Model Evaluation

In [2]:
```
import pandas as pd
import numpy as np
import scipy
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
```

In [3]:
```
#!pip install ipywidgets
#!pip install tqdm
```

In [4]:
```
path = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/module
```

In [5]:
```
df = pd.read_csv(path)
```

In [6]:
```
df.to_csv('module_5_auto.csv')
```

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

Out[7]:

	Unnamed: 0.1	Unnamed: 0	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	...	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	city-L/100km
0	0	0	3	122	88.6	0.811148	0.890278	48.8	2548	130	...	2.68	9.0	111.0	5000.0	21	27	13495.0	11.190476
1	1	1	3	122	88.6	0.811148	0.890278	48.8	2548	130	...	2.68	9.0	111.0	5000.0	21	27	16500.0	11.190476
2	2	2	1	122	94.5	0.822681	0.909722	52.4	2823	152	...	3.47	9.0	154.0	5000.0	19	26	16500.0	12.368421
3	3	3	2	164	99.8	0.848630	0.919444	54.3	2337	109	...	3.40	10.0	102.0	5500.0	24	30	13950.0	9.791667
4	4	4	2	164	99.4	0.848630	0.922222	54.3	2824	136	...	3.40	8.0	115.0	5500.0	18	22	17450.0	13.055556

5 rows × 21 columns

In [8]:
```
from ipywidgets import interact, interactive, fixed, interact_manual
```

Function

In [28]:
```
def DistributionPlot(RedFunction, BlueFunction, RedName, BlueName, Title):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    ax1 = sns.kdeplot(RedFunction, color="r", label=RedName) # update distplot
    ax2 = sns.kdeplot(BlueFunction, color="b", label=BlueName, ax=ax1) # update distplot

    plt.title(Title)
    plt.xlabel('Price (in dollars)')
    plt.ylabel('Proportion of Cars')

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

In [10]:
```
def PollyPlot(xtrain, xtest, y_train, y_test, lr,poly_transform):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    #poly_transform:  polynomial transformation object

    xmax=max([xtrain.values.max(), xtest.values.max()])

    xmin=min([xtrain.values.min(), xtest.values.min()])

    x=np.arange(xmin, xmax, 0.1)


    plt.plot(xtrain, y_train, 'ro', label='Training Data')
    plt.plot(xtest, y_test, 'go', label='Test Data')
    plt.plot(x, lr.predict(poly_transform.fit_transform(x.reshape(-1, 1))), label='Predicted Function')
    plt.ylim([-10000, 60000])
    plt.ylabel('Price')
    plt.legend()
```

Part 1: Training and Testing

In [11]:
```
y_data = df['price']
```

In [12]:
```
x_data=df.drop('price',axis=1)
```

In [13]:
```
from sklearn.model_selection import train_test_split


x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.10, random_state=1)


print("number of test samples :", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
```

