Predicting Fuel Economy: Regression based Models

|| Building a Linear Regression Model to Predict a Vehicle's Fuel Efficiency ||

The Situation:

Suppose we've been hired as a **Product Data Scientist** for Consumer Reports Magazine, a trusted source for information on Consumer Products and Services.

Fuel Economy is a Major Factor in the Cost of owning a Car, so they are Studying the Automobile Characteristics that Impact Fuel Economy for an upcoming Article.

We have been asked to **Build a Regression Model to Predict Fuel Economy based on Characteristics like the Car's Weight, Model Year, Accelaration, and more.**

The Objectives:

1. Preparing and Exploring the Data
2. Splitting the data and Build a Multiple Regression Model
3. Evaluating Model Test Performance and Interpret the Model
4. Building a Ridge Regression Model and Compare the Results.

Objective 1: Preparing and Exploring the Data

Our First Objective is to Prepare the Data for Modelling, Explore the Target ('mpg') and other Features in the Dataset, and Fix any Issues we encounter.

Specific Tasks:

- Reading the auto-mpg.csv dataset, and check datatypes and columns for missing or unusual values.
- Convert 'origin' to a Categorical Feature.
- Calculating Summary Statistics for each of the numeric columns in the dataset including min, max and mean, then build a Histogram of the Target Variable ('mpg').
- Explore Relationships between the Features and the 'mpg' column, and use Scatterplots and build a Correlation Heatmap. Which column is most strongly Correlated with 'mpg'?

In [32]:
```python
# Importing the Required Libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [33]:
```python
# First, make sure your Google Drive is mounted in Colab:
# from google.colab import drive
# drive.mount('/content/drive')
```

In [34]:
```python
# churn_cust_info = pd.read_excel("Bank_Churn_Messy.xlsx")
#mpg = pd.read_csv("/content/drive/MyDrive/Data Analytics & BI Career Path/Batch 2/Python/Predicting Fuel Economy using Regression based Models/auto

#mpg = pd.read_csv("auto-mpg.csv")
mpg.head()
```

Out[34]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	1	ford torino

In [35]:
```python
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    object
 8   car name      398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

In [36]:
```python
mpg["origin"] = mpg["origin"].astype("object")
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    object
 8   car name      398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

In [37]:
```python
mpg["horsepower"].unique()
```

Out[37]:
```
array([130.        , 165.        , 150.        , 140.        ,
       198.        , 220.        , 215.        , 225.        ,
       190.        , 170.        , 160.        ,  95.        ,
        97.        ,  85.        ,  88.        ,  46.        ,
        87.        ,  90.        , 113.        , 200.        ,
       210.        , 193.        , 104.46938776, 100.        ,
       105.        , 175.        , 153.        , 180.        ,
       110.        ,  72.        ,  86.        ,  70.        ,
        76.        ,  65.        ,  69.        ,  60.        ,
        80.        ,  54.        , 208.        , 155.        ,
       112.        ,  92.        , 145.        , 137.        ,
       158.        , 167.        ,  94.        , 107.        ,
       230.        ,  49.        ,  75.        ,  91.        ,
       122.        ,  67.        ,  83.        ,  78.        ,
        52.        ,  61.        ,  93.        , 148.        ,
       129.        ,  96.        ,  71.        ,  98.        ,
       115.        ,  53.        ,  81.        ,  79.        ,
       120.        , 152.        , 102.        , 108.        ,
        68.        ,  58.        , 149.        ,  89.        ,
        63.        ,  48.        ,  66.        , 139.        ,
       103.        , 125.        , 133.        , 138.        ,
       135.        , 142.        ,  77.        ,  62.        ,
       132.        ,  84.        ,  64.        ,  74.        ,
       116.        ,  82.        ])
```

In [38]:
```python
mpg.query("horsepower == '?'")
```

Out[38]:

mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name

In [39]:
```python
mpg["horsepower"] = pd.to_numeric(mpg["horsepower"], errors = "coerce")

mpg.query("horsepower == '?'")
```

Out[39]:

mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name

In [40]:
```python
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    object
 8   car name      398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

