#Evaluación de Modelo y Refinamiento

- Evaluar y refinar los modelos de predicción.

##Se cargan las librerias

```
import pandas as pd
import numpy as np

path = '/content/sample_data/module_5_auto.csv'

df = pd.read_csv(path)

df.to_csv('module_5_auto.csv')

df=df._get_numeric_data()
df.head()
```

```
   Unnamed: 0  Unnamed: 0.1  symboling  normalized-losses  wheel-base
\
0           0             0          3                122        88.6

1           1             1          3                122        88.6

2           2             2          1                122        94.5

3           3             3          2                164        99.8

4           4             4          2                164        99.4


     length     width  height  curb-weight  engine-size  ...
stroke  \
0  0.811148  0.890278    48.8         2548          130  ...    2.68

1  0.811148  0.890278    48.8         2548          130  ...    2.68

2  0.822681  0.909722    52.4         2823          152  ...    3.47

3  0.848630  0.919444    54.3         2337          109  ...    3.40

4  0.848630  0.922222    54.3         2824          136  ...    3.40


   compression-ratio  horsepower  peak-rpm  city-mpg  highway-mpg
price  \
0                9.0       111.0    5000.0        21           27
13495.0
1                9.0       111.0    5000.0        21           27
16500.0
```

```
2                9.0       154.0    5000.0        19           26
16500.0
3               10.0       102.0    5500.0        24           30
13950.0
4                8.0       115.0    5500.0        18           22
17450.0

   city-L/100km  diesel  gas
0     11.190476       0    1
1     11.190476       0    1
2     12.368421       0    1
3      9.791667       0    1
4     13.055556       0    1

[5 rows x 21 columns]
```

```
#Se cargan las librerias para graficado
from ipywidgets import interact, interactive, fixed, interact_manual
```

##Se definen funciones para graficado

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

    ax1 = sns.distplot(RedFunction, hist=False, color="r",
label=RedName)
    ax2 = sns.distplot(BlueFunction, hist=False, color="b",
label=BlueName, ax=ax1)

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

    plt.show()
    plt.close()

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


    #training data
    #testing data
    # lr:  linear regression object
    #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()
```

##Entrenamiento y Prueba

```
# Dividimos nuestros datos
#Salida o Target (Clase)
y_data = df['price']

#Carateristicas, Descriptores o Variables.
x_data=df.drop('price',axis=1)

#División de los datos de entrenamiento y prueba
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)

#Se imprimen el número de datos para entrenamiento y prueba
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
```

###**Pregunta 1:** Utiliza la ffunción "train_test_split" para dividir el conjunto de datos en 40% de las muestras utilizadas para pruab. Ajusta el parametros "random_state igual a zero". La salida de la función debe de ser la siguietne: "x_train1", "x_test1", y_train1" y "y_test1".

```
#Similar al código anterior con los respectivos ajustes
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_data,
y_data, test_size=0.40, random_state=0)

#Se imprimen el número de datos para entrenamiento y prueba
print("number of test samples :", x_test1.shape[0])
print("number of training samples:",x_train1.shape[0])
```

```
number of test samples : 81
number of training samples: 120
```

###Importamos el modelo o algoritmo a utilizar

```
from sklearn.linear_model import LinearRegression

#Creamos el objeto de Regresor Lineal
lre = LinearRegression()

#Ajustamos el modelo utilizando la variables "horsepower"
lre.fit(x_train[['horsepower']], y_train)

LinearRegression()

#Calculamos la metrica R^2 con los datos de prueba
lre.score(x_test[['horsepower']], y_test)

0.36358755750788263

#Calulamos la metrica R^2 con los datos de entrenamiento
lre.score(x_train[['horsepower']], y_train)

0.6619724197515104
```

###**Pregunta 2:** Determinar R^2 sobre los datos de prueba utilizando el conjunto de datos de 40%,para prueba.

