#**Maestría en Inteligencia Artificial Aplicada**

##**Curso: Ciencia y analítica de datos**

###**Tecnológico de Monterrey**

###**PhD. María de la Paz Rico Fdz**

## Actividad Semanal -- 7

###**Regresiones y K means.**

###**Fecha de entrega: 09/11/2022.**


## Alumno: Maximiliano Morones Gómez

## Matrícula: A01793815


# Linear Models

Este criterio depende de una competencia de aprendizaje

Ejercicio 1. Costo en la industria de manufactura.

- Realiza división de datos
- Realiza modelos de regresion lineal, polinomial, ridge y lasso. -Cada modelo contiene su ecuación, error y r cuadrada.
- Se incluye graficas de: MAE (de los cuatro métodos) R2 (de los cuatro métodos)

Brinda explicación de resultados y conclusiones del experimento a través de las preguntas planteadas en la actividad.

- In supervised learning, the training data fed to the algorithm includes the desired solutions, called labels.
- In **regression**, the labels are continuous quantities.
- Linear models predict by computing a weighted sum of input features plus a bias term.

```
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV,
train_test_split, RepeatedKFold, cross_validate
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import power_transform
from sklearn.model_selection import RepeatedKFold,
RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV,
train_test_split, RepeatedKFold, cross_validate
# to make this notebook's output stable across runs
np.random.seed(42)
```

```
5-2
```

```
3
```

## Simple Linear Regression

Simple linear regression equation:

$y=ax+b$ $a$: slope $b$: intercept

Generate linear-looking data with the equation:

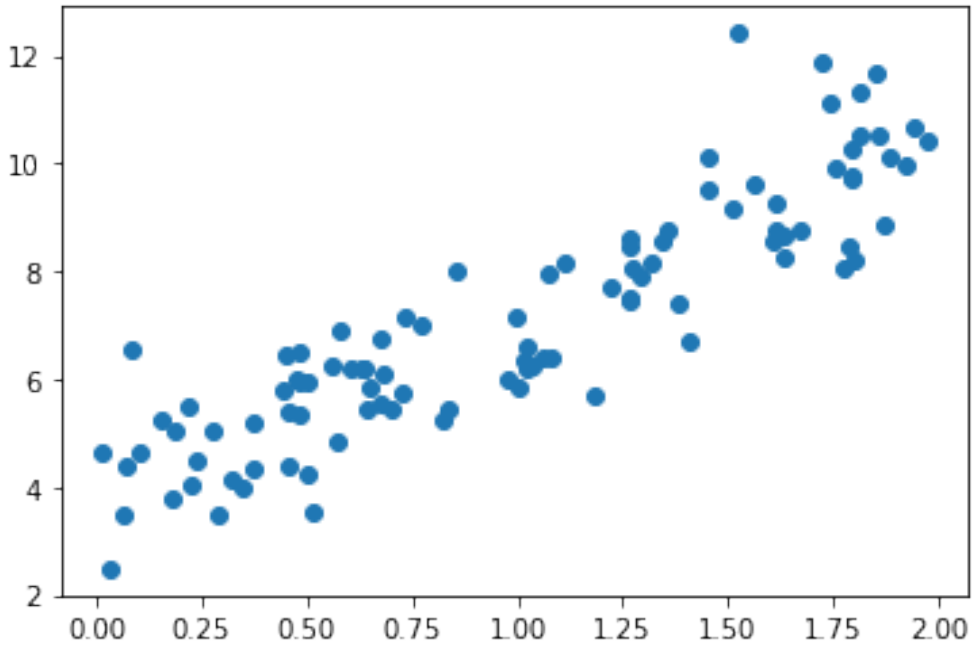
$y=3X+4+noise$

```
np.random.rand(100, 1)
```

```
array([[0.37454012],
       [0.95071431],
       [0.73199394],
       [0.59865848],
       [0.15601864],
       [0.15599452],
       [0.05808361],
       [0.86617615],
       [0.60111501],
       [0.70807258],
       [0.02058449],
       [0.96990985],
       [0.83244264],
       [0.21233911],
       [0.18182497],
       [0.18340451],
       [0.30424224],
       [0.52475643],
       [0.43194502],
       [0.29122914],
```

[0.61185289],
[0.13949386],
[0.29214465],
[0.36636184],
[0.45606998],
[0.78517596],
[0.19967378],
[0.51423444],
[0.59241457],
[0.04645041],
[0.60754485],
[0.17052412],
[0.06505159],
[0.94888554],
[0.96563203],
[0.80839735],
[0.30461377],
[0.09767211],
[0.68423303],
[0.44015249],
[0.12203823],
[0.49517691],
[0.03438852],
[0.9093204 ],
[0.25877998],
[0.66252228],
[0.31171108],
[0.52006802],
[0.54671028],
[0.18485446],
[0.96958463],
[0.77513282],
[0.93949894],
[0.89482735],
[0.59789998],
[0.92187424],
[0.0884925 ],
[0.19598286],
[0.04522729],
[0.32533033],
[0.38867729],
[0.27134903],
[0.82873751],
[0.35675333],
[0.28093451],
[0.54269608],
[0.14092422],
[0.80219698],
[0.07455064],
[0.98688694],

```
        [0.77224477],
        [0.19871568],
        [0.00552212],
        [0.81546143],
        [0.70685734],
        [0.72900717],
        [0.77127035],
        [0.07404465],
        [0.35846573],
        [0.11586906],
        [0.86310343],
        [0.62329813],
        [0.33089802],
        [0.06355835],
        [0.31098232],
        [0.32518332],
        [0.72960618],
        [0.63755747],
        [0.88721274],
        [0.47221493],
        [0.11959425],
        [0.71324479],
        [0.76078505],
        [0.5612772 ],
        [0.77096718],
        [0.4937956 ],
        [0.52273283],
        [0.42754102],
        [0.02541913],
        [0.10789143]])

X = 2*np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
plt.scatter(X, y);
```




```
import pandas as pd
pd.DataFrame(y)
```

```
             0
0    3.508550
1    8.050716
2    6.179208
3    6.337073
4   11.311173
..        ...
95   5.441928
96  10.121188
97   9.787643
98   8.061635
99   9.597115

[100 rows x 1 columns]
```