In [41]:
```python
mpg["horsepower"] = mpg["horsepower"].fillna(mpg["horsepower"].mean())

mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    object
 8   car name      398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

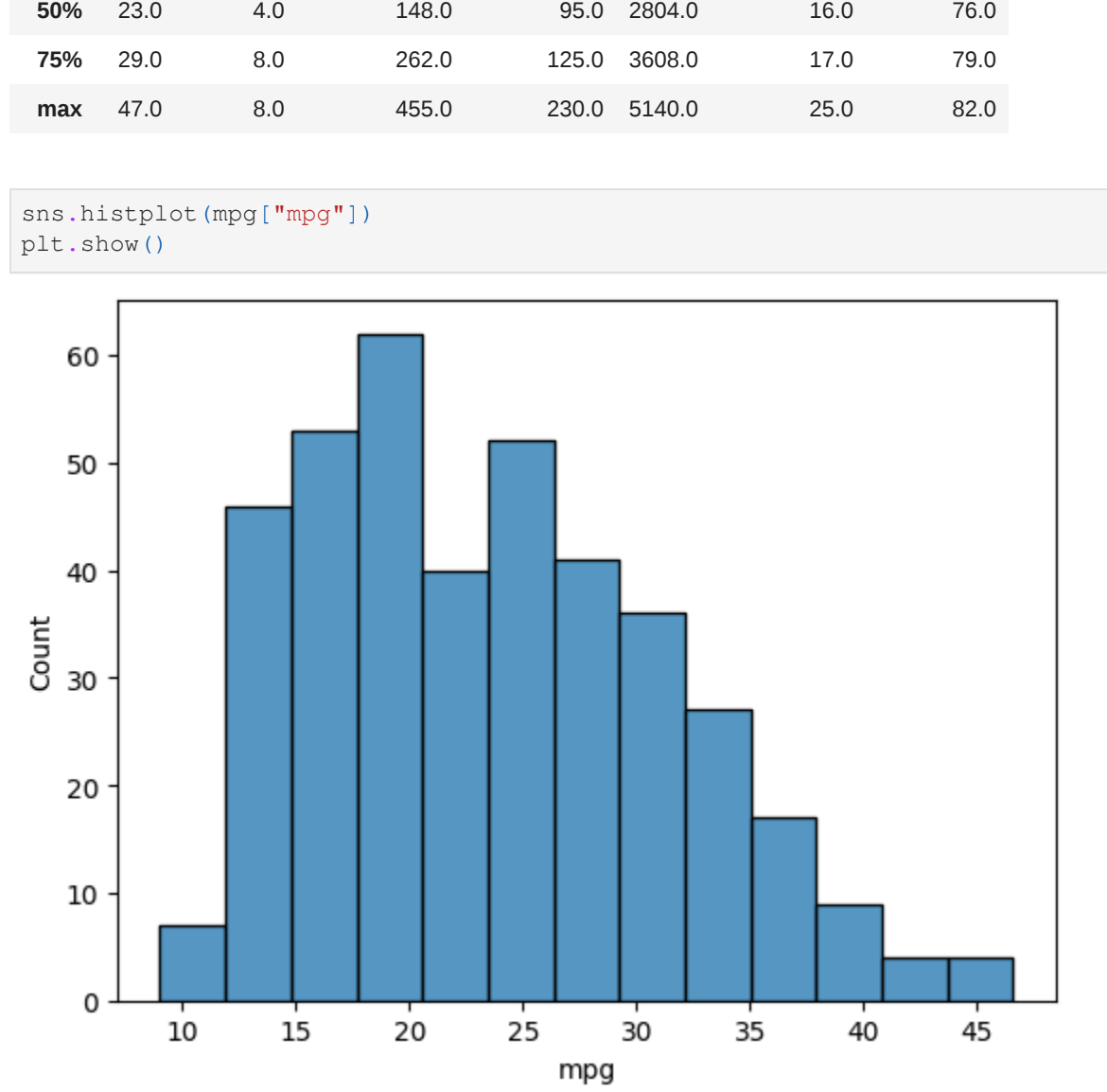
So, we have our QA Process complete. We can now focus on calculating the summary statistics to understand the distribution of the variables.

In [42]:
```python
mpg.describe().round()
```

Out[42]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year
count	398.0	398.0	398.0	398.0	398.0	398.0	398.0
mean	24.0	5.0	193.0	104.0	2970.0	16.0	76.0