```
number of test samples : 21
number of training samples: 180
```

In [14]:
```
from sklearn.linear_model import LinearRegression
```

In [15]:
```
lre=LinearRegression()
```

In [16]:
```
lre.fit(x_train[['horsepower']], y_train)
```

Out[16]:
```
▾ LinearRegression
LinearRegression()
```

In [17]:
```
lre.score(x_test[['horsepower']], y_test)
```

Out[17]:
```
0.3635875575078824
```

In [18]:
```
lre.score(x_train[['horsepower']], y_train)
```

Out[18]:
```
0.6619724197515103
```

The R^2 is much smaller using the test data compared to the training data.

Part 2: Overfitting, Underfitting and Model Selection

To Create Multiple Linear Regression objects and train the model using 'horsepower', 'curb-weight', 'engine-size' and 'highway-mpg' as features.

In [19]:
```
lr = LinearRegression()
lr.fit(x_train[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_train)
```

Out[19]:
```
▾ LinearRegression
LinearRegression()
```

In [20]:
```
yhat_train = lr.predict(x_train[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])
yhat_train[0:5]
```

Out[20]:
```
array([ 7426.6731551 , 28323.75090803, 14213.38819709,  4052.34146983,
       34500.19124244])
```

In [21]:
```
yhat_test = lr.predict(x_test[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])
yhat_test[0:5]
```

Out[21]:
```
array([11349.35089149,  5884.11059106, 11208.6928275 ,  6641.07786278,
       15565.79920282])
```

In [22]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Let's examine the distribution of the predicted values of the training data.

In [29]:
```
Title = 'Distribution  Plot of  Predicted Value Using Training Data vs Training Data Distribution'
DistributionPlot(y_train, yhat_train, "Actual Values (Train)", "Predicted Values (Train)", Title)