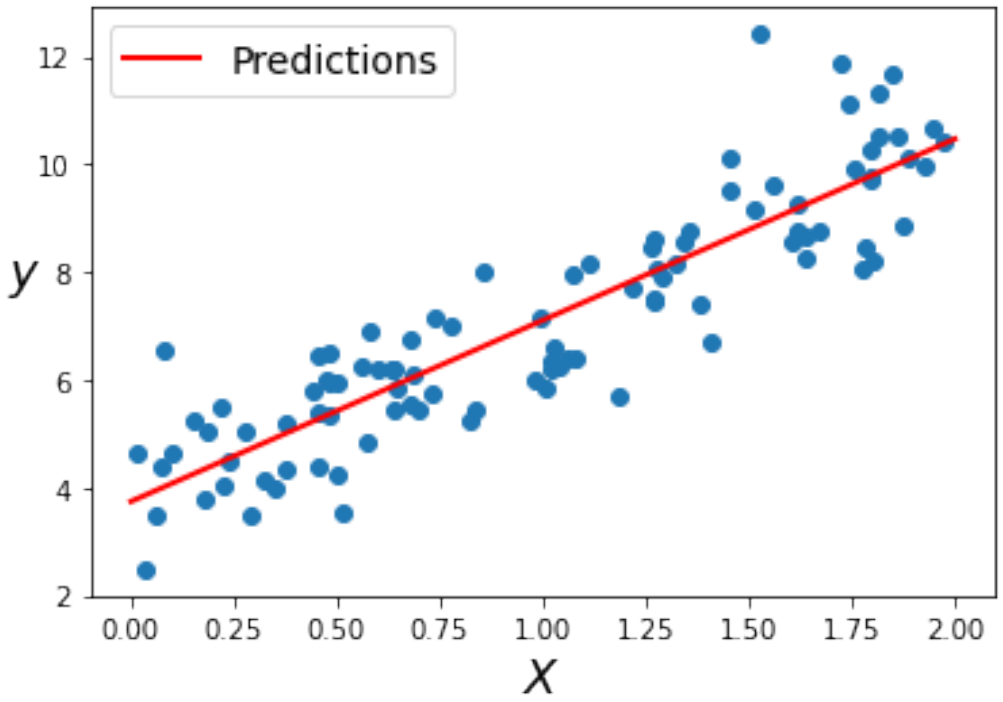
```
from sklearn.linear_model import LinearRegression

linear_reg = LinearRegression(fit_intercept=True)
linear_reg.fit(X, y)
```

```
LinearRegression()
```

Plot the model's predictions:

```
#X_fit[]

# construct best fit line
X_fit = np.linspace(0, 2, 100)
```

```
y_fit = linear_reg.predict(X_fit[:, np.newaxis])

plt.scatter(X, y)
plt.plot(X_fit, y_fit, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```




Predictions are a good fit.

Generate new data to make predictions with the model:

```
X_new = np.array([[0], [2]])
X_new
```

```
array([[0],
       [2]])
```

```
X_new.shape
```

```
(2, 1)
```

```
y_new = linear_reg.predict(X_new)
y_new
```

```
array([[ 3.74406122],
       [10.47517611]])
```

```
 linear_reg.coef_, linear_reg.intercept_
```

```
(array([[3.36555744]]), array([3.74406122]))
```

The model estimates:

$\hat{y} = 3.36\,X + 3.74$

```
#|VENTAS|GANANCIAS|
#COEF*VENTAS+B
#|VENTAS|COMPRAS|GANANCIAS|
#COEF1*X1+COEF2*X2+B=Y
```

## Polynomial Regression

If data is more complex than a straight line, you can use a linear model ti fit non-linear data adding powers of each feature as new features and then train a linear model on the extended set of features.

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + \ldots$$

to

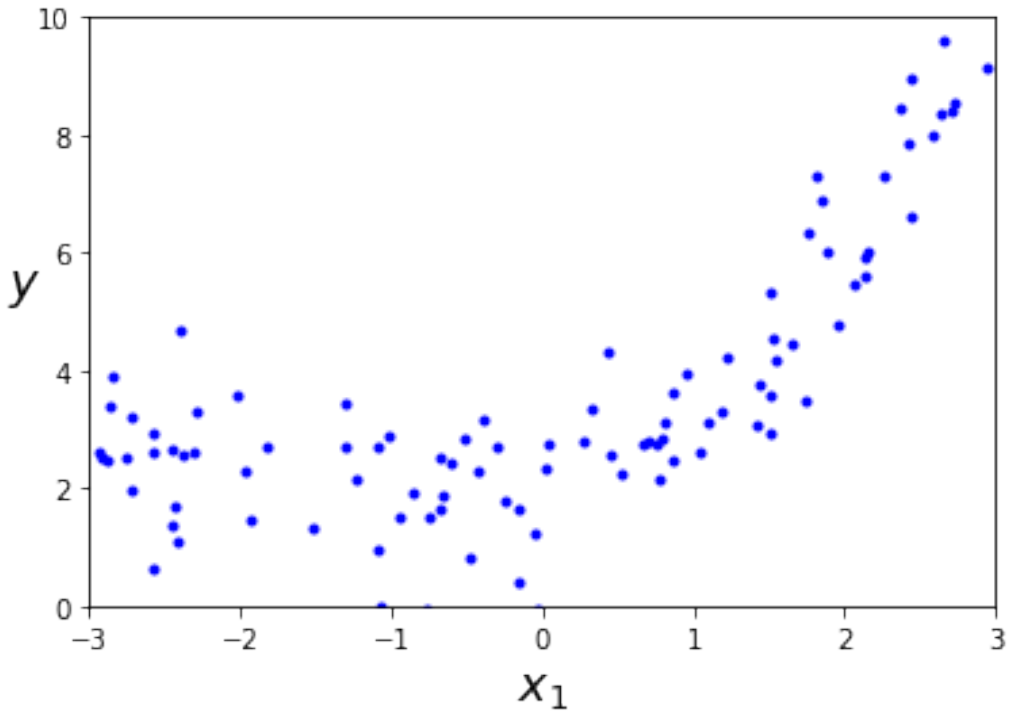
$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots$$

This is still a linear model, the linearity refers to the fact that the coefficients never multiply or divide each other.

To generate polynomial data we use the function:

$y = 0.50\,X^2 + X + 2 + noise$

```
# generate non-linear data e.g. quadratic equation
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)

plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([-3, 3, 0, 10]);
```




```
import pandas as pd
pd.DataFrame(y)
```

```
           0
0   8.529240
1   3.768929
2   3.354423
3   2.747935
4   0.808458
..       ...
95  5.346771
96  6.338229
97  3.488785
98  1.372002
99 -0.072150

[100 rows x 1 columns]
```

Now we can use `PolynomialFeatues` to transform training data adding the square of each feature as new features.

