```python
In [91]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```python
In [92]:
path = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/automobileEDA.csv'
df = pd.read_csv(path)
df.head()
```

Out[92]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	city-L/100km	horsepower-binned	diesel	gas
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21	27	13495.0	11.190476	Medium	0	
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21	27	16500.0	11.190476	Medium	0	
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154.0	5000.0	19	26	16500.0	12.368421	Medium	0	
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102.0	5500.0	24	30	13950.0	9.791667	Medium	0	
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115.0	5500.0	18	22	17450.0	13.055556	Medium	0	

5 rows × 29 columns

1. Linear Regression and Multiple Linear Regression

```python
In [93]:
from sklearn.linear_model import LinearRegression
```

```python
In [94]:
lm = LinearRegression()
lm
```

Out[94]:

```
▾ LinearRegression
LinearRegression()
```

```python
In [95]:
X = df[['highway-mpg']]
Y = df['price']
lm.fit(X,Y)
```

Out[95]:

```
▾ LinearRegression
LinearRegression()
```

```python
In [96]:
Yhat=lm.predict(X)
Yhat[0:5]
```

```
Out[96]:
array([16236.50464347, 16236.50464347, 17058.23802179, 13771.3045085 ,
       20345.17153508])
```

```python
In [97]:
a=lm.intercept_
b=lm.coef_
print("The value of a is",a,"and the value of b is",float(b))
```

```
The value of a is 38423.3058581574 and the value of b is -821.7333783219254
```

```python
In [98]:
lm1 = LinearRegression()
lm1.fit(df[['engine-size']], df[['price']])
a1=lm1.intercept_
b1=lm1.coef_
print("The value of a is",a1,"and the value of b is",float(b1))
```

```
The value of a is [-7963.33890628] and the value of b is 166.86001569141595
```

```python
In [99]:
Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

```python
In [100]:
lm2=LinearRegression()
lm2.fit(Z, df['price'])
a2=lm2.intercept_
b2=lm2.coef_
print("The value of a is",a2,"and the values of b are",b2)
```

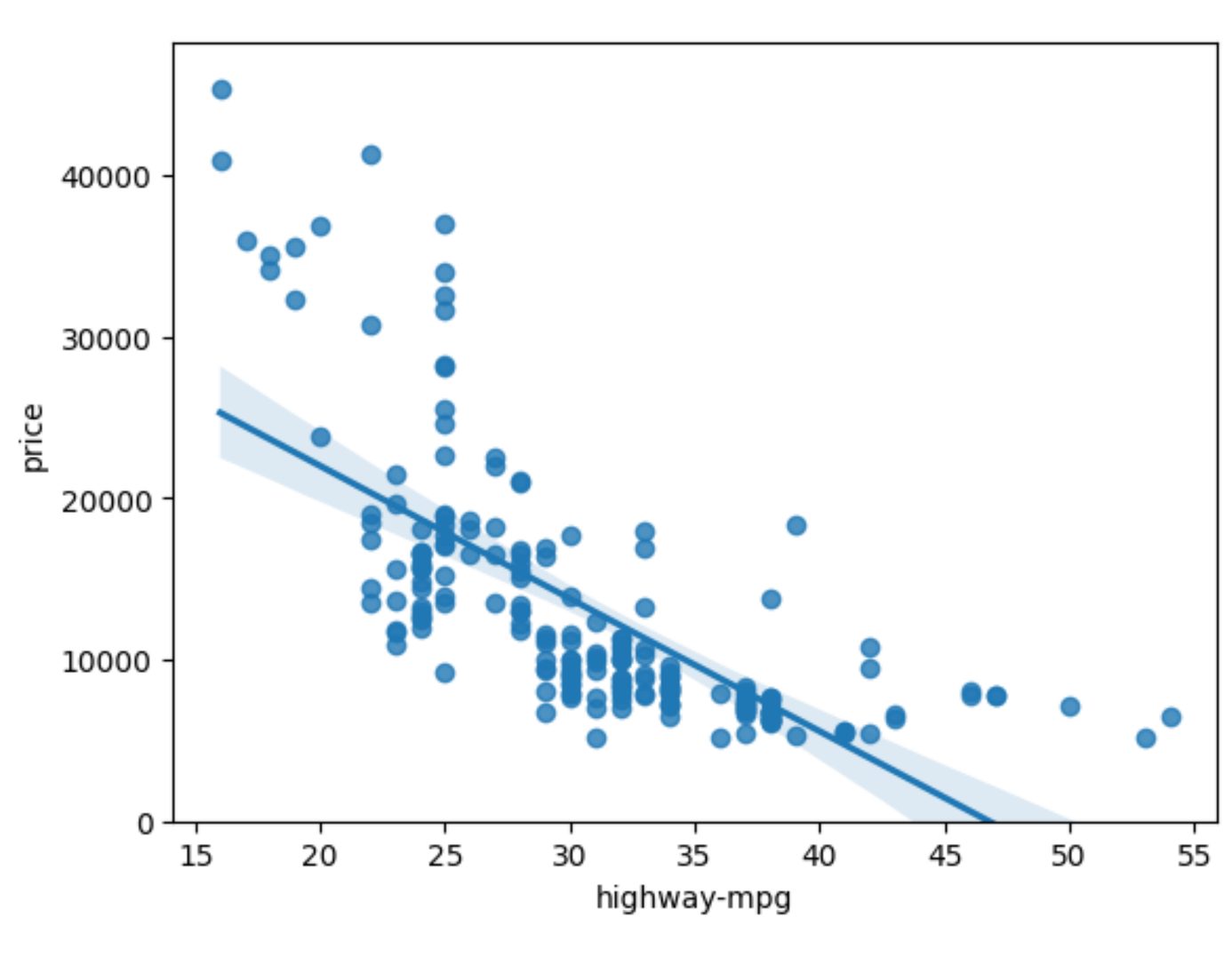
```
The value of a is -15806.62462632922 and the values of b are [53.49574423  4.70770099 81.53026382 36.05748882]
```

1. Model Evaluation Using Visualization