std	8.0	2.0	104.0	38.0	847.0	3.0	4.0
min	9.0	3.0	68.0	46.0	1613.0	8.0	70.0
25%	18.0	4.0	104.0	76.0	2224.0	14.0	73.0
50%	23.0	4.0	148.0	95.0	2804.0	16.0	76.0
75%	29.0	8.0	262.0	125.0	3608.0	17.0	79.0
max	47.0	8.0	455.0	230.0	5140.0	25.0	82.0

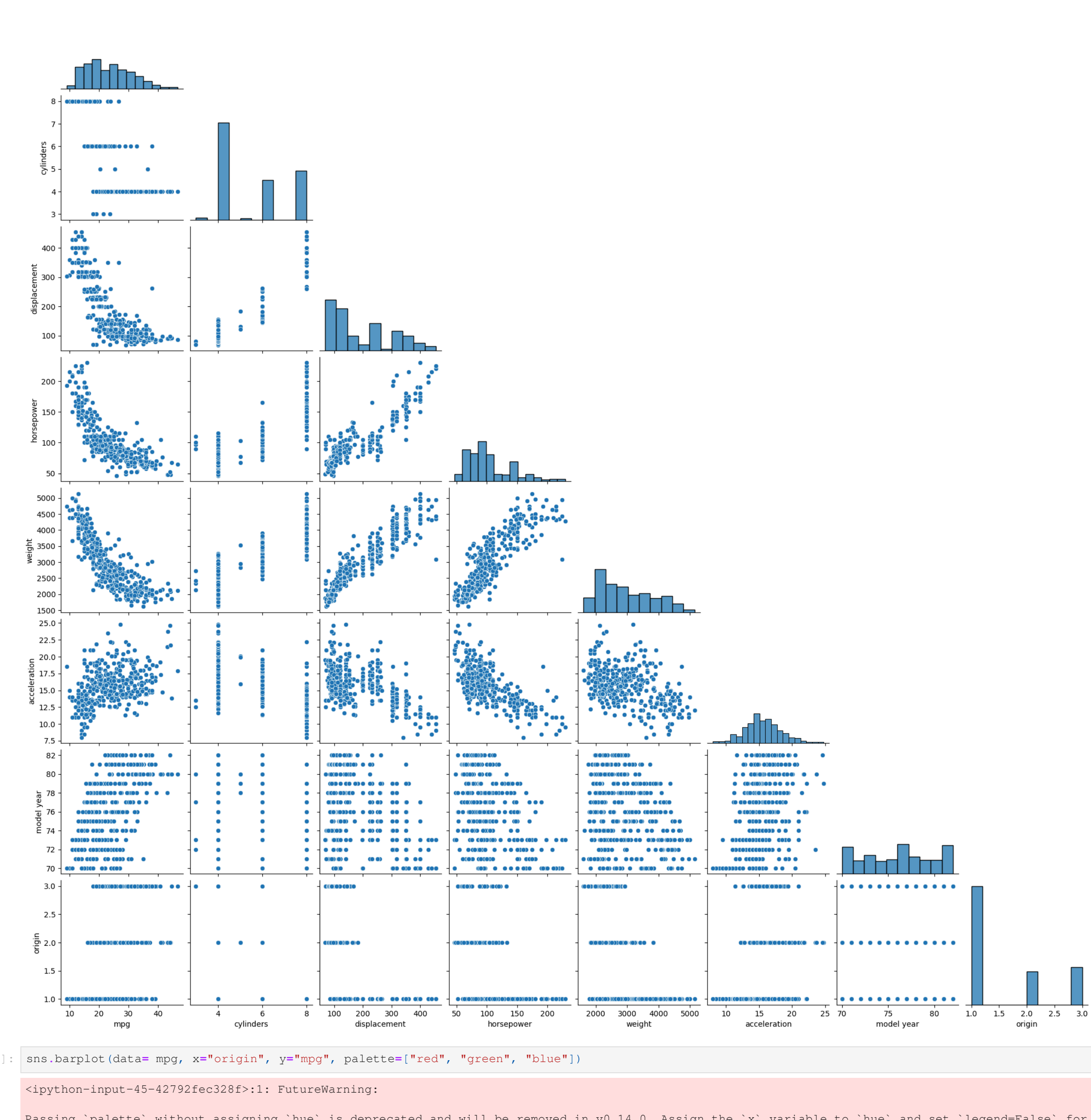
In [43]:
```python
sns.histplot(mpg["mpg"])
plt.show()
```



Explore the Feature-Target Relationshiop:

In [44]:
```python
sns.pairplot(mpg, corner= True)
```

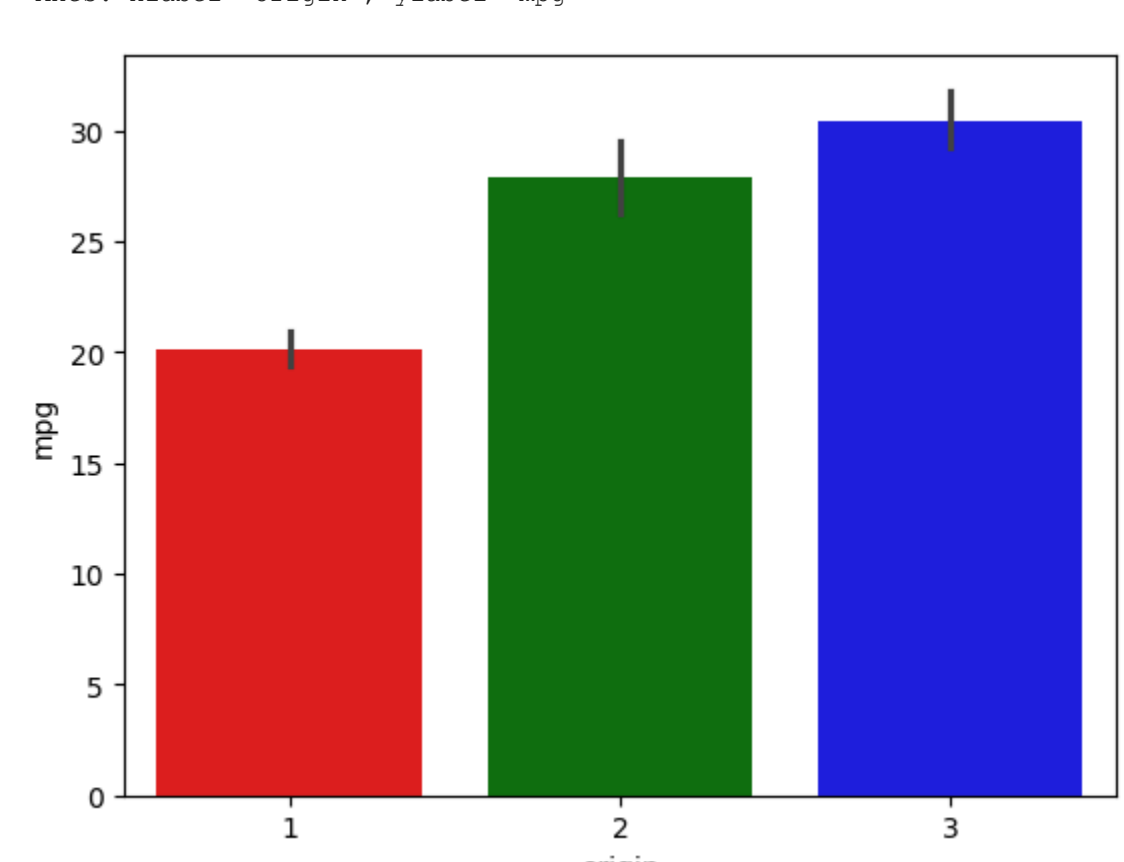
Out[44]:
```
<seaborn.axisgrid.PairGrid at 0x78e6c4537370>
```



In [45]:
```python
sns.barplot(data= mpg, x="origin", y="mpg", palette=["red", "green", "blue"])
```