```

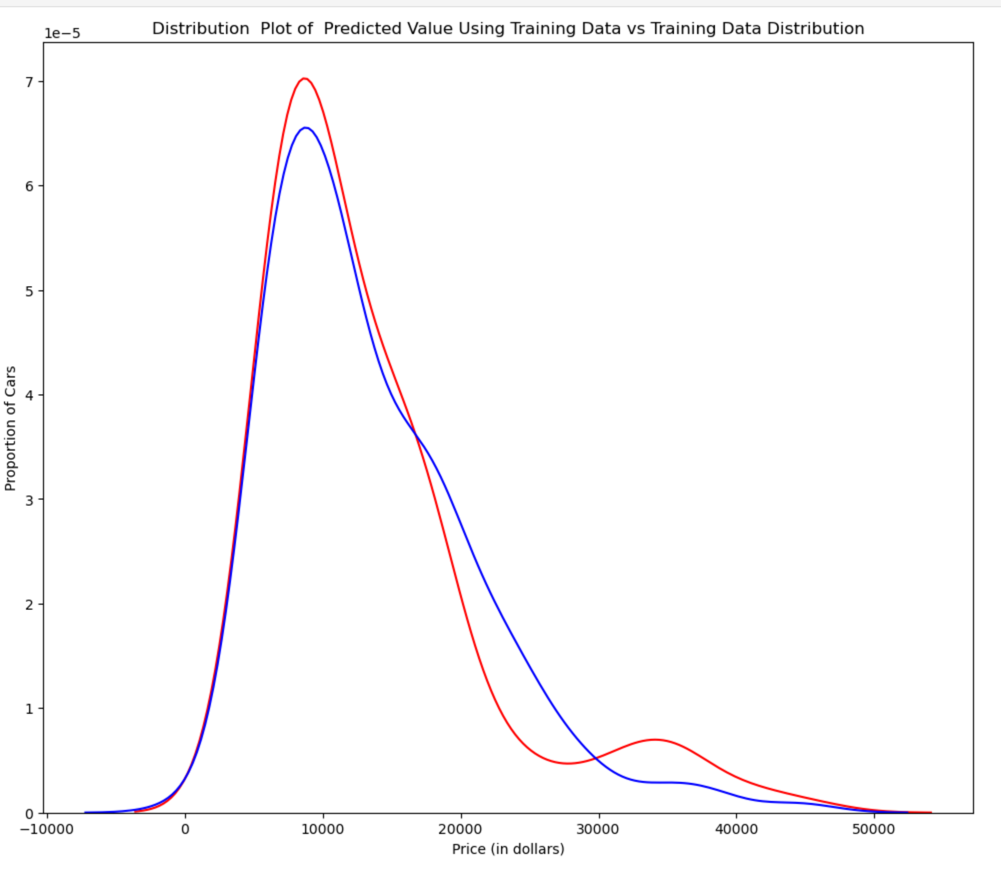


Figure 1: Plot of predicted values using the training data compared to the actual values of the training data.

In [30]:
```
Title='Distribution  Plot of  Predicted Value Using Test Data vs Data Distribution of Test Data'
DistributionPlot(y_test,yhat_test,"Actual Values (Test)","Predicted Values (Test)",Title)
```

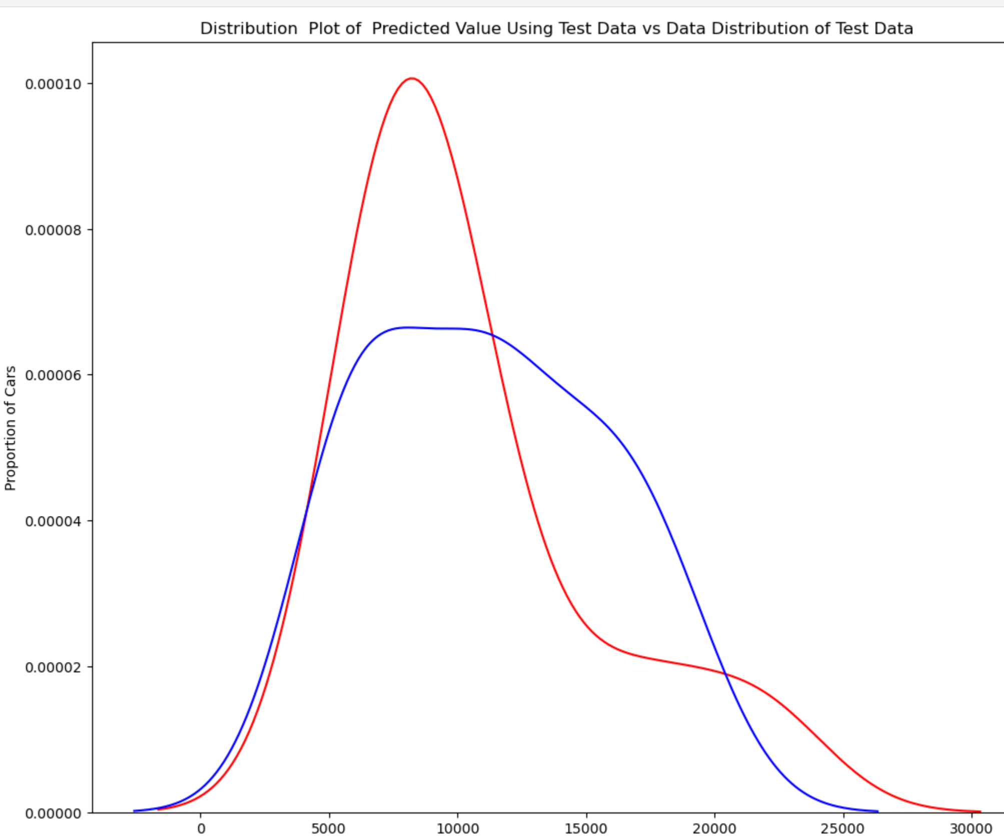


Figure 2: Plot of predicted value using the test data compared to the actual values of the test data.

Comparing Figure 1 and Figure 2, it is evident that the distribution of the test data in Figure 1 is much better at fitting the data. This difference in Figure 2 is apparent in the range of 5000 to 15,000. This is where the shape of the distribution is extremely different. Let's see if polynomial regression also exhibits a drop in the prediction accuracy when analysing the test dataset.

In [31]:
```
from sklearn.preprocessing import PolynomialFeatures
```

Overfitting

In [32]:
```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.45, random_state=0)
```

In [33]:
```
pr = PolynomialFeatures(degree=5)
x_train_pr = pr.fit_transform(x_train[['horsepower']])
x_test_pr = pr.fit_transform(x_test[['horsepower']])
pr
```

Out[33]:
```
▾ PolynomialFeatures
PolynomialFeatures(degree=5)
```

In [34]:
```
poly = LinearRegression()
poly.fit(x_train_pr, y_train)
```

Out[34]:
```
▾ LinearRegression
LinearRegression()
```

In [35]:
```
yhat = poly.predict(x_test_pr)
yhat[0:5]
```

Out[35]:
```
array([ 6728.7450134 ,  7308.0678859 , 12213.81567729, 18893.11763607,
       19995.80011736])
```

In [36]:
```
print("Predicted values:", yhat[0:4])
print("True values:", y_test[0:4].values)
```