```python
In [101]:
import seaborn as sns
%matplotlib inline
```

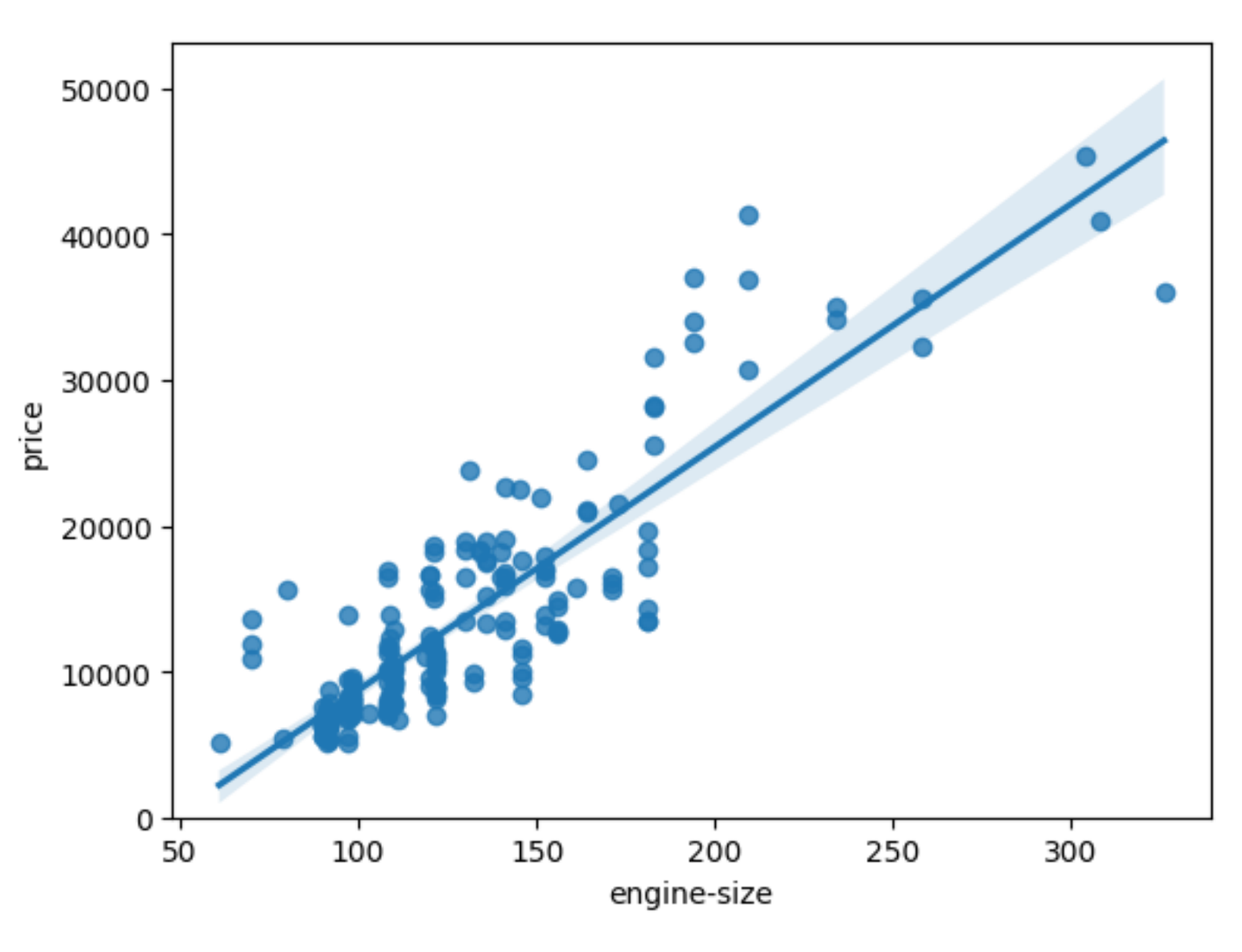
```python
In [102]:
sns.regplot(x="highway-mpg", y="price", data=df)
plt.ylim(0,)
```

```
Out[102]: (0.0, 48188.920329532535)
```




```python
In [103]:
sns.regplot(x="engine-size", y="price", data=df)
plt.ylim(0,)
```