```
#Se obtiene con los conjuntos previamente divididos utilizando un
test_size de 0.4
lre.fit(x_train1[['horsepower']],y_train1)
lre.score(x_test1[['horsepower']], y_test1)

0.7139364665406973

#Solución presentada por IBM, no hay necesidad de la primera linea de
codigo.
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_data,
y_data, test_size=0.4, random_state=0)
lre.fit(x_train1[['horsepower']],y_train1)
lre.score(x_test1[['horsepower']],y_test1)

0.7139364665406973
```

#Validación Cruzada

```
#Importamos las librerias para realizar la validación cruzada
from sklearn.model_selection import cross_val_score

#Se lleva a cabo la validación cruzada
Rcross = cross_val_score(lre, x_data[['horsepower']], y_data, cv=4)

#El valor de cada elemeto del arreglo es el promedio de la metríca R^2
de
```

```
#cada pliegue de la funcion de validación cruzada
Rcross
```

```
array([0.7746232 , 0.51716687, 0.74785353, 0.04839605])
```

```
#Calculamos el promedio y la desviación estandar de nuestro estimado
print("The mean of the folds are", Rcross.mean(), "and the standard
deviation is" , Rcross.std())
```

```
The mean of the folds are 0.522009915042119 and the standard deviation
is 0.291183944475603
```

```
#Calculamos el error cuadratico promedio ajjustando el parámetro de
'scoring'
-1 * cross_val_score(lre,x_data[['horsepower']],
y_data,cv=4,scoring='neg_mean_squared_error')
```

```
array([20254142.84026702, 43745493.2650517 , 12539630.34014931,
       17561927.72247591])
```

###**Pregunta 3:** Determinar R^2 promedio utilizando 2 pliegues utilizando la caracteristica de "horsepower"

```
#Realizamos la validación cruzada con los parametros especifícados
Rc = cross_val_score(lre, x_data[['horsepower']],y_data,cv=2)
Rc.mean()
```

```
0.5166761697127429
```

```
#Cargamos la libreria que nos permite utilizar la función
"cross_val_predict"
from sklearn.model_selection import cross_val_predict
```

```
#Realizamos validación cruzada con sus respectivvas predicciones
yhat = cross_val_predict(lre,x_data[['horsepower']], y_data,cv=4)
yhat[0:5]
```

```
array([14141.63807508, 14141.63807508, 20814.29423473, 12745.03562306,
       14762.35027598])
```

#Sobre-entrenamiento, sub-entrenamiento y selección del modelo

```
#Creamos un objeto de Regresión Lineal Múltiple y realizamos el
entrenamiento
#Las variables a utilizar son: "horsepower", "curb-weight", "engine-
size" y "highway-mpg"
lr = LinearRegression()
lr.fit(x_train[['horsepower', 'curb-weight', 'engine-size', 'highway-
mpg']], y_train)
```