```
<ipython-input-45-42792fec328f>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data= mpg, x="origin", y="mpg", palette=["red", "green", "blue"])
```

Out[45]:
```
<Axes: xlabel='origin', ylabel='mpg'>
```



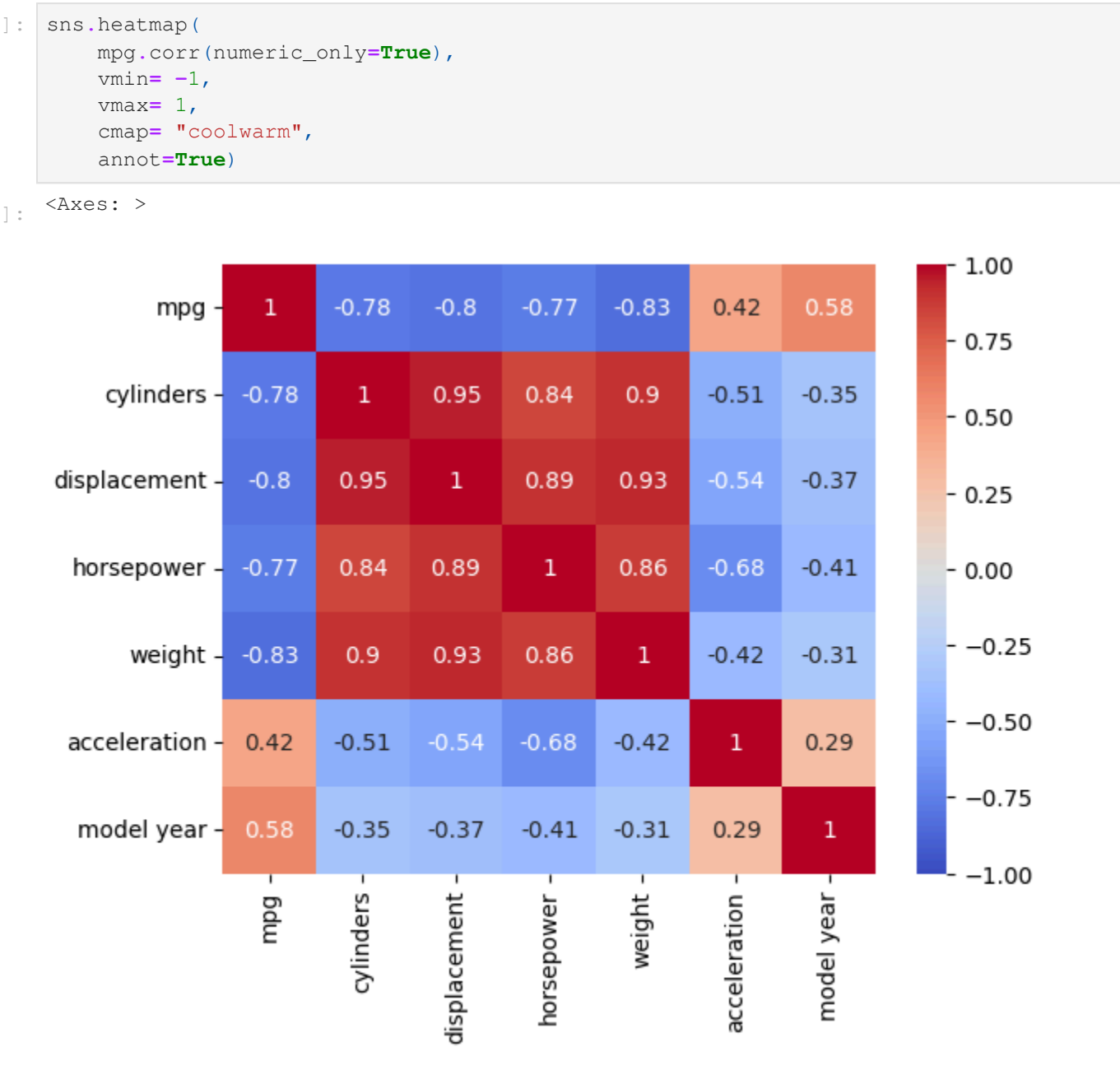
In [46]:
```python
mpg.corr(numeric_only=True).round(2)
```

Out[46]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year
mpg	1.00	-0.78	-0.80	-0.77	-0.83	0.42	0.58
cylinders	-0.78	1.00	0.95	0.84	0.90	-0.51	-0.35
displacement	-0.80	0.95	1.00	0.89	0.93	-0.54	-0.37
horsepower	-0.77	0.84	0.89	1.00	0.86	-0.68	-0.41
weight	-0.83	0.90	0.93	0.86	1.00	-0.42	-0.31
acceleration	0.42	-0.51	-0.54	-0.68	-0.42	1.00	0.29
model year	0.58	-0.35	-0.37	-0.41	-0.31	0.29	1.00

In [47]:
```python
sns.heatmap(
    mpg.corr(numeric_only=True),
    vmin= -1,
    vmax= 1,
    cmap= "coolwarm",
    annot=True)
```

Out[47]:
```
<Axes: >
```