```
Out[103]: (0.0, 53133.67852292333)
```

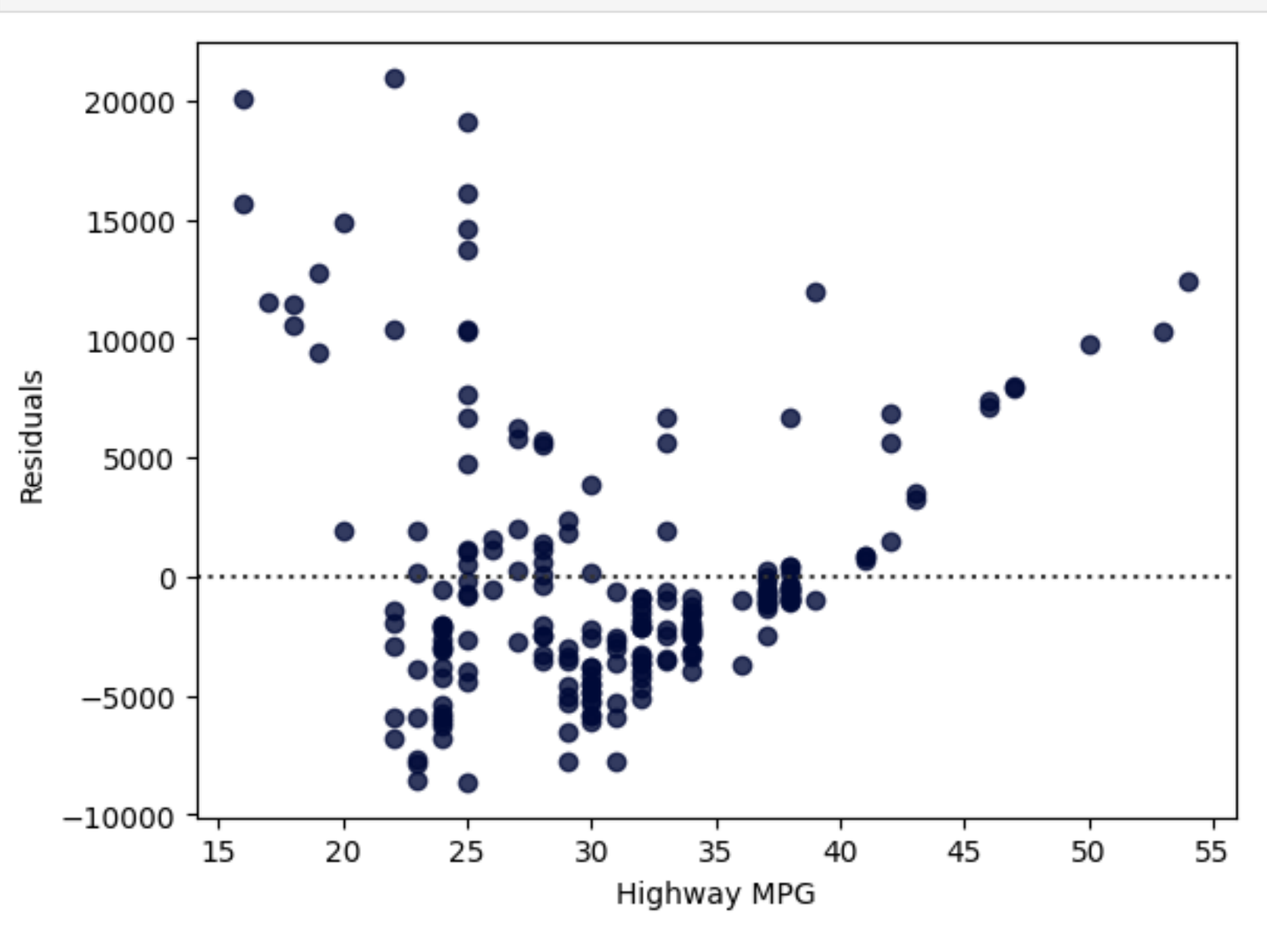



```python
In [104]:
df[["engine-size","highway-mpg","price"]].corr()
```

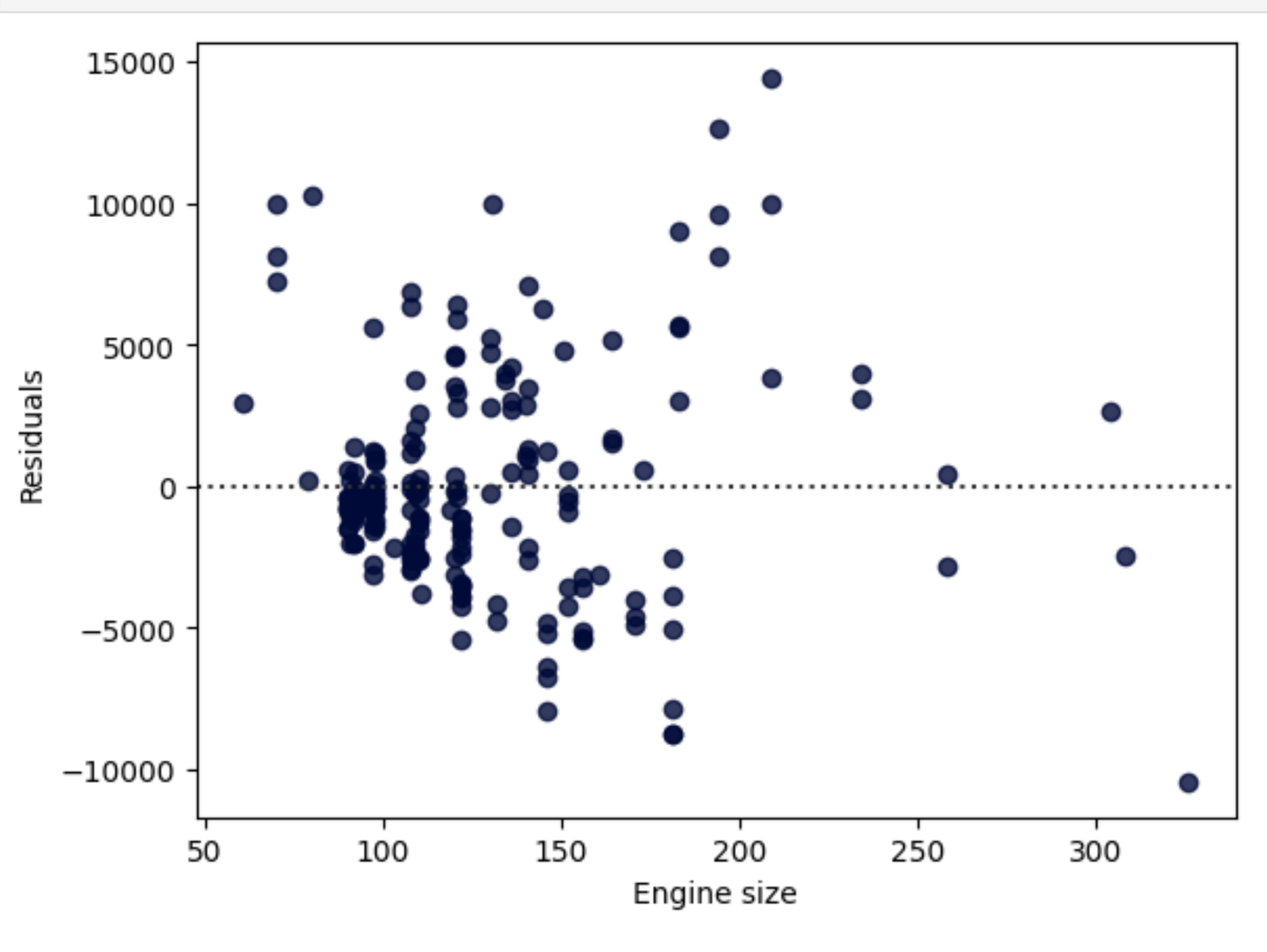
Out[104]:

	engine-size	highway-mpg	price
engine-size	1.000000	-0.679571	0.872335
highway-mpg	-0.679571	1.000000	-0.704692
price	0.872335	-0.704692	1.000000