```
LinearRegression()
```

```
#Realizamos predicciones utilizando datos de entrenamiento
yhat_train = lr.predict(x_train[['horsepower', 'curb-weight', 'engine-
size', 'highway-mpg']])
yhat_train[0:5]
```

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

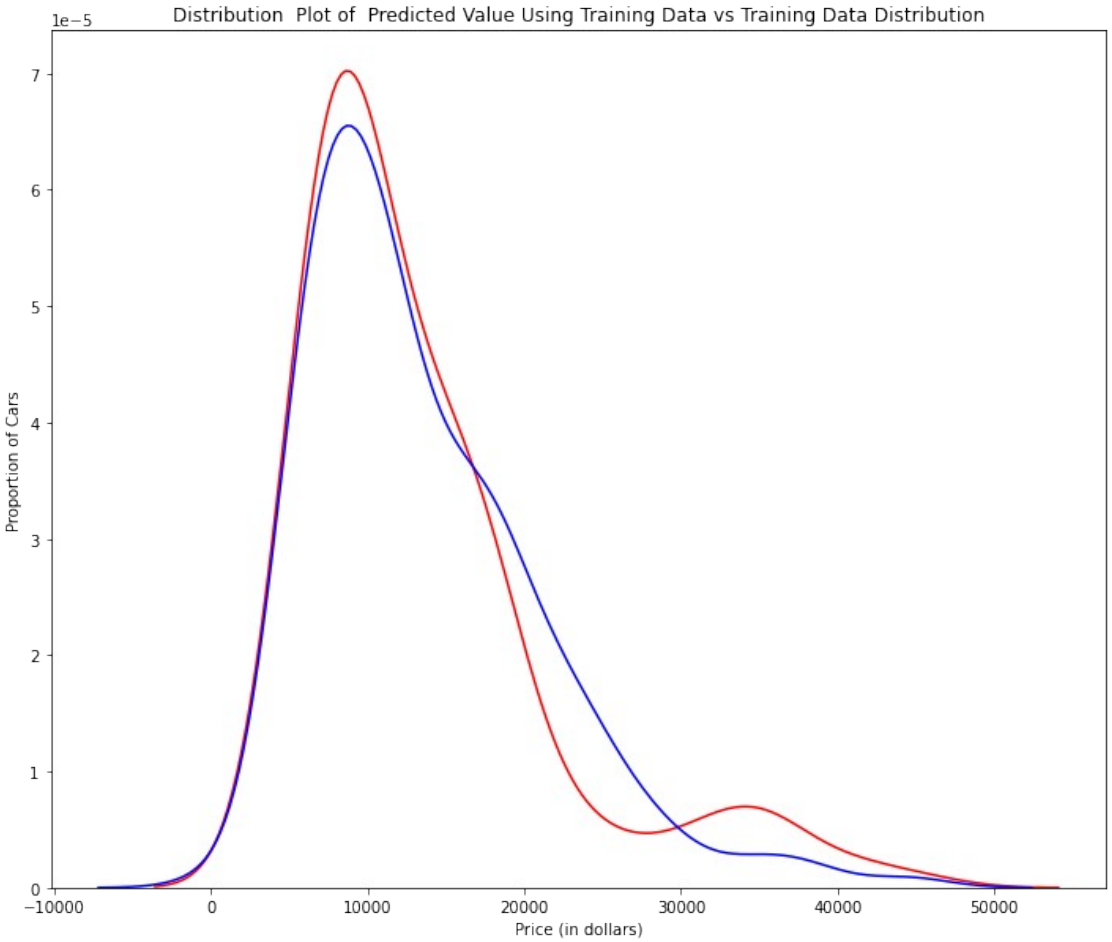
```
#Realizamos predicciones utilizanod datos de prueba
yhat_test = lr.predict(x_test[['horsepower', 'curb-weight', 'engine-
size', 'highway-mpg']])
yhat_test[0:5]
```

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

```
#Realizamos la evaluación del modelo utilizando los datos de
entrenamiento y prueba de forma separada.
#Se importan libreria para el graficado de los datos.
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

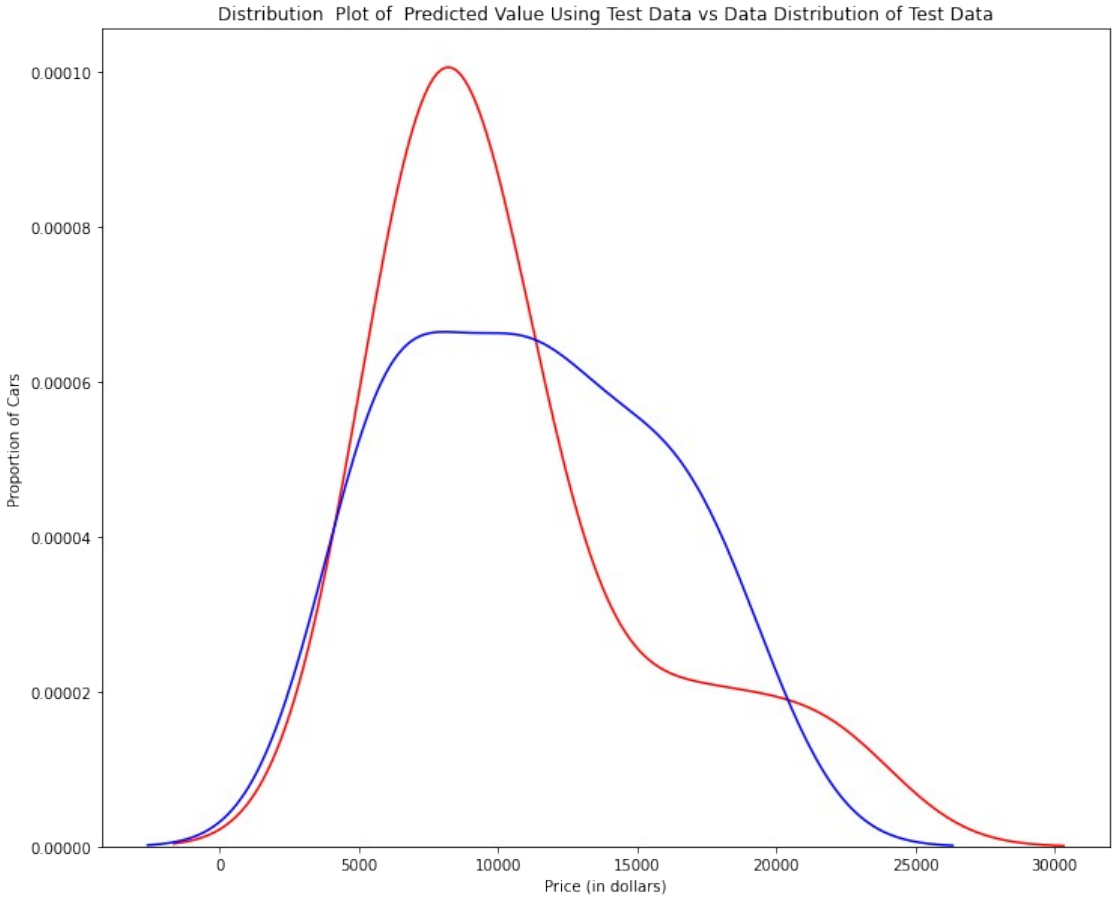
```
#Examinamos la distribución de los valor de prediccion de los daots de
entrenamiento.
Title = 'Distribution  Plot of  Predicted Value Using Training Data vs
Training Data Distribution'
DistributionPlot(y_train, yhat_train, "Actual Values (Train)",
"Predicted Values (Train)", Title)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. 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).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. 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).
  warnings.warn(msg, FutureWarning)
```




```
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)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. 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).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. 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).
  warnings.warn(msg, FutureWarning)
```