```
from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)

X_poly
```

```
array([[ 2.72919168e+00,  7.44848725e+00],
       [ 1.42738150e+00,  2.03741795e+00],
       [ 3.26124315e-01,  1.06357069e-01],
       [ 6.70324477e-01,  4.49334905e-01],
       [-4.82399625e-01,  2.32709399e-01],
       [-1.51361406e+00,  2.29102753e+00],
       [-8.64163928e-01,  7.46779295e-01],
       [ 1.54707666e+00,  2.39344620e+00],
       [-2.91363907e+00,  8.48929262e+00],
       [-2.30356416e+00,  5.30640783e+00],
       [-2.72398415e+00,  7.42008964e+00],
       [-2.75562719e+00,  7.59348119e+00],
       [ 2.13276350e+00,  4.54868016e+00],
       [ 1.22194716e+00,  1.49315485e+00],
       [-1.54957025e-01,  2.40116797e-02],
       [-2.41299504e+00,  5.82254504e+00],
       [-5.03047493e-02,  2.53056780e-03],
       [-1.59169375e-01,  2.53348900e-02],
       [-1.96078878e+00,  3.84469264e+00],
       [-3.96890105e-01,  1.57521755e-01],
       [-6.08971594e-01,  3.70846402e-01],
       [ 6.95100588e-01,  4.83164828e-01],
       [ 8.10561905e-01,  6.57010602e-01],
       [-2.72817594e+00,  7.44294397e+00],
       [-7.52324312e-01,  5.65991871e-01],
       [ 7.55159494e-01,  5.70265862e-01],
       [ 1.88175515e-02,  3.54100244e-04],
       [ 2.13893905e+00,  4.57506025e+00],
       [ 9.52161790e-01,  9.06612074e-01],
       [-2.02239344e+00,  4.09007522e+00],
       [-2.57658752e+00,  6.63880323e+00],
       [ 8.54515669e-01,  7.30197029e-01],
       [-2.84093214e+00,  8.07089541e+00],
       [ 5.14653488e-01,  2.64868212e-01],
       [ 2.64138145e+00,  6.97689596e+00],
       [ 4.52845067e-01,  2.05068655e-01],
       [-6.70980443e-01,  4.50214755e-01],
       [ 8.59729311e-01,  7.39134488e-01],
       [-2.50482657e-01,  6.27415615e-02],
       [ 2.73700736e-01,  7.49120928e-02],
       [ 2.64878885e+00,  7.01608239e+00],
       [-6.83384173e-01,  4.67013928e-01],
       [ 2.76714338e+00,  7.65708250e+00],
       [ 2.43210385e+00,  5.91512915e+00],
       [-1.82525319e+00,  3.33154921e+00],
       [-2.58383219e+00,  6.67618881e+00],
       [-2.39533199e+00,  5.73761535e+00],
       [-2.89066905e+00,  8.35596753e+00],
       [-2.43334224e+00,  5.92115443e+00],
       [ 1.09804064e+00,  1.20569325e+00],
```

```
[-2.57286811e+00,  6.61965031e+00],
[-1.08614622e+00,  1.17971361e+00],
[ 2.06925187e+00,  4.28180328e+00],
[-2.86036839e+00,  8.18170730e+00],
[ 1.88681090e+00,  3.56005536e+00],
[-1.30887135e+00,  1.71314421e+00],
[-2.29101103e+00,  5.24873156e+00],
[ 1.18042299e+00,  1.39339844e+00],
[ 7.73657081e-01,  5.98545278e-01],
[ 2.26483208e+00,  5.12946436e+00],
[ 1.41042626e+00,  1.98930224e+00],
[ 1.82088558e+00,  3.31562430e+00],
[-1.30779256e+00,  1.71032139e+00],
[-1.93536274e+00,  3.74562893e+00],
[ 1.50368851e+00,  2.26107913e+00],
[ 1.84100844e+00,  3.38931206e+00],
[ 2.94303085e+00,  8.66143060e+00],
[-5.24293939e-01,  2.74884134e-01],
[-7.67891485e-01,  5.89657333e-01],
[ 1.65847776e+00,  2.75054850e+00],
[-9.55178758e-01,  9.12366461e-01],
[ 2.58454395e+00,  6.67986745e+00],
[ 2.15047651e+00,  4.62454922e+00],
[-4.26035836e-01,  1.81506533e-01],
[ 1.50522641e+00,  2.26570654e+00],
[ 1.52725724e+00,  2.33251469e+00],
[-2.38125679e+00,  5.67038389e+00],
[ 2.41531744e+00,  5.83375834e+00],
[ 3.15142347e-02,  9.93146988e-04],
[ 1.95874480e+00,  3.83668118e+00],
[-1.07970239e+00,  1.16575726e+00],
[ 2.37313937e+00,  5.63179047e+00],
[-6.64789928e-01,  4.41945648e-01],
[-2.93497409e+00,  8.61407292e+00],
[ 2.43229186e+00,  5.91604369e+00],
[-2.45227994e+00,  6.01367690e+00],
[-1.08411817e+00,  1.17531222e+00],
[ 2.70037180e+00,  7.29200787e+00],
[ 2.70364288e+00,  7.30968483e+00],
[ 4.40627329e-01,  1.94152443e-01],
[ 7.91023273e-01,  6.25717818e-01],
[-3.09326868e-01,  9.56831113e-02],
[-1.24073537e+00,  1.53942426e+00],
[-1.02801273e+00,  1.05681017e+00],
[ 1.03511074e+00,  1.07145424e+00],
[ 1.51424718e+00,  2.29294451e+00],
[ 1.74947426e+00,  3.06066019e+00],
[ 1.73770886e+00,  3.01963207e+00],
[-2.45276338e+00,  6.01604821e+00],
[-3.34781718e-02,  1.12078799e-03]])
```

X_poly now contains the original feature of X plus the square of the feature:

```
print(X[0])
print(X[0]*X[0])
```

```
[2.72919168]
[7.44848725]
```

```
X_poly[0]
```

```
array([2.72919168, 7.44848725])
```

Fit the model to this extended training data:

```
lin_reg = LinearRegression(fit_intercept=True)
lin_reg.fit(X_poly, y)
lin_reg.coef_, lin_reg.intercept_
```

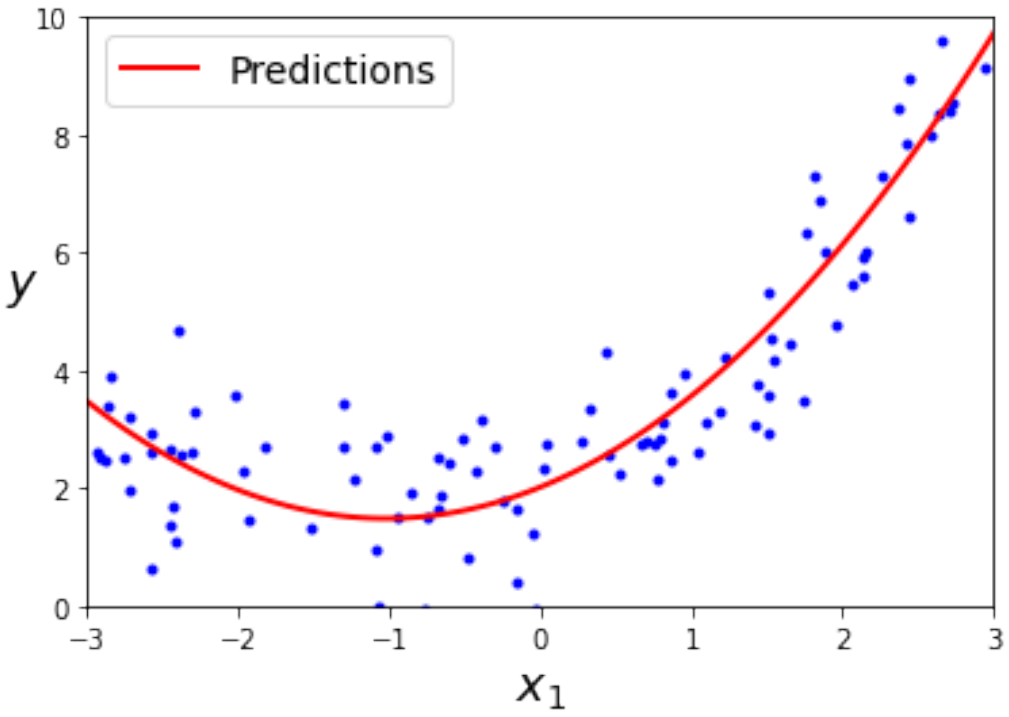
```
(array([[1.04271531, 0.50866711]]), array([2.01873554]))
```

The model estimates:

$\hat{y} = 0.89\,X + 0.48\,X^2 + 2.09$

Plot the data and the predictions:

```
X_new=np.linspace(-3, 3, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)
plt.plot(X, y, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([-3, 3, 0, 10]);
```

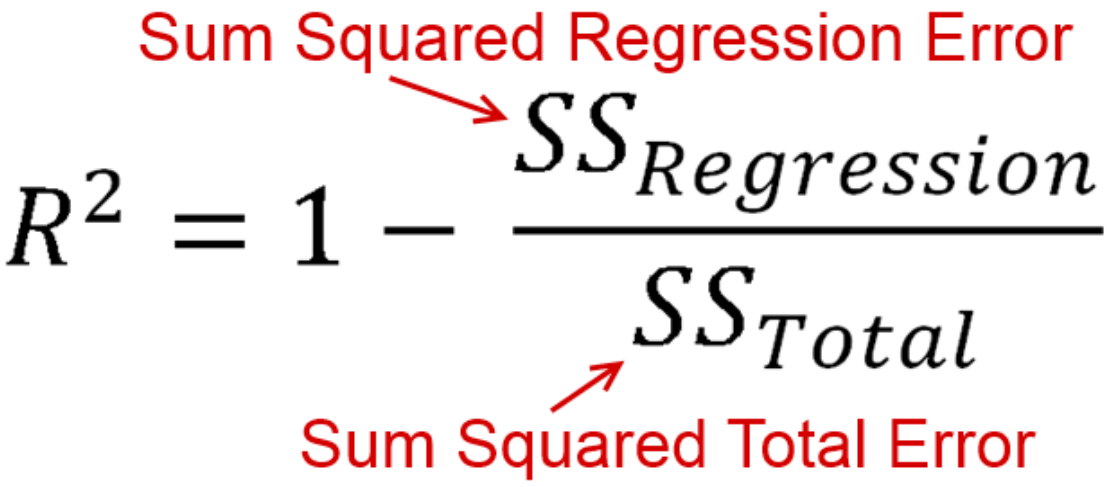



## R square

$R^2$ es una medida estadística de qué tan cerca están los datos de la línea de regresión ajustada. También se conoce como el coeficiente de determinación o el coeficiente de determinación múltiple para la regresión múltiple. Para decirlo en un lenguaje más simple, $R^2$ es una medida de ajuste para los modelos de regresión lineal.

$R^2$ no indica si un modelo de regresión se ajusta adecuadamente a sus datos. Un buen modelo puede tener un valor $R^2$ bajo. Por otro lado, un modelo sesgado puede tener un valor alto de $R^2$.

SSres + SSreg = SStot, $R^2$ = Explained variation / Total Variation




$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \Rightarrow 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

$$R^2 = \frac{SS_{\text{reg}}}{SS_{\text{tot}}}$$

## Ejercicio 1

Utiliza la base de datos de https://www.kaggle.com/vinicius150987/manufacturing-cost

Suponga que trabaja como consultor de una empresa de nueva creación que busca desarrollar un modelo para estimar el costo de los bienes vendidos a medida que varían el volumen de producción (número de unidades producidas). La startup recopiló datos y le pidió que desarrollara un modelo para predecir su costo frente a la cantidad de unidades vendidas.