```python
In [105]:
sns.residplot(x=df['highway-mpg'], y=df['price'],color="#000a38")
plt.xlabel("Highway MPG")
plt.ylabel("Residuals")
plt.show()
```

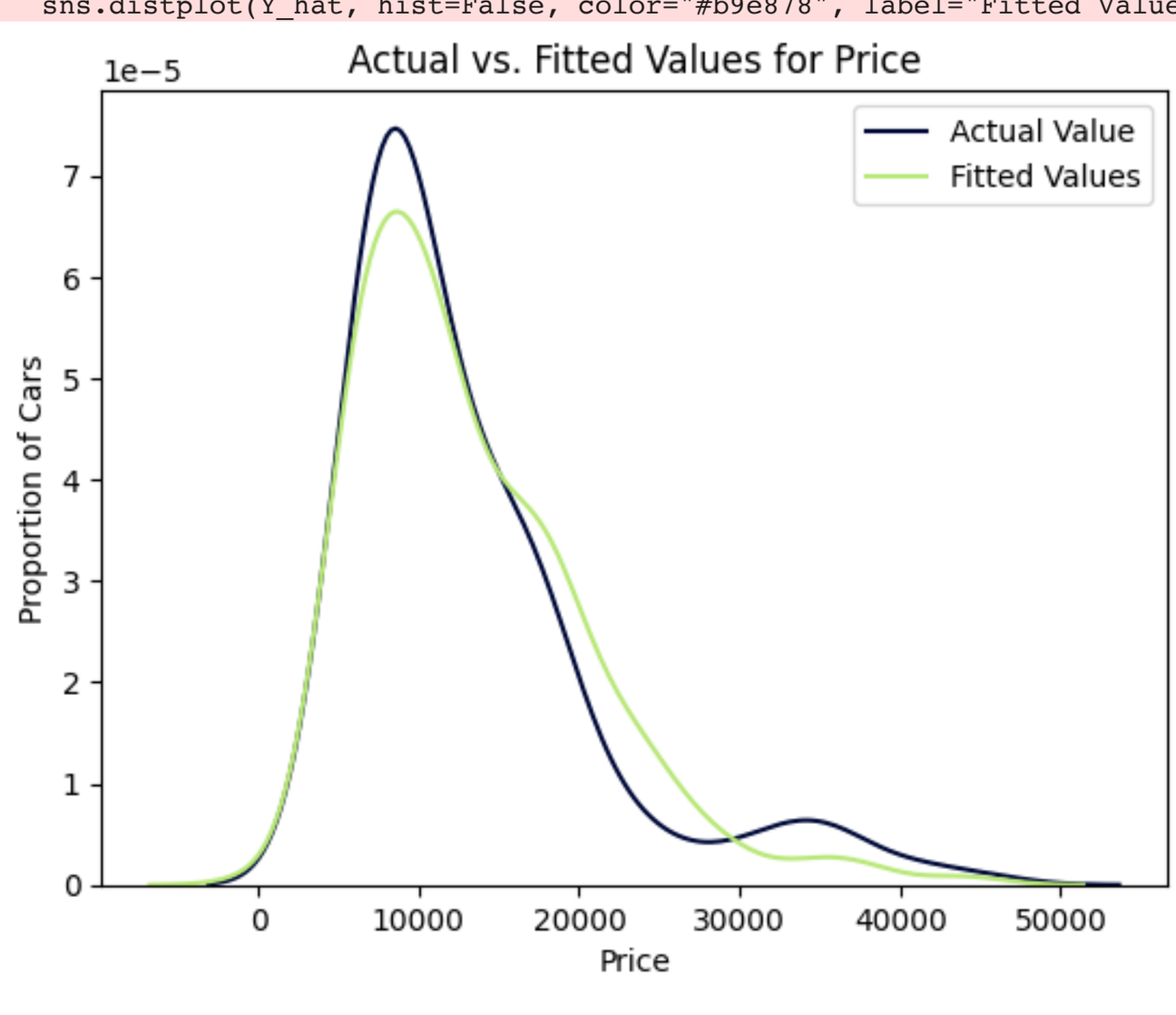



```python
In [106]:
sns.residplot(x=df['engine-size'], y=df['price'], color="#000a38")
plt.xlabel("Engine size")
plt.ylabel("Residuals")
plt.show()
```