###Sobreentrenamiento

```
#Importamos una libreria para realizar transformacion polinomiales
from sklearn.preprocessing import PolynomialFeatures

#Utilizamos 55% de los datos para entrenamiento y el resto para prueba
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data,
test_size=0.45, random_state=0)

#Realizamos una transformación polinomial de grado5 sobre la variable
"horsepower"
pr = PolynomialFeatures(degree=5)
x_train_pr = pr.fit_transform(x_train[['horsepower']])
x_test_pr = pr.fit_transform(x_test[['horsepower']])
pr

PolynomialFeatures(degree=5)

#Creamos un objeto de Regrsion Lineal Polinomial y lo entrenamos
poly = LinearRegression()
poly.fit(x_train_pr, y_train)

LinearRegression()
```

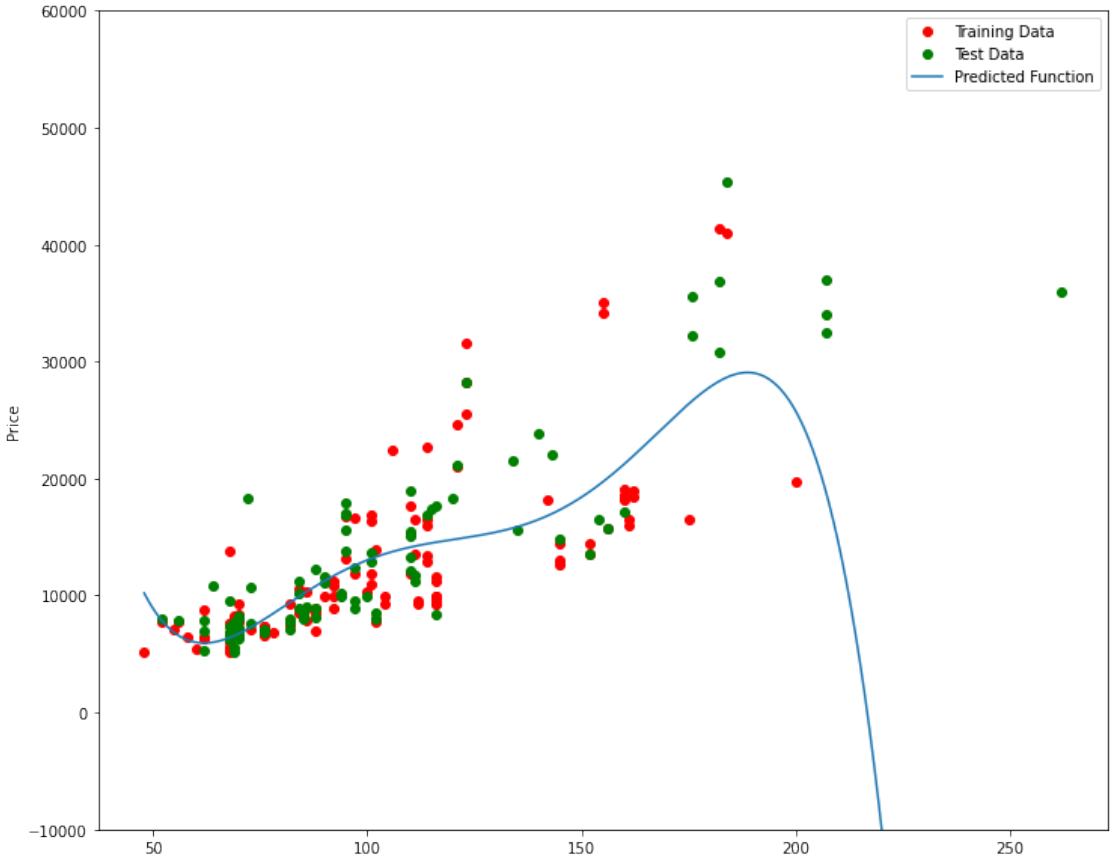
```
#Realizamos predicciones
yhat = poly.predict(x_test_pr)
yhat[0:5]
```

```
array([ 6728.65561887,  7307.98782321, 12213.78770965, 18893.24804015,
       19995.95195136])
```

```
#Observamos los primeros 5 valores obtenidos de la predicción y los
targets
print("Predicted values:", yhat[0:4])
print("True values:", y_test[0:4].values)
```

```
Predicted values: [ 6728.65561887  7307.98782321 12213.78770965
18893.24804015]
True values: [ 6295. 10698. 13860. 13499.]
```

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




```
#Obtenemos la metríca R^2 de ls datos de entrenamiento
poly.score(x_train_pr, y_train)
```

```
0.556771690212023
```

```
#Obtenemos la metríca R^2 de ls datos de prueba
poly.score(x_test_pr, y_test)
```