```
Predicted values: [ 6728.7450134   7308.0678859  12213.81567729 18893.11763607]
True values: [ 6295. 10698. 13860. 13499.]
```

In [37]:
```
PollyPlot(x_train[['horsepower']], x_test[['horsepower']], y_train, y_test, poly,pr)
```

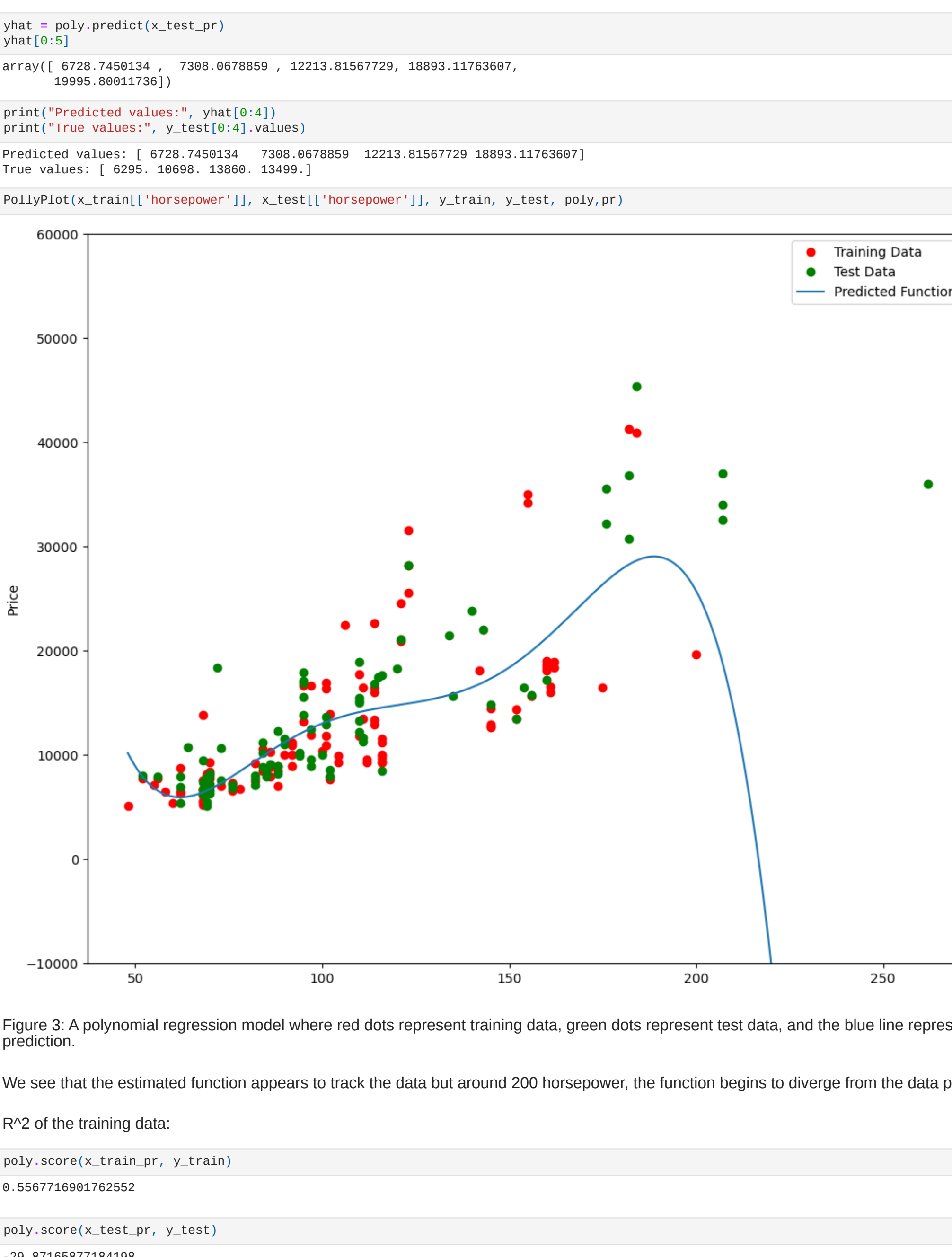


Figure 3: A polynomial regression model where red dots represent training data, green dots represent test data, and the blue line represents the model prediction.

We see that the estimated function appears to track the data but around 200 horsepower, the function begins to diverge from the data points.

R^2 of the training data:

In [38]:
```
poly.score(x_train_pr, y_train)
```

Out[38]:
```
0.5567716901762552
```

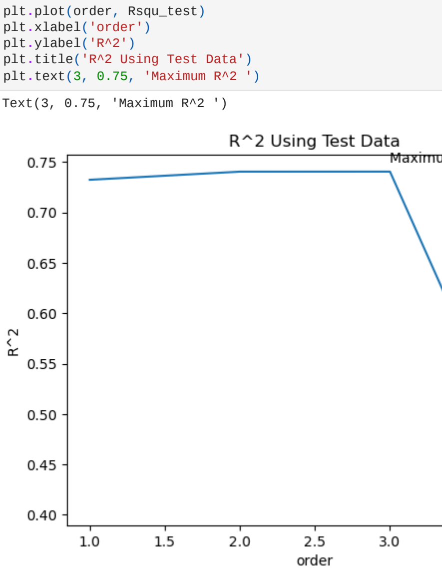
In [39]:
```
poly.score(x_test_pr, y_test)
```

Out[39]:
```
-29.87165877184198
```

We see the R^2 for the training data is 0.5567 while the R^2 on the test data was -29.87. The lower the R^2, the worse the model. A negative R^2 is a sign of overfitting.

In [40]:
```
Rsqu_test = []