Objective 2: Splitting the Data and Building a Multiple Regression Model

Our Second Objective is to Split the Data into Training and Test Data, then Fit a Multiple Regression Model using the Validation Scheme of our choice.

Also, Performing Feature Engineering and Variable Selection, and Check whether any Assumptions are Violated.

Specific Tasks:

- Splitting the Data into Train and Test, then then set up a Validation Scheme of your choice.
- Fitting a Baseline Regression Model using the Feature with the strongest Correlation to the Target ('mpg').
- Fitting a Multiple Regression Model. Perform any Feature Selection and Feature Engineering necessary, fixing any Violated Assumptions along the way.

In [48]:
```python
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score as r2, mean_squared_error as mae
import statsmodels.api as sm
```

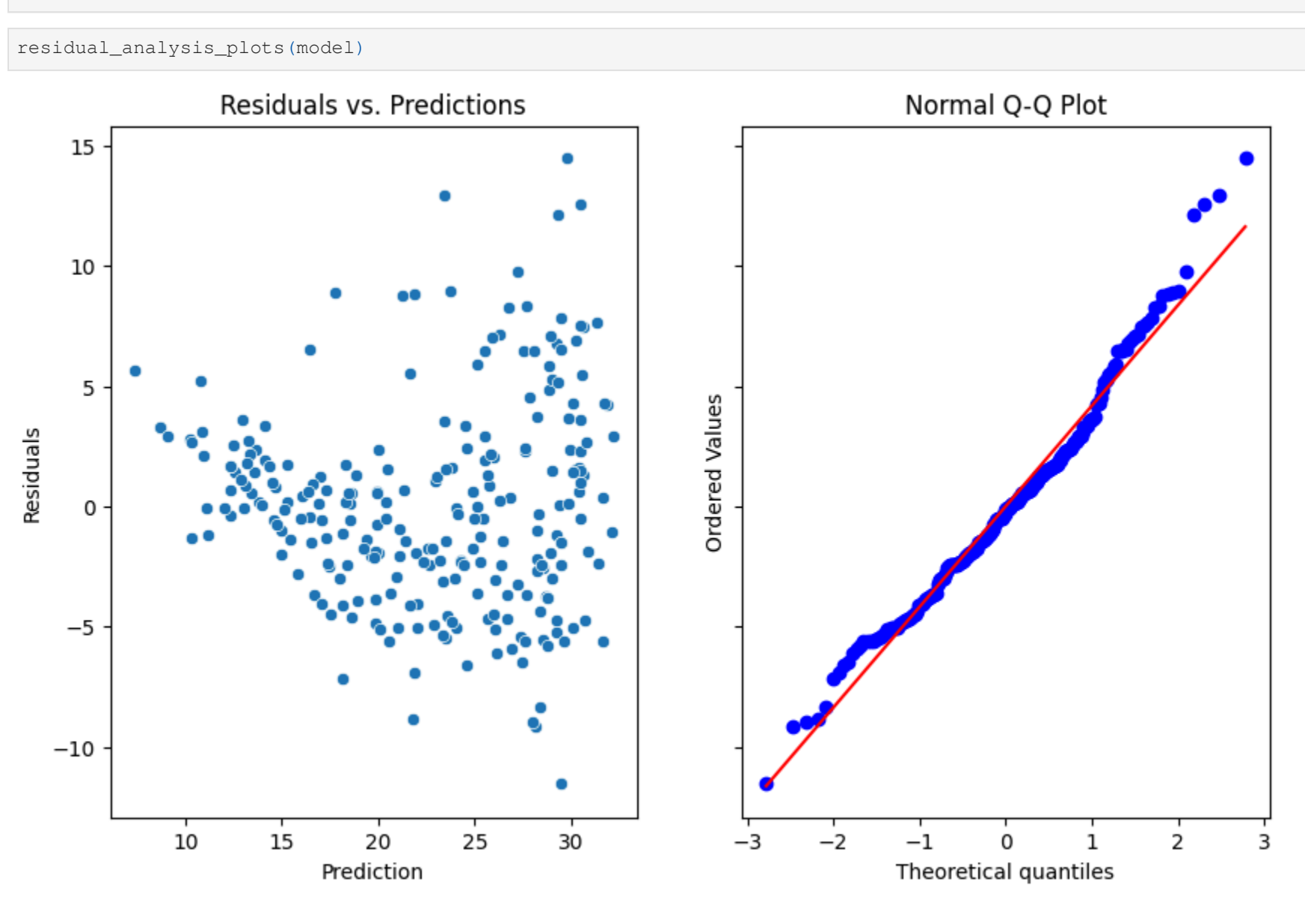
In [49]:
```python
feature = ["weight"]