```python
In [107]:
Y_hat = lm2.predict(Z)

ax1 = sns.distplot(df['price'], hist=False, color="#000a38", label="Actual Value")
sns.distplot(Y_hat, hist=False, color="#b9e878", label="Fitted Values" , ax=ax1)


plt.title('Actual vs. Fitted Values for Price')
plt.xlabel('Price')
plt.ylabel('Proportion of Cars')
plt.legend()

plt.show()
plt.close()
```

```
<ipython-input-107-e66354e0c2b5>:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax1 = sns.distplot(df['price'], hist=False, color="#000a38", label="Actual Value")
<ipython-input-107-e66354e0c2b5>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(Y_hat, hist=False, color="#b9e878", label="Fitted Values" , ax=ax1)
```



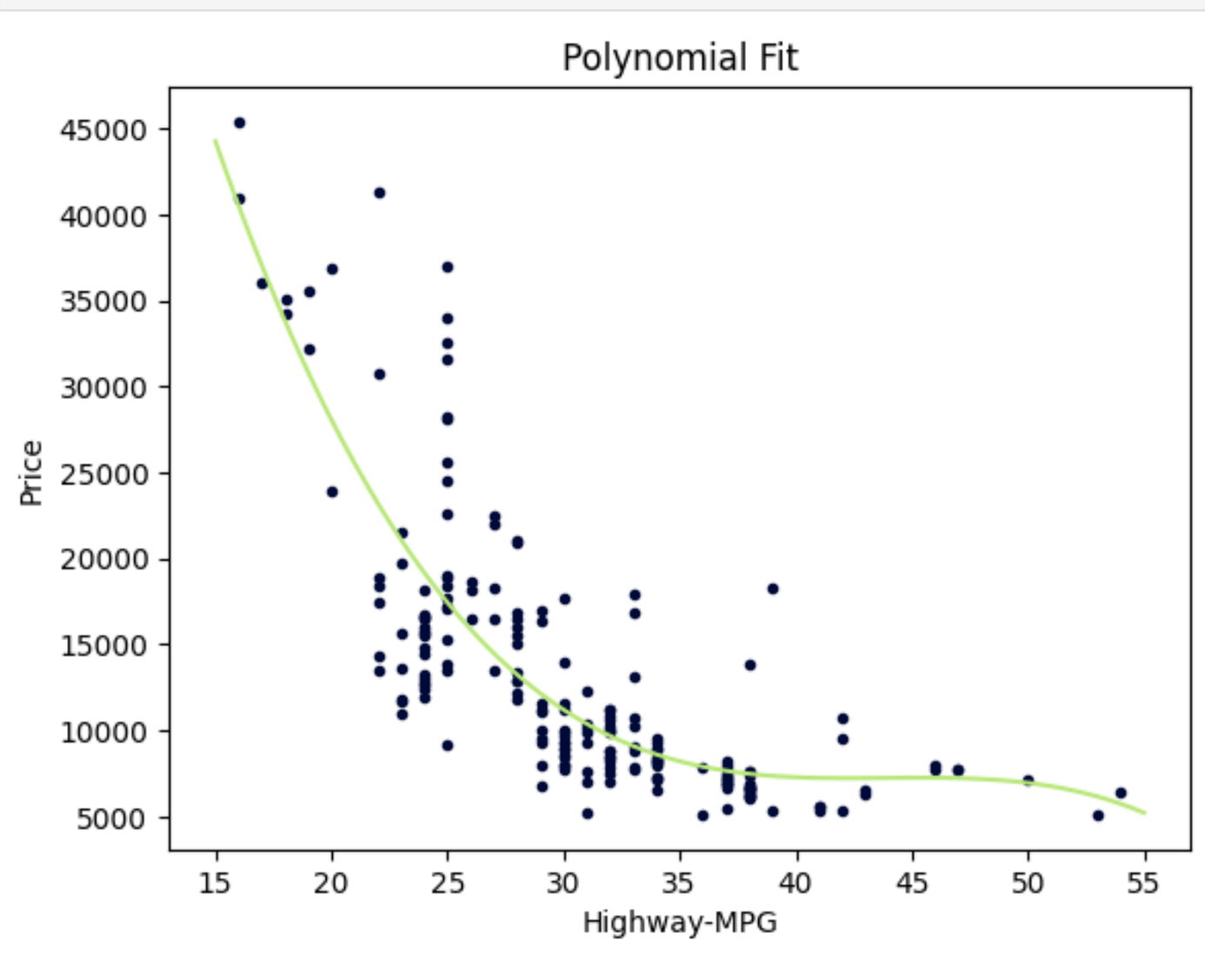
1. Polynomial Regression and Pipelines

```python
In [108]:
def PlotPolly(model, independent_variable, dependent_variabble, Name):
    x_new = np.linspace(15, 55, 100)
    y_new = model(x_new)

    plt.plot(independent_variable, dependent_variabble, '.',color="#000a38")
    plt.plot(x_new, y_new, '-',color="#b9e878")
    plt.title('Polynomial Fit')
    ax = plt.gca()
    ax.set_facecolor((1, 1, 1))
    fig = plt.gcf()
    plt.xlabel('Highway-MPG')
    plt.ylabel('Price')