```
import pandas as pd
df =
pd.read_csv('https://raw.githubusercontent.com/marypazrf/bdd/main/
EconomiesOfScale.csv')
df.sample(10)
```

| | Number of Units | Manufacturing Cost |
|---|---|---|
| 93 | 2.756427 | 46.838555 |
| 693 | 5.111410 | 30.448430 |
| 100 | 2.769290 | 51.463569 |
| 558 | 4.620193 | 41.924336 |
| 973 | 7.169644 | 25.311833 |

```
990          7.900366           27.595130
393          4.118062           45.308239
514          4.486535           30.321302
485          4.392126           38.259120
264          3.661754           33.937080
```

```
X = df[['Number of Units']]
y = df['Manufacturing Cost']
```

```
len(X)
```

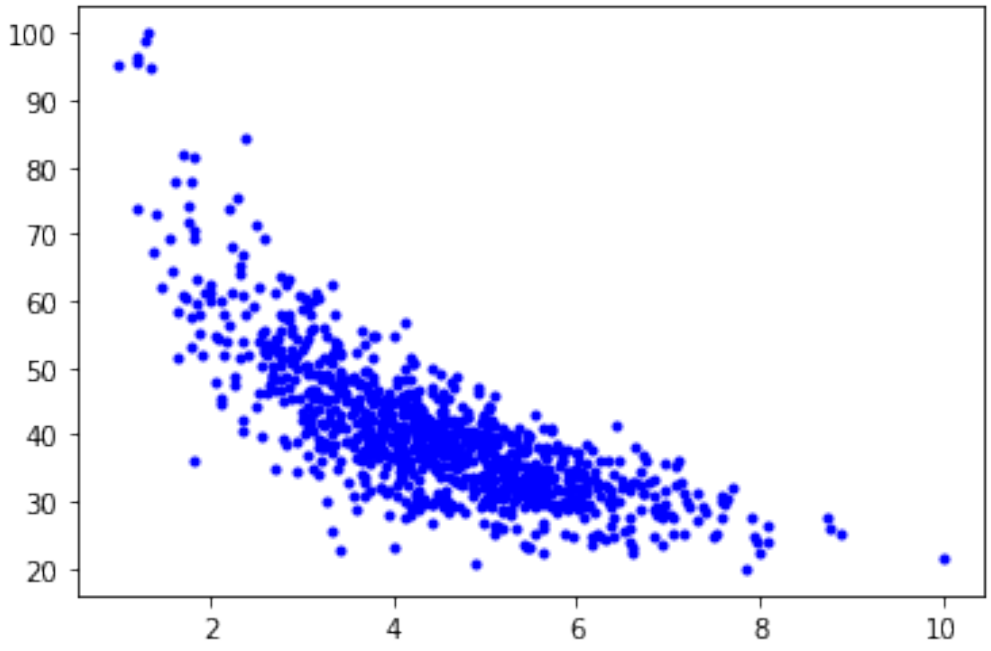
```
1000
```

```
y.describe
```

```
<bound method NDFrame.describe of 0      95.066056
1      96.531750
2      73.661311
3      95.566843
4      98.777013
         ...
995    23.855067
996    27.536542
997    25.973787
998    25.138311
999    21.547777
Name: Manufacturing Cost, Length: 1000, dtype: float64>
```

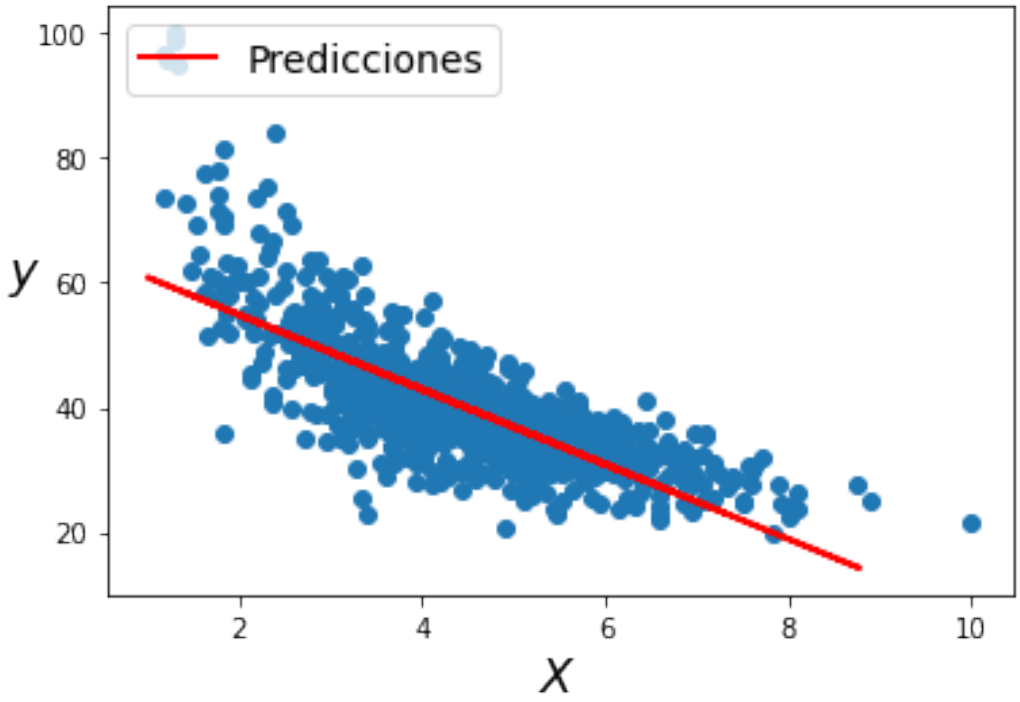
```
plt.plot(X,y,'b.')
```