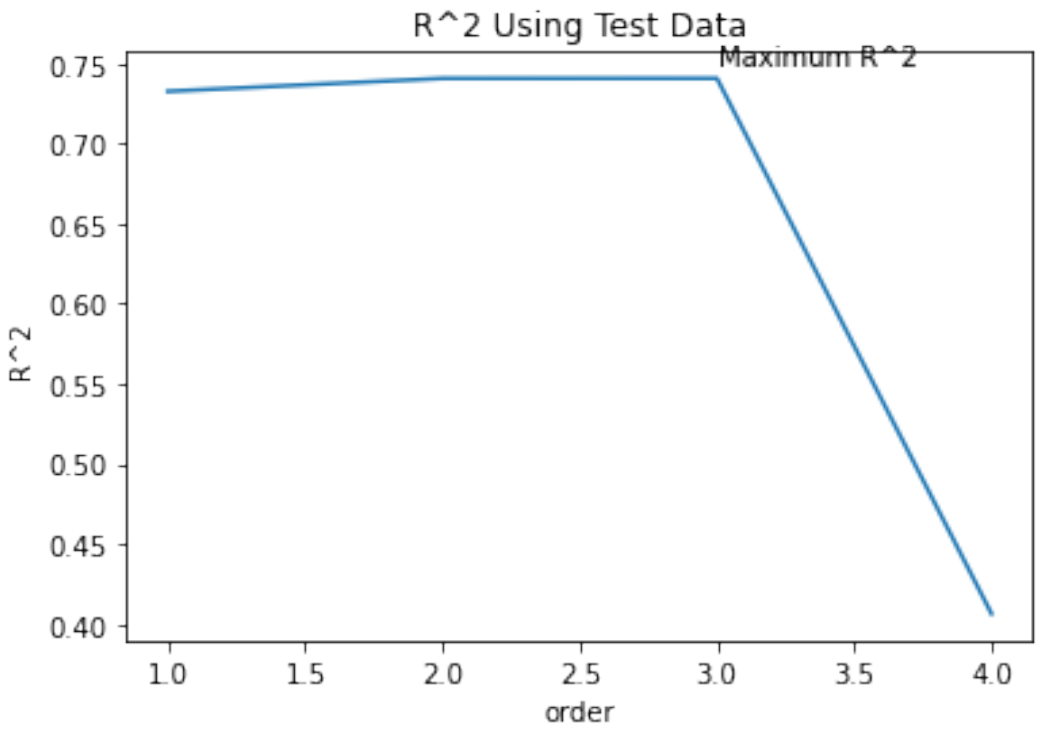
```
-29.87134030204415
```

El signo negativo representa sobre-entrenamiento.

###Observemos como la metríca R^2 cambia sobre los datos de entrenamiento para diferentes grados polinomiales

```
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 ')
```

```
Text(3, 0.75, 'Maximum R^2 ')
```



#Definimos una función

```
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)

interact(f, order=(0, 6, 1), test_data=(0.05, 0.95, 0.05))
```


```
{"version_major":2,"version_minor":0,"model_id":"97ea51ec995c4f68a4c57
94f617d1107"}
```


```
<function __main__.f(order, test_data)>
```

###**Pregunta 4-A:** Podemos realizar trnnasformaciones polinomiales con más de una característica. Crea un objeto "PolynominalFeatures" object "pr1"de 2 grados.

```
pr1 = PolynomialFeatures(degree=2)