    plt.show()
    plt.close()
```

```python
In [109]:
x = df['highway-mpg']
y = df['price']
f = np.polyfit(x, y, 3)
p = np.poly1d(f)
print(p)
```

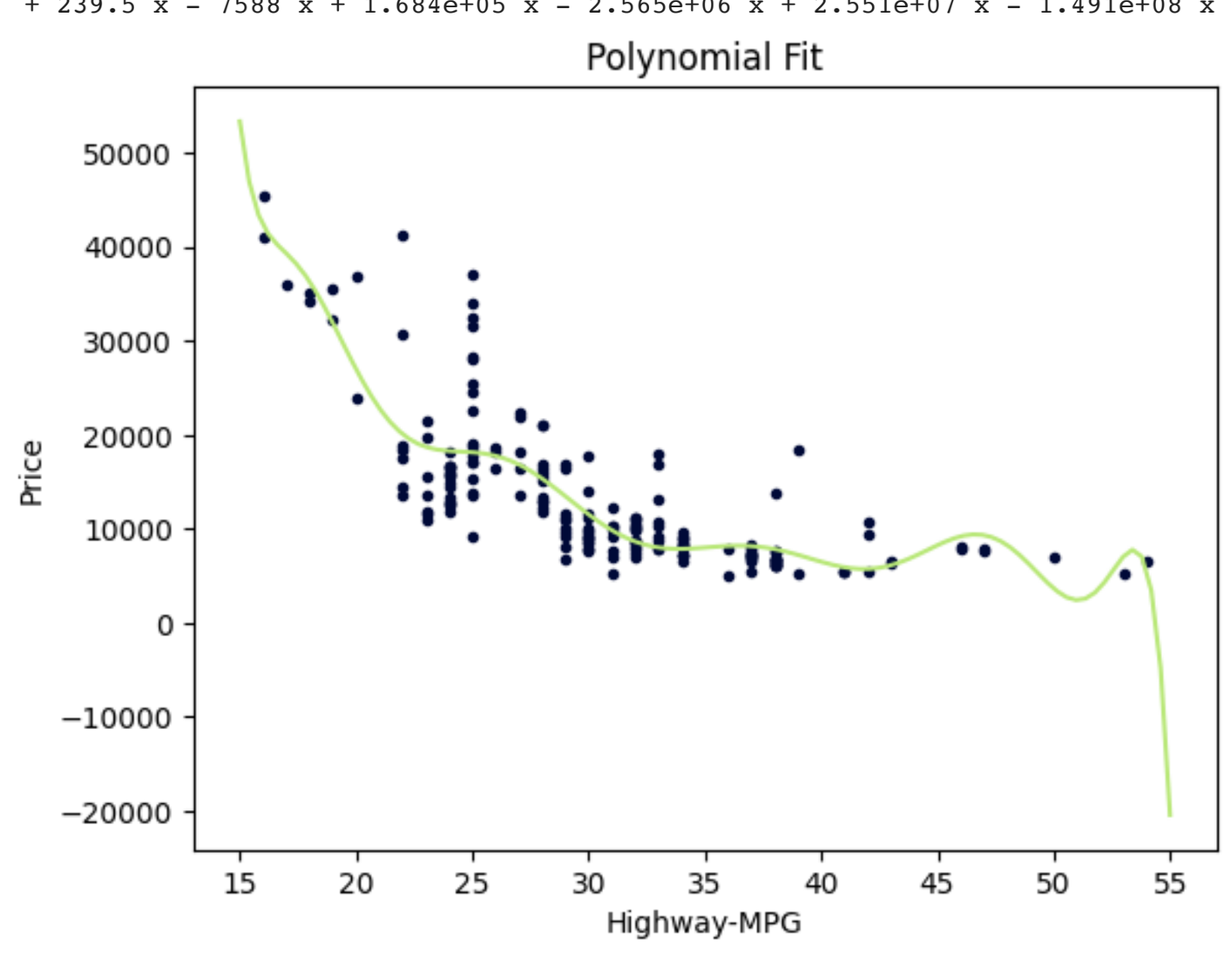
```
       3         2
-1.557 x + 204.8 x - 8965 x + 1.379e+05
```

```python
In [110]:
PlotPolly(p, x, y, 'highway-mpg')
```




```python
In [111]:
f1 = np.polyfit(x, y, 11)
p1 = np.poly1d(f1)
print(p1)
PlotPolly(p1,x,y, 'Highway MPG')
```