order = [1, 2, 3, 4]
for n in order:
    pr = PolynomialFeatures(degree=n)

    x_train_pr = pr.fit_transform(x_train[['horsepower']])

    x_test_pr = pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr, y_train)

    Rsqu_test.append(lr.score(x_test_pr, y_test))

plt.plot(order, Rsqu_test)
plt.xlabel('order')
plt.ylabel('R^2')
plt.title('R^2 Using Test Data')
plt.text(3, 0.75, 'Maximum R^2 ')
```

Out[40]:
```
Text(3, 0.75, 'Maximum R^2 ')
```



We see the R^2 gradually increases until an order three polynomial is used. Then, the R^2 dramatically decreases at an order four polynomial.

In [41]:
```
def f(order, test_data):
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=test_data, random_state=0)
    pr = PolynomialFeatures(degree=order)
    x_train_pr = pr.fit_transform(x_train[['horsepower']])
    x_test_pr = pr.fit_transform(x_test[['horsepower']])
    poly = LinearRegression()
    poly.fit(x_train_pr,y_train)
    PollyPlot(x_train[['horsepower']], x_test[['horsepower']], y_train,y_test, poly, pr)
```

In [42]:
```
from sklearn.impute import SimpleImputer
```

In [43]:
```
interact(f, order=(0, 6, 1), test_data=(0.05, 0.95, 0.05))
```

Out[43]:
```
interactive(children=(IntSlider(value=3, description='order', max=6), FloatSlider(value=0.45, description='tes…
<function __main__.f(order, test_data)>
```

Part 3: Ridge Regression

In [44]:
```
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg','normalized-losses','symboling']])
x_test_pr=pr.fit_transform(x_test[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg','normalized-losses','symboling']])
```

In [45]:
```
from sklearn.linear_model import Ridge
```

To create a Ridge regression object, setting the regularization parameter (alpha) to 0.1

In [46]:
```
RigeModel=Ridge(alpha=1)
```

In [47]:
```
RigeModel.fit(x_train_pr, y_train)
```

Out[47]:
```
▾ Ridge
Ridge(alpha=1)
```

In [48]:
```
yhat = RigeModel.predict(x_test_pr)
```

In [49]:
```
print('predicted:', yhat[0:4])
print('test set :', y_test[0:4].values)
```

```
predicted: [ 6570.82441941  9636.24891471 20949.92322737 19403.60313255]
test set : [ 6295. 10698. 13860. 13499.]
```

To select the value of alpha that minimizes the test error. To do so, can use a for loop

In [50]:
```
from tqdm import tqdm

Rsqu_test = []
Rsqu_train = []
dummy1 = []
Alpha = 10 * np.array(range(0,1000))
pbar = tqdm(Alpha)

for alpha in pbar:
    RigeModel = Ridge(alpha=alpha)
    RigeModel.fit(x_train_pr, y_train)
    test_score, train_score = RigeModel.score(x_test_pr, y_test), RigeModel.score(x_train_pr, y_train)

    pbar.set_postfix({"Test Score": test_score, "Train Score": train_score})

    Rsqu_test.append(test_score)
    Rsqu_train.append(train_score)
```

```
100%|██████████████████████████████████████| 1000/1000 [00:04<00:00, 244.44it/s, Test Score=0.564, Train Score=0.859]
```

In [51]:
```
width = 12
height = 10
plt.figure(figsize=(width, height))