```
[<matplotlib.lines.Line2D at 0x7fc972136f10>]
```



```
#lineal
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state = 101)
lista_para_mae =[]
lista_para_r2 =[]

#Visualización correspondiente al modelo de regresión lineal.
linear_reg = LinearRegression(fit_intercept=True)
linear_reg.fit(X_train, y_train)
X_para_regresion = X_test  #Aqui recordemos que se hace con los de
prueba, no con los de entrenamiento
y_para_regresion = linear_reg.predict(X_para_regresion)

#Realizamos el ploteo correspondiente al modelo de regresión lineal.
plt.scatter(X_train, y_train)
plt.plot(X_para_regresion, y_para_regresion, "r-", linewidth=2,
label="Predicciones")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);

#Ecuación para el modelo de regresión lineal.
linear_reg.coef_, linear_reg.intercept_
mae_regresion_lineal_simple =
metrics.mean_absolute_error(y_test,y_para_regresion)
lista_para_mae.append(mae_regresion_lineal_simple)
r2_regresion_lineal_simple = r2_score(y_test,y_para_regresion)
lista_para_r2.append(r2_regresion_lineal_simple)

#Descripción del modelo de regresión lineal.
print('El modelo es: Y =', linear_reg.coef_,'X +',
linear_reg.intercept_)

#Errores del modelo de regresión lineal.
print('Error Medio Absoluto (MAE):',
metrics.mean_absolute_error(y_test,y_para_regresion))
print('Error Medio Cuadrado (RMSE):',
np.sqrt(metrics.mean_squared_error(y_test, y_para_regresion)))

#R cuadrada del modelo de regresión lineal.
print('r2_score:', r2_score(y_test,y_para_regresion))
```

```
El modelo es: Y = [-5.98882699] X + 66.83650741226988
Error Medio Absoluto (MAE): 5.013587781954963
Error Medio Cuadrado (RMSE): 7.108963321847682
r2_score: 0.6116251549562579
```




```
#Polinomial
'''pr = PolynomialFeatures(degree=2, include_bias=False)
X_train_pr = pr.fit_transform(X_train)
X_test_pr = pr.fit_transform(X_test)
nueva_regresion = LinearRegression()
nueva_regresion.fit(X_train_pr, y_train)
nueva_regresion.intercept_, nueva_regresion.coef_'''
```

```
{"type":"string"}
```

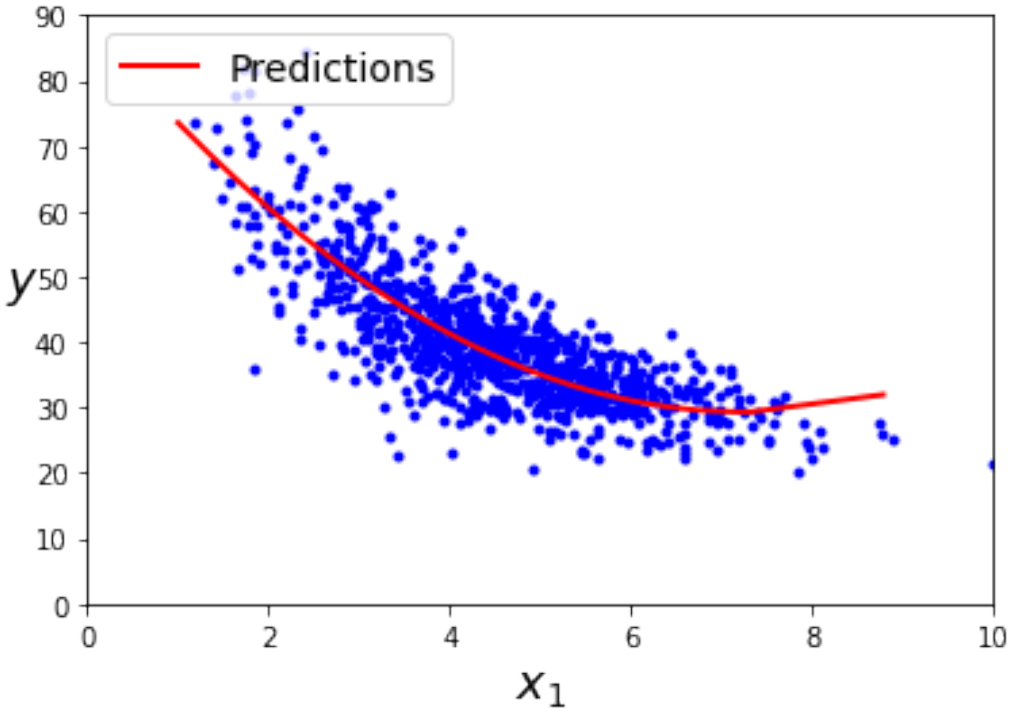
```
#Creación de las características y asignación de un grado de nivel
dos, esto se debe a que se desea ajustar a un polinomio de segundo
grado.
caracteristicas_para_poly = PolynomialFeatures(degree=2,
include_bias=False) #Nuestro datos originales se elevaran al cuadrado.
X_polinomial = caracteristicas_para_poly.fit_transform(X_train)
print("Input",caracteristicas_para_poly.n_input_features_)
print("Ouput",caracteristicas_para_poly.n_output_features_)
print("Powersn",caracteristicas_para_poly.powers_)

regresion_lineal_poli = LinearRegression(fit_intercept=True)
regresion_lineal_poli.fit(X_polinomial, y_train)
regresion_lineal_poli.coef_, regresion_lineal_poli.intercept_
```

```
Input 1
Ouput 2
Powersn [[1]
 [2]]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/
deprecation.py:103: FutureWarning: The attribute `n_input_features_`
was deprecated in version 1.0 and will be removed in 1.2.
  warnings.warn(msg, category=FutureWarning)

(array([-16.40638102,   1.13136095]), 88.80179909112496)

X_polinomial.shape

(900, 2)

X_polinomial_test = caracteristicas_para_poly.fit_transform(X_test)
X_polinomial_test.shape

(100, 2)

y_con_regresion_poli =
regresion_lineal_poli.predict(X_polinomial_test)
y_con_regresion_poli.shape

(100,)

#Visualización correspondiente al modelo de regresión polinomial.
order = np.argsort(X_test.values.ravel()) #Obtenemos el orden que
deben de tener cada uno de los datos.