```
            11             10             9           8         7
-1.243e-08 x  + 4.722e-06 x  - 0.0008028 x + 0.08056 x - 5.297 x
          6        5             4             3             2
 + 239.5 x - 7588 x + 1.684e+05 x - 2.565e+06 x + 2.551e+07 x - 1.491e+08 x + 3.879e+08
```

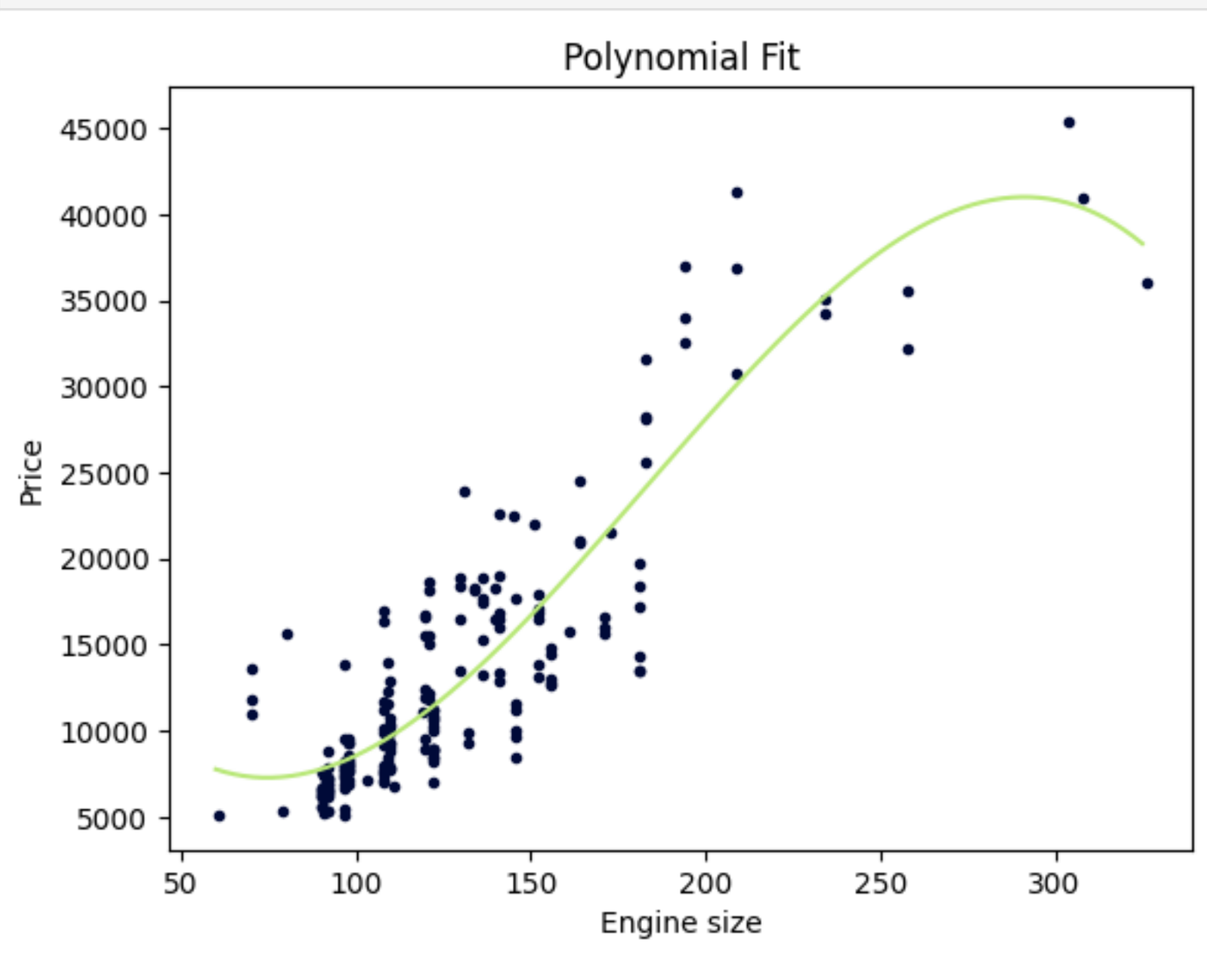



```python
In [112]:
def PlotPolly1(model, independent_variable, dependent_variabble, Name):
    x_new = np.linspace(60, 325, 100)
    y_new = model(x_new)

    plt.plot(independent_variable, dependent_variabble, '.',color="#000a38")
    plt.plot(x_new, y_new, '-',color="#b9e878")
    plt.title('Polynomial Fit')
    ax = plt.gca()
    ax.set_facecolor((1, 1, 1))
    fig = plt.gcf()
    plt.xlabel('Engine size')
    plt.ylabel('Price')

    plt.show()
    plt.close()
```

```python
In [113]:
x1 = df['engine-size']
y1 = df['price']
f2 = np.polyfit(x1, y1, 3)
p2 = np.poly1d(f2)
print(p2)
```

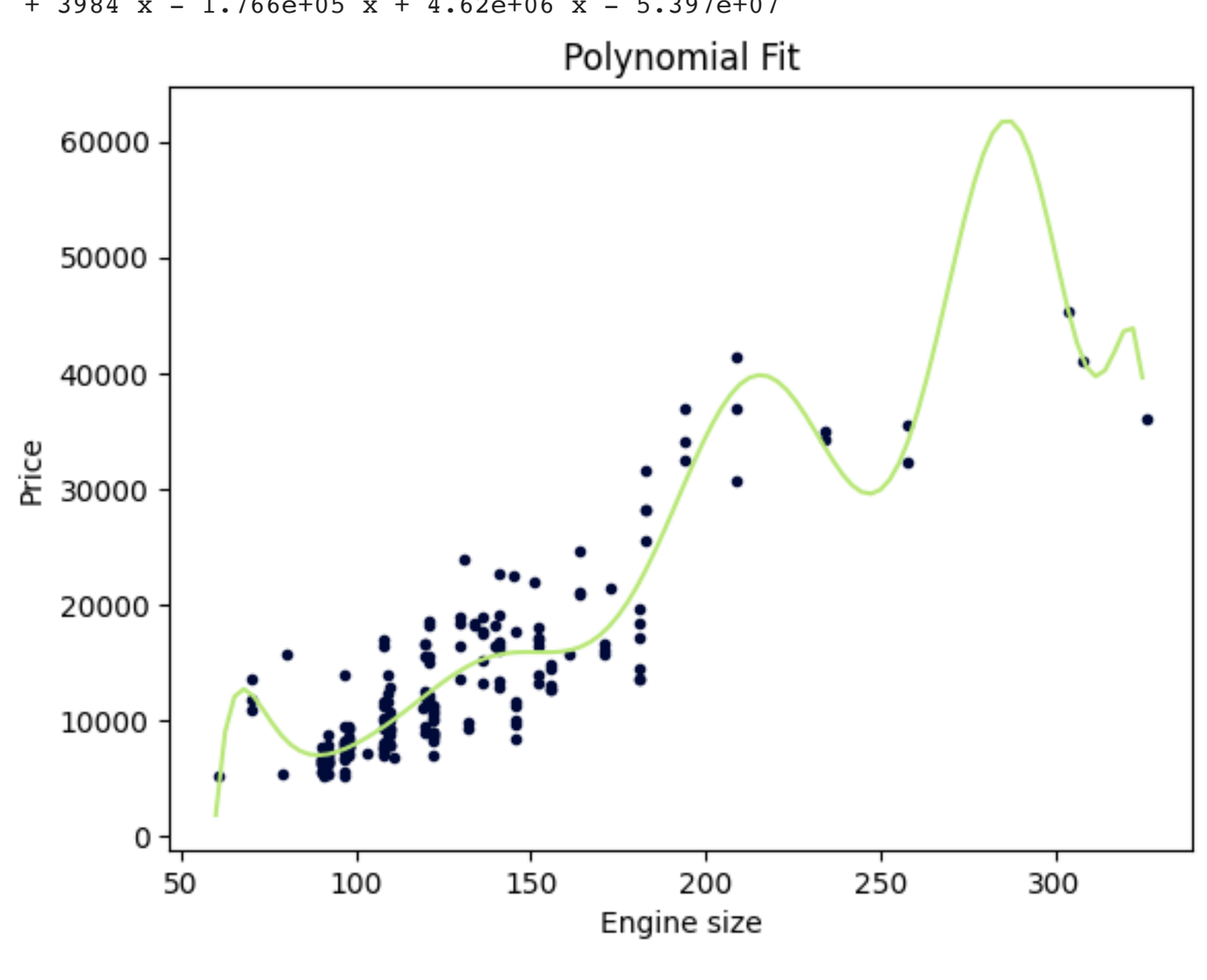
```
          3         2
-0.006641 x + 3.645 x - 433.4 x + 2.208e+04
```

