.predict(new_car)
print(f"Predicted price: {np.exp(predicted_price[0]):.3f} euros")
```

```
Predicted price: 1.636 euros
```

In [343]:

```
new_car = pd.DataFrame({
    'Age': np.log(148),
    'Annualkms': np.log(40),
    'Inspection':np.log(9),
    'Sunroof': [1],
    'ABS': [1]
})

predicted_price = model_transformation.predict(new_car)
print(f"Predicted price: {np.exp(predicted_price[0]):.3f} euros")
```

```
Predicted price: 1.590 euros
```

In [344]:

```
# Train-test car database

train_data, test_data = train_test_split(car, test_size=0.2, random_state=42)

model_transformation = smf.ols('Price ~ Age + Inspection + C(ABS) +
C(Sunroof) + Annualkms', data=train_data).fit()
# you would normally fit and transform the training data but log must be
applied for both fitting and transforming, as we want the same scale for
both

y_pred = model_transformation.predict(test_data)

r2_test = r2_score(test_data['Price'], y_pred)

print("Train R-squared:", model_transformation.rsquared)
print("Test R-squared:", r2_test)
```

```
Train R-squared: 0.6550245493686415
```

```
Test R-squared: 0.5911407009740445
```

In [345]:

```
# Cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
cv = []

for train_index, val_index in kf.split(train_data):
    fold_train = train_data.iloc[train_index]  # for trainning
    fold_validate = train_data.iloc[val_index]      # for validating

    model_transformation = smf.ols('Price ~ Age + Inspection + C(ABS) +
C(Sunroof) + Annualkms', data=fold_train).fit()
    # It did take a while to understand that looping into fold-train than
actual train data, using k-fold
    y_val_pred = model_transformation.predict(fold_validate)
    # You do predict the scaled data by log for the validating
    r2 = r2_score(fold_validate['Price'], y_val_pred)
    cv.append(r2)

print("Cross-validated R-squareds result:", cv)
print(np.quantile(cv, [0.025, 0.975]))
print("Average CV R-squared result:", np.mean(cv))
```

```
Cross-validated R-squareds result: [0.8511238289233365, 0.7110102955455111,
0.5661451178109761, 0.6207352257491261, 0.4161020626098847,
0.33504571690299334, 0.2984171838379017, 0.6914932462306971,
0.43347470692278967, 0.7569065789211618]
[0.3066586  0.82992495]
Average CV R-squared result: 0.5680453963454377
```