X = sm.add_constant(mpg[feature])
y = mpg["mpg"]

X, X_test, y, y_test = train_test_split(X, y, test_size=0.2, random_state=2023)
```

In [50]:
```python
kf = KFold(n_splits=5, shuffle=True, random_state=2023)

#Create a list to nstore validation scores for each fold
cv_lm_r2s = []
cv_lm_mae = []

#Loop through each fold in X and y
for train_ind, val_ind in kf.split(X, y):
  #Subset data baed on CV folds
  X_train, y_train = X.iloc[train_ind], y.iloc[train_ind]
  X_val, y_val = X.iloc[val_ind], y.iloc[val_ind]
  #Fit the model on fold's training data
  model = sm.OLS(y_train, X_train).fit()
  #Append Validation score to the list
  cv_lm_r2s.append(r2(y_val, model.predict(X_val),))
  cv_lm_mae.append(mae(y_val, model.predict(X_val)))

print("All Validation R2s: ", [round(x, 3) for x in cv_lm_r2s])
print(f"Cross Val R2s: {round(np.mean(cv_lm_r2s), 3)} +- {round(np.std(cv_lm_r2s), 3)}")

print("All Validation MAEs: ", [round(x, 3) for x in cv_lm_mae])
print(f"Cross Val MAEs: {round(np.mean(cv_lm_mae), 3)} +- {round(np.std(cv_lm_mae),3)}")
```

```
All Validation R2s:  [0.775, 0.7, 0.645, 0.603, 0.662]
Cross Val R2s: 0.677 +- 0.058
All Validation MAEs:  [12.992, 16.408, 20.77, 21.532, 26.694]
Cross Val MAEs: 19.679 +- 4.674
```

In [55]:
```python
def residual_analysis_plots(model):

  import scipy.stats as stats

  predictions = model.predict()
  residuals = model.resid

  fig, ax = plt.subplots(1,2, sharey= "all", figsize=(10,6))

  sns.scatterplot(x= predictions, y= residuals, ax= ax[0])
  ax[0].set_xlabel("Prediction")
  ax[0].set_ylabel("Residuals")
  ax[0].set_title("Residuals vs. Predictions")

  stats.probplot(residuals, dist= "norm", plot= ax[1])
  ax[1].set_title("Normal Q-Q Plot")
```

In [56]:
```python
residual_analysis_plots(model)
```



A curve relationship can be observed which is expected as we saw a curve relationship between weight and mpg. So, fitting a line through a curve pattern will lead to residuals like this. So, we need to add a polynomial model, most likely a squared term.

For this, we need to do some Feature Engineering:

In [69]:
```python
mpg_model = mpg.assign(
    weight2 = mpg["weight"] ** 2
).drop("car name", axis = 1)