```python
In [114]:
PlotPolly1(p2, x1, y1, 'Engine Size')
```




```python
In [115]:
f3 = np.polyfit(x1, y1,12)
p3 = np.poly1d(f3)
print(p3)
PlotPolly1(p3,x1,y1, 'Engine Size')
```

```
            12             11             10             9
-1.684e-19 x  + 3.757e-16 x  - 3.751e-13 x  + 2.216e-10 x
             8             7            6          5         4
 - 8.614e-08 x + 2.321e-05 x - 0.004441 x + 0.6081 x - 59.14 x
        3             2
 + 3984 x - 1.766e+05 x + 4.62e+06 x - 5.397e+07
```




```python
In [116]:
from sklearn.preprocessing import PolynomialFeatures
```

```python
In [117]:
pr=PolynomialFeatures(degree=2)
Z_pr=pr.fit_transform(Z)
Z_pr.shape
```

```
Out[117]: (201, 15)
```

```python
In [118]:
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```python
In [119]:
Input=[('scale',StandardScaler()), ('polynomial', PolynomialFeatures(include_bias=False)), ('model',LinearRegression())]
pipe=Pipeline(Input)
pipe
```

Out[119]:

```
▸ Pipeline
  ▸ StandardScaler
  ▸ PolynomialFeatures
  ▸ LinearRegression
```

```python
In [120]:
Z = Z.astype(float)
pipe.fit(Z,y)
```

Out[120]:

```
▸ Pipeline
  ▸ StandardScaler
  ▸ PolynomialFeatures
  ▸ LinearRegression
```

```python
In [121]:
ypipe=pipe.predict(Z)
ypipe[0:4]
```

```
Out[121]: array([13102.74784201, 13102.74784201, 18225.54572197, 10390.29636555])
```

1. Measures for in-sample evaluation

Model 1: Simple Linear Regression (Highway-MPG)

```python
In [122]:
lm.fit(X, Y)
print('The R-square is: ', lm.score(X, Y))
```

```
The R-square is:  0.4965911884339176
```

```python
In [123]:
Yhat=lm.predict(X)
print('The output of the first four predicted value is: ', Yhat[0:4])
```

```
The output of the first four predicted value is:  [16236.50464347 16236.50464347 17058.23802179 13771.3045085 ]
```

```python
In [124]:
from sklearn.metrics import mean_squared_error
```

```python
In [125]:
mse = mean_squared_error(df['price'], Yhat)
print('The mean square error of price and predicted value is: ', mse)
```

```
The mean square error of price and predicted value is:  31635042.944639888
```

Model 1: Simple Linear Regression (Engine-size)

```python
In [126]:
lm1.fit(df[['engine-size']], df[['price']])
print('The R-square is: ', lm1.score(df[['engine-size']], df[['price']]))
```

```
The R-square is:  0.7609686443622008
```

```python
In [127]:
Yhat1=lm1.predict(df[['engine-size']])
print('The output of the first four predicted value is: ', Yhat1[0:4])
```

```
The output of the first four predicted value is:  [[13728.4631336 ]
 [13728.4631336 ]
 [17399.38347881]
 [10224.40280408]]
```

```python
In [128]:
mse1 = mean_squared_error(df['price'], Yhat1)
print('The mean square error of price and predicted value is: ', mse1)
```

```
The mean square error of price and predicted value is:  15021126.025174143
```

Model 2: Multiple Linear Regression

```python
In [129]:
lm.fit(Z, df['price'])
print('The R-square is: ', lm.score(Z, df['price']))
```

```
The R-square is:  0.8093562806577457
```

```python
In [130]:
Y_predict_multifit = lm.predict(Z)
print('The mean square error of price and predicted value using multifit is: ', \
      mean_squared_error(df['price'], Y_predict_multifit))
```

```
The mean square error of price and predicted value using multifit is:  11980366.87072649
```

Model 3: Polynomial Fit (Highway-MPG)

```python
In [131]:
from sklearn.metrics import r2_score
```

```python
In [132]:
r_squared = r2_score(y, p(x))
print('The R-square value is: ', r_squared)
```

```
The R-square value is:  0.674194666390652
```

```python
In [133]:
mean_squared_error(df['price'], p(x))
```

```
Out[133]: 20474146.426361218
```

Model 3: Polynomial Fit (Engine-size)

```python
In [134]:
r_squared1 = r2_score(y1, p2(x1))
print('The R-square value is: ', r_squared1)
```

```
The R-square value is:  0.7901519931574332
```

```python
In [135]:
mean_squared_error(df['price'], p2(x1))
```

```
Out[135]: 13187196.08355573
```