```

###**Pregunta 4-B:** transformamos las muestras de entrenamiento y prueba para las variables "horsepower", "cub-weight", "engine-size" y "highway-mpg". Utiliza el método "fit_transform".

```
x_train_pr1=pr1.fit_transform(x_train[['horsepower', 'curb-weight',
'engine-size', 'highway-mpg']])

x_test_pr1=pr1.fit_transform(x_test[['horsepower', 'curb-weight',
'engine-size', 'highway-mpg']])
```

###**Pregunta 4-C:** Cuandas dimensiones tiene las nuevas caracteristicas. Utiliza el atributo "shape

```
x_train_pr1.shape
```

```
(110, 15)
```

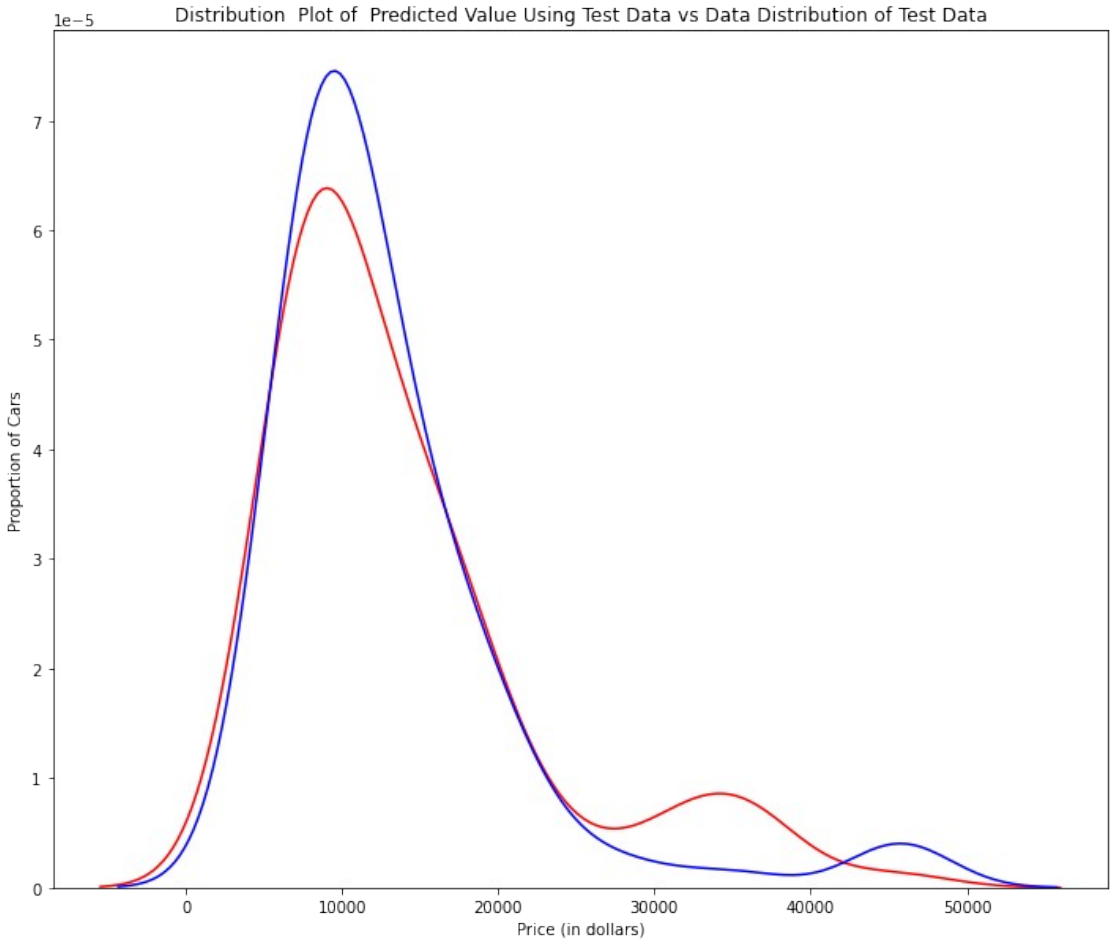
###**Pregunta 4-D:** Crea un modelos de regresión lineal "poly1". Entrena el modelo utilizando el metodo "fit" utilizando las caracteristicas polinomiales.

```
poly1 =LinearRegression().fit(x_train_pr1,y_train)
```