mpg_model = pd.get_dummies(mpg_model, drop_first= True, dtype = "int")
mpg_model.head()
```

Out[69]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	weight2	origin_2	origin_3
0	18.0	8	307.0	130.0	3504	12.0	70	12278016	0	0
1	15.0	8	350.0	165.0	3693	11.5	70	13638249	0	0
2	18.0	8	318.0	150.0	3436	11.0	70	11806096	0	0
3	16.0	8	304.0	150.0	3433	12.0	70	11785489	0	0
4	17.0	8	302.0	140.0	3449	10.5	70	11895601	0	0

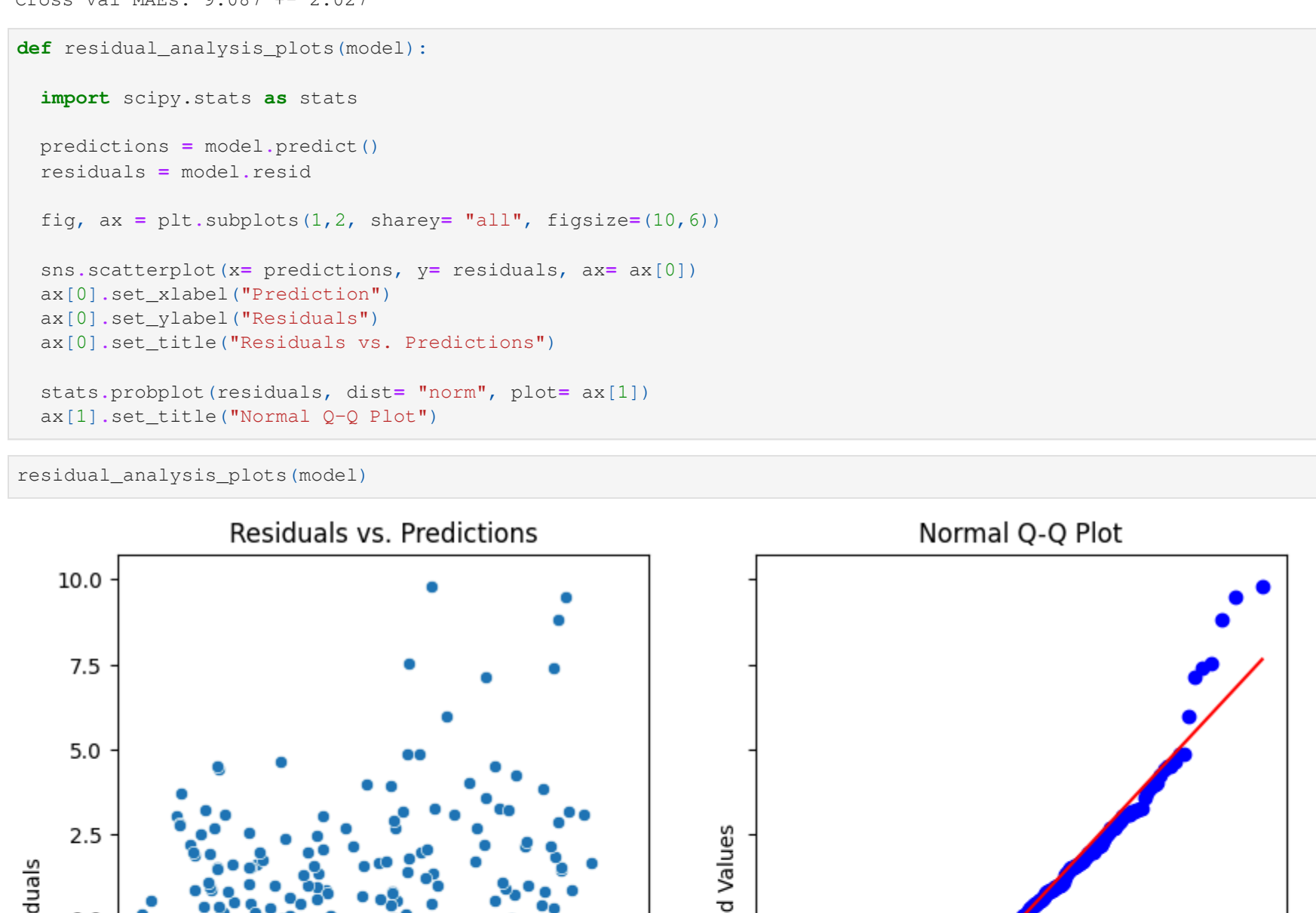
In [70]:
```python
features = [
    "weight",
    "weight2",
    "cylinders",
    "displacement",
    "horsepower",
    "acceleration",
    "model year",
    "origin_2",
    "origin_3"
    ]

X = sm.add_constant(mpg_model[features])
y = mpg_model["mpg"]

X, X_test, y, y_test = train_test_split(X, y, test_size=0.2, random_state=2023)
```

In [71]:
```python
kf = KFold(n_splits=5, shuffle=True, random_state=2023)

#Create a list to nstore validation scores for each fold
cv_lm_r2s = []
cv_lm_mae = []

#Loop through each fold in X and y
for train_ind, val_ind in kf.split(X, y):
  #Subset data baed on CV folds
  X_train, y_train = X.iloc[train_ind], y.iloc[train_ind]
  X_val, y_val = X.iloc[val_ind], y.iloc[val_ind]
  #Fit the model on fold's training data
  model = sm.OLS(y_train, X_train).fit()
  #Append Validation score to the list
  cv_lm_r2s.append(r2(y_val, model.predict(X_val),))
  cv_lm_mae.append(mae(y_val, model.predict(X_val)))

print("All Validation R2s: ", [round(x, 3) for x in cv_lm_r2s])
print(f"Cross Val R2s: {round(np.mean(cv_lm_r2s), 3)} +- {round(np.std(cv_lm_r2s), 3)}")

print("All Validation MAEs: ", [round(x, 3) for x in cv_lm_mae])
print(f"Cross Val MAEs: {round(np.mean(cv_lm_mae), 3)} +- {round(np.std(cv_lm_mae),3)}")
```

```
All Validation R2s:  [0.888, 0.846, 0.838, 0.845, 0.84]
Cross Val R2s: 0.851 +- 0.018
All Validation MAEs:  [6.485, 8.414, 9.459, 8.421, 12.656]
Cross Val MAEs: 9.087 +- 2.027
```

In [72]:
```python
def residual_analysis_plots(model):

  import scipy.stats as stats

  predictions = model.predict()
  residuals = model.resid

  fig, ax = plt.subplots(1,2, sharey= "all", figsize=(10,6))

  sns.scatterplot(x= predictions, y= residuals, ax= ax[0])
  ax[0].set_xlabel("Prediction")
  ax[0].set_ylabel("Residuals")
  ax[0].set_title("Residuals vs. Predictions")