plt.plot(Alpha,Rsqu_test, label='validation data  ')
plt.plot(Alpha,Rsqu_train, 'r', label='training Data ')
plt.xlabel('alpha')
plt.ylabel('R^2')
plt.legend()
```

Out[51]:
```
<matplotlib.legend.Legend at 0x15c489d7850>
```

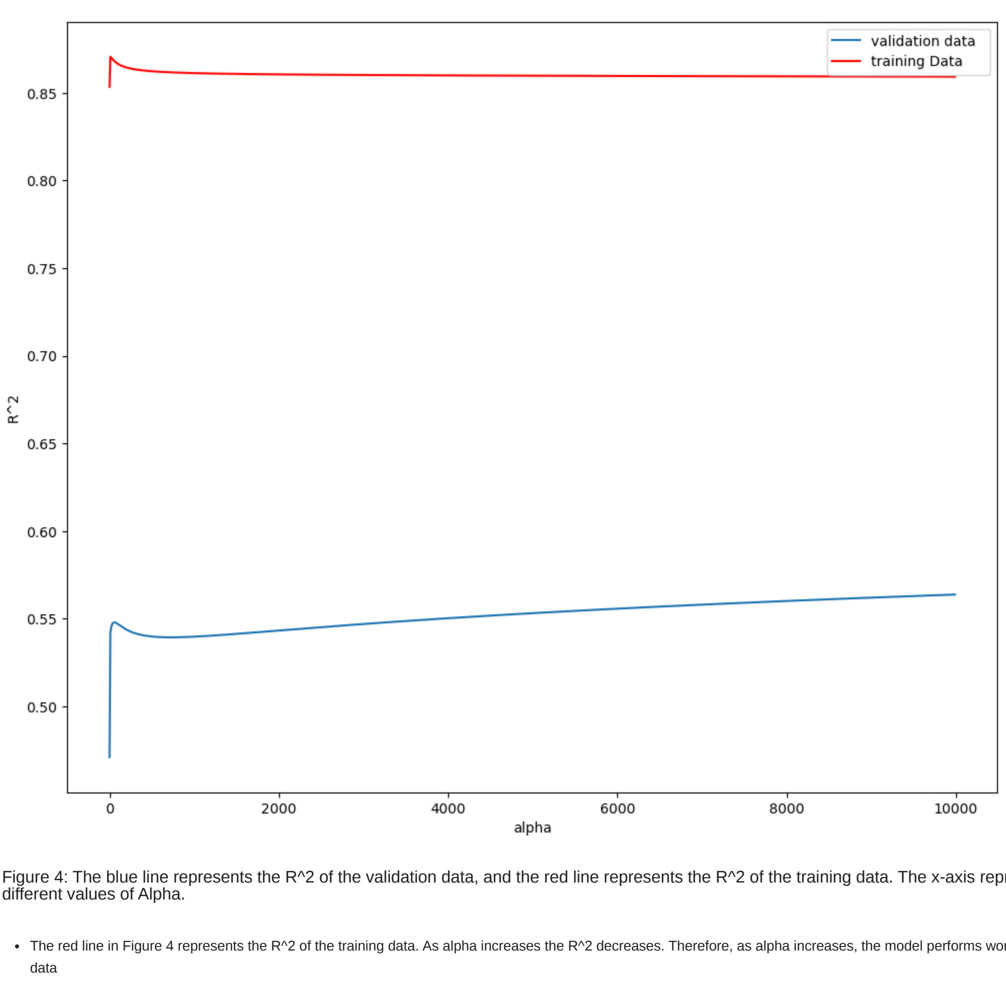


Figure 4: The blue line represents the R^2 of the validation data, and the red line represents the R^2 of the training data. The x-axis represents the different values of Alpha.

- The red line in Figure 4 represents the R^2 of the training data. As alpha increases the R^2 decreases. Therefore, as alpha increases, the model performs worse on the training data
- The blue line represents the R^2 on the validation data. As the value for alpha increases, the R^2 increases and converges at a point.

Part 4: Grid Search

In [52]:
```
from sklearn.model_selection import GridSearchCV
```

In [53]:
```
# to create a dictionary of parameter values:
parameters1= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000, 100000]}]
parameters1
```

Out[53]:
```
[{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 10000, 100000, 100000]}]
```

In [54]:
```
RR=Ridge()
RR
```

Out[54]:
```
▾ Ridge
Ridge()
```

In [55]:
```
Grid1 = GridSearchCV(RR, parameters1,cv=4)
```

In [56]:
```
Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_data)
```

Out[56]:
```
▸ GridSearchCV
▸ estimator: Ridge
  ▸ Ridge
```

In [57]:
```
BestRR=Grid1.best_estimator_
BestRR
```

Out[57]:
```
▾ Ridge
Ridge(alpha=10000)
```

In [58]:
```
BestRR.score(x_test[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_test)
```

Out[58]:
```
0.8411649831036149
```

Conclusion:

- Alpha = 10000, the best score of R-squared is 0.84
- The price car is well explained by the model with 84%