#Ordenamos cada uno de los datos en el orden adecuado.
sortedXPoly = X_test.values.ravel()[order]
sortedYPoly = y_test.values.ravel()[order]
sorted_predicPoly = y_con_regresion_poli[order]

#Realizamos el ploteo correspondiente al modelo de regresión
polinomial.
plt.plot(X, y, "b.")
plt.plot(sortedXPoly, sorted_predicPoly, "r-", linewidth=2,
label="Predictions")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 10, 0, 90]);

#Descripción del modelo de regresión polinomial.
print('El modelo es: Y =', regresion_lineal_poli.coef_[1],'X^2 +',
regresion_lineal_poli.coef_[0],'X +',regresion_lineal_poli.intercept_)
mae_regresion_lineal_multiple =
metrics.mean_absolute_error(y_test,y_con_regresion_poli)
lista_para_mae.append(mae_regresion_lineal_multiple)
r2_regresion_lineal_multiple = r2_score(y_test,y_con_regresion_poli)
lista_para_r2.append(r2_regresion_lineal_multiple)
metrica_mae = metrics.mean_absolute_error(y_test,
y_con_regresion_poli)
r2Score = r2_score(y_test, y_con_regresion_poli)
```

```
#Errores del modelo de regresión polinomial.
print('Error medio Absoluto (MAE):', metrica_mae)
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_con_regresion_poli)))

#R cuadrada del modelo de regresión polinomial.
print('r2_score',r2Score)
```

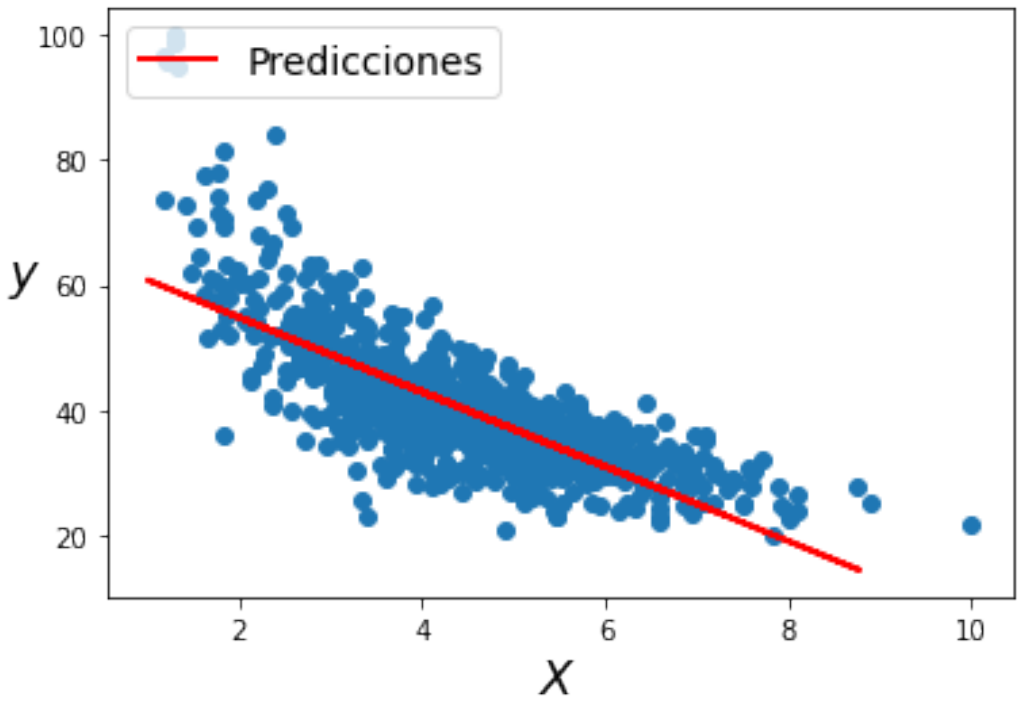
```
El modelo es: Y = 1.1313609537119216 X^2 + -16.406381017212386 X +
88.80179909112496
Error medio Absoluto (MAE): 4.3833025759681075
Root Mean Squared Error: 5.832771301068423
r2_score 0.7385501224942537
```




```
#Ridge
#Definición de características.
mi_ridge = Ridge(alpha=5.0,fit_intercept=True) # el 5  es
recomendacion de los que ya probaron, pero le pueden poner lo que sea
para hacer prueba
mi_ridge.fit(X_train, y_train)
X_para_ridge = X_test  #Aqui recordemos que se hace con los de prueba,
no con los de entrenamiento
y_para_ridge = mi_ridge.predict(X_para_ridge)