  stats.probplot(residuals, dist= "norm", plot= ax[1])
  ax[1].set_title("Normal Q-Q Plot")
```

In [73]:
```python
residual_analysis_plots(model)
```



No clear pattern in the scatter plot. Although a few outliers can be seen in the Q-Q plot.

In [74]:
```python
model.summary()
```

Out[74]:

OLS Regression Results

Dep. Variable:	mpg	R-squared:	0.866
Model:	OLS	Adj. R-squared:	0.861
Method:	Least Squares	F-statistic:	176.4
Date:	Wed, 08 Jan 2025	Prob (F-statistic):	1.01e-101
Time:	12:27:36	Log-Likelihood:	-619.73
No. Observations:	255	AIC:	1259.
Df Residuals:	245	BIC:	1295.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.3469	5.554	0.062	0.950	-10.593	11.287
weight	-0.0206	0.002	-10.972	0.000	-0.024	-0.017
weight2	2.238e-06	2.82e-07	7.932	0.000	1.68e-06	2.79e-06
cylinders	-0.3086	0.331	-0.931	0.353	-0.961	0.344
displacement	0.0152	0.008	1.818	0.070	-0.001	0.032
horsepower	-0.0237	0.014	-1.651	0.100	-0.052	0.005
acceleration	-0.0648	0.104	-0.620	0.536	-0.270	0.141
model year	0.8498	0.054	15.639	0.000	0.743	0.957
origin_2	2.1248	0.607	3.499	0.001	0.929	3.321
origin_3	1.0057	0.630	1.596	0.112	-0.235	2.247

Omnibus:	12.558	Durbin-Watson:	1.938
Prob(Omnibus):	0.002	Jarque-Bera (JB):	19.244
Skew:	0.316	Prob(JB):	6.63e-05
Kurtosis:	4.188	Cond. No.	3.62e+08

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.62e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

We have several insignicant variable (P>|t| > 0.05)

Let's fit our final model:

In [75]:
```python
model = sm.OLS(y, X).fit()
model.summary()
```

Out[75]:

OLS Regression Results

Dep. Variable:	mpg	R-squared:	0.866
Model:	OLS	Adj. R-squared:	0.863
Method:	Least Squares	F-statistic:	222.0
Date:	Wed, 08 Jan 2025	Prob (F-statistic):	5.61e-129
Time:	12:30:42	Log-Likelihood:	-790.69
No. Observations:	318	AIC:	1601.
Df Residuals:	308	BIC:	1639.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.5478	5.158	-0.300	0.764	-11.698	8.602
weight	-0.0216	0.002	-12.305	0.000	-0.025	-0.018
weight2	2.334e-06	2.65e-07	8.796	0.000	1.81e-06	2.86e-06
cylinders	-0.2249	0.321	-0.701	0.484	-0.856	0.406
displacement	0.0165	0.008	2.105	0.036	0.001	0.032
horsepower	-0.0191	0.013	-1.442	0.150	-0.045	0.007
acceleration	0.0267	0.095	0.280	0.780	-0.161	0.214
model year	0.8669	0.051	16.905	0.000	0.766	0.968
origin_2	2.3249	0.566	4.105	0.000	1.211	3.439
origin_3	1.4970	0.577	2.592	0.010	0.361	2.633

Omnibus:	30.297	Durbin-Watson:	1.911
Prob(Omnibus):	0.000	Jarque-Bera (JB):	57.166
Skew:	0.541	Prob(JB):	3.86e-13
Kurtosis:	4.773	Cond. No.	3.44e+08

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.44e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Objective 3: Evaluating Model Test Performance and Interpret the Model

Our Third Objective is to Score our Model on the Test Data Set; Use our Model to Predict a New Batch of Cars, and Interpret our Model Results.

Specific Tasks:

- Score our Final Model on the Test Set, Calculating both R2 and MAE. If our Test R2 is less than 8, Revisit the Modelling Process.
- Interpret our Model. What Impact does a One-year Increase in Model Year have on the Predicted Mileage?
- Repeat the Modelling Process using Ridge Regression. How much better was the Ridge Model than Traditional Regression, if at all?

Scoring onm Test Data- reporting R2, and MAE

In [76]:
```python
print(f"Test R2: {round(r2(y_val, model.predict(X_val)), 3)}")
print(f"Test MAE: {round(mae(y_val, model.predict(X_val)), 3)}")
```

```
Test R2: 0.852
Test MAE: 11.71
```

Objective 4: Building a Ridge Regression Model and Comparing the Results

Ridge Regression:

In [77]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeCV
```

In [78]:
```python
std = StandardScaler()
X_m = std.fit_transform(X.values)
X_te = std.transform(X_test.values)
```

In [79]:
```python
n_alphas = 200
alphas = 10 ** np.linspace(10, -2, n_alphas)

ridge_model = RidgeCV(alphas= alphas, cv= 5)
ridge_model.fit(X_m, y)

print(ridge_model.score(X_m, y))
print(mae(y, ridge_model.predict(X_m)))
print(ridge_model.alpha_)
```

```
0.8664355699560385
8.457445434084903
0.02643081486974103
```

In [80]:
```python
print(f"Test R2: {round(r2(y_test, ridge_model.predict(X_te)), 3)}")
print(f"Test MAE: {round(mae(y_test, ridge_model.predict(X_te)), 3)}")
```

```
Test R2: 0.818
Test MAE: 9.372
```

So, the R2 is reduced for the Ridge Model compared to our Linear Model. So, we can stick to the Linear Model.