###**Pregunta 4-E:** Utiliza el método "predic", para predecir la salida de las caracteíisticas polinomiales, después utiliza la funcion "DistributionPlot" para desplegar la distribució de prediccion de la salida de prueba contra los datos de prueba actuales.

```
yhat_test1=poly1.predict(x_test_pr1)

Title='Distribution  Plot of  Predicted Value Using Test Data vs Data
Distribution of Test Data'

DistributionPlot(y_test, yhat_test1, "Actual Values (Test)",
"Predicted Values (Test)", Title)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. 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).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. 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).
  warnings.warn(msg, FutureWarning)
```



###**Pregunta 4-F:** Utilizando la grafica de distribucion anterior, describe en palabras, las dos regiones donde las predicciones de los precios son menos precisos que los precios actuales.

```
#Los valores de predicción se encuentran más altos que los valores
actuales en el rano de 10,000,
#se observa en la grafica que en el rango de 25,000 a 40,000 que los
valores de predicción son menores
#que los valores actuales. Por lo tanto el modelo obtenido no es tan
exacto dentro de estos rangos.
```

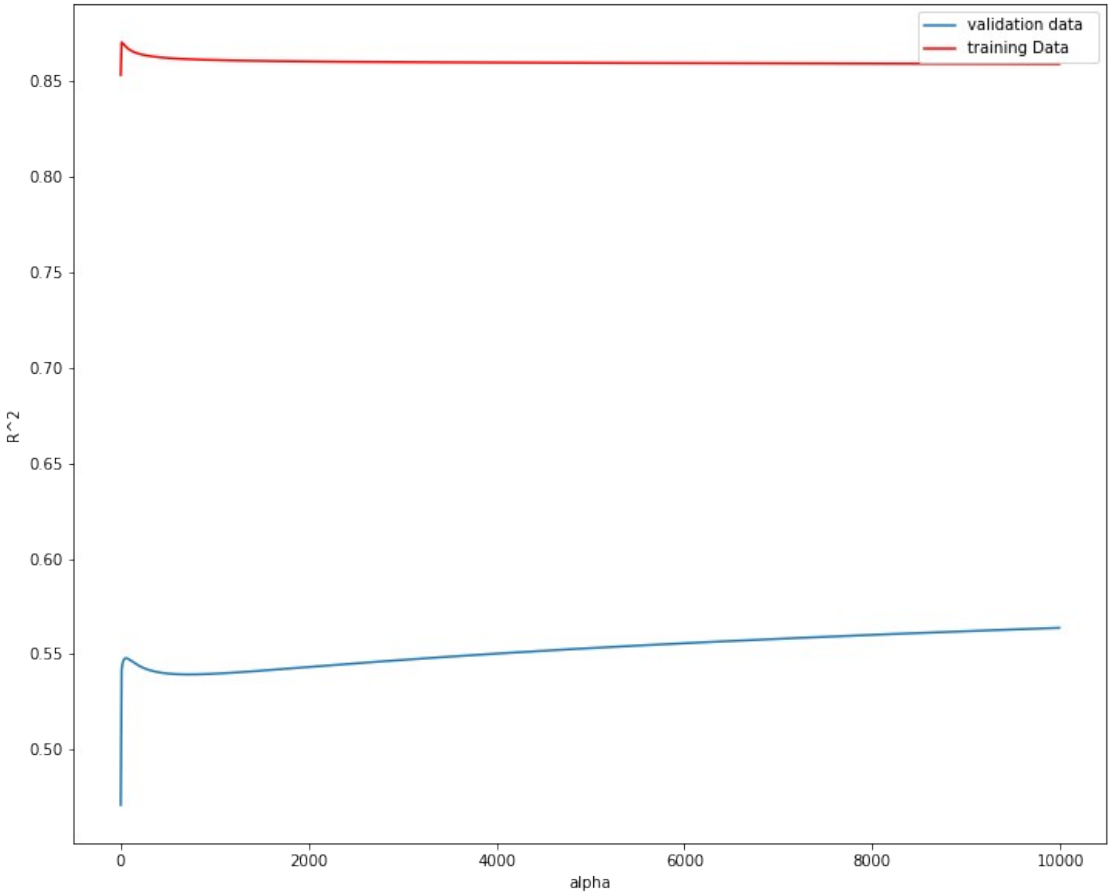
##Parte 3: Regresión Ridge

```
#Transformamos los datos
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']])

#Importamos la libreria Ridge del modulo de modelos lineales
from sklearn.linear_model import Ridge
```

```
#Creamos el objeto Ridge
RigeModel=Ridge(alpha=1)

#Ajustamos el modelo
RigeModel.fit(x_train_pr, y_train)

Ridge(alpha=1)

#Obtenemos una predicción
yhat = RigeModel.predict(x_test_pr)

#Comparamos las predicciones con los datos de prueba
print('predicted:', yhat[0:4])
print('test set :', y_test[0:4].values)

predicted: [ 6570.82441941  9636.2489147  20949.92322737
19403.60313256]
test set : [ 6295. 10698. 13860. 13499.]

#Seleccionamos el valor adecuado de alpha
#de tal forma que se minimize el error
#de prueba
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 [00:05<00:00, 173.46it/s, Test Score=0.564,
Train Score=0.859]

#Imprimimos el valor R^2 para diferentes alphas
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()
```

```
<matplotlib.legend.Legend at 0x7f901eb69390>
```



###**Pregunta 5:** Reallizauna regresión Ridge, calcula R^2 utilizando las características polinomiales, utiliza los datos de entrenamiento del modelo y los datos de prueba en el modelo. El parametro alpha se debe de ajustar en 10.

```
RigeModel = Ridge(alpha=10)
RigeModel.fit(x_train_pr, y_train)
RigeModel.score(x_test_pr, y_test)
```

```
0.5418576440206702
```

##Parte 4: Busqueda de malla.

```
#Importamos el modulo GridSearchCV
from sklearn.model_selection import GridSearchCV

#Creamos el diccionario de los parametros
parameters1= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000,
```

```
100000]}]
parameters1
```

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

```
#Creamos el objeto de regresion Ridge
RR=Ridge()
RR
```

```
Ridge()
```

```
#Creamos el objeto de busqueda Grid
Grid1 = GridSearchCV(RR, parameters1,cv=4)
```

```
#Realizamos el ajuste del modelo
Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size',
'highway-mpg']], y_data)
```

```
GridSearchCV(cv=4, estimator=Ridge(),
             param_grid=[{'alpha': [0.001, 0.1, 1, 10, 100, 1000,
10000, 100000,
                                    100000]}])
```

```
#El objeto encuentra los mejores parametros en los datos de validación
BestRR=Grid1.best_estimator_
BestRR
```

```
Ridge(alpha=10000)
```

```
#Probamos el modelos sobre los datos de prueba
BestRR.score(x_test[['horsepower', 'curb-weight', 'engine-size',
'highway-mpg']], y_test)
```

```
0.8411649831036151
```