#Realizamos el ploteo correspondiente al modelo de Ridge.
plt.scatter(X_train, y_train)
plt.plot(X_para_ridge, y_para_ridge, "r-", linewidth=2,
```

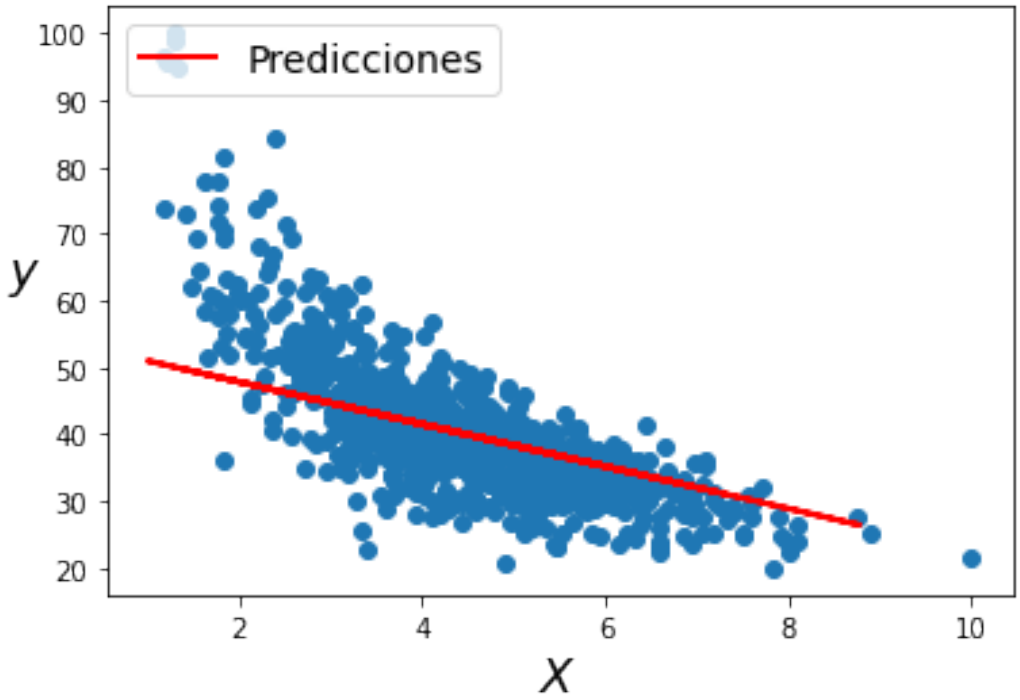
```
label="Predicciones")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);

#Definimos las métricas de Ridge
mae_ridge = metrics.mean_absolute_error(y_test,y_para_ridge)
lista_para_mae.append(mae_ridge)
r2_ridge= r2_score(y_test,y_para_ridge)
lista_para_r2.append(r2_ridge)
metrica_mae_ridge = metrics.mean_absolute_error(y_test, y_para_ridge)
r2Score = r2_score(y_test, y_para_ridge)

#Descripción del modelo de Ridge.
print('El modelo es: Y =', mi_ridge.coef_,'X +', mi_ridge.intercept_)

#Errores del modelo de Ridge.
print('Error medio Absoluto (MAE):', metrica_mae_ridge)
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_para_ridge)))

#R cuadrada del modelo de Ridge.
print('r2_score',r2Score)
```

```
El modelo es: Y = [-5.97003397] X + 66.75243237759665
Error medio Absoluto (MAE): 5.0162057389928325
Root Mean Squared Error: 7.1111119498200965
r2_score 0.6113903530239646
```



```
#Lasso
#Definición de características.
mi_lasso = Lasso(alpha=5.0,fit_intercept=True)
mi_lasso.fit(X_train, y_train)
X_para_lasso = X_test
y_para_lasso = mi_lasso.predict(X_para_ridge)

#Realizamos el ploteo correspondiente al modelo de Lasso.
plt.scatter(X_train, y_train)
plt.plot(X_para_lasso, y_para_lasso, "r-", linewidth=2,
label="Predicciones")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);

#Definimos las métricas de Lasso.
mae_lasso = metrics.mean_absolute_error(y_test,y_para_lasso)
lista_para_mae.append(mae_lasso)
r2_lasso= r2_score(y_test,y_para_lasso)
lista_para_r2.append(r2_lasso)
metrica_mae_lasso = metrics.mean_absolute_error(y_test, y_para_lasso)
r2Score = r2_score(y_test, y_para_lasso)

#Descripción del modelo de Lasso.
print('El modelo es: Y =', mi_lasso.coef_,'X +', mi_lasso.intercept_)

#Errores del modelo de Lasso.
print('Error medio Absoluto (MAE):', metrica_mae_lasso)
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_para_lasso)))

#R cuadrada del modelo de Lasso.
print('r2_score',r2Score)
```

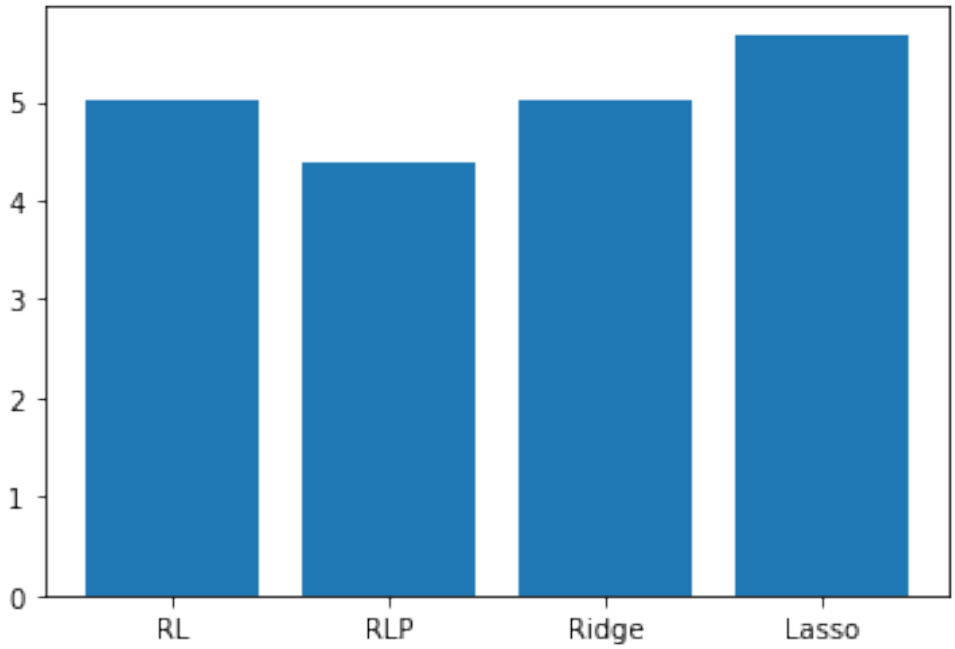
```
El modelo es: Y = [-3.15572458] X + 54.16195119377413
Error medio Absoluto (MAE): 5.681207654677401
Root Mean Squared Error: 8.409660991642687
r2_score 0.456505036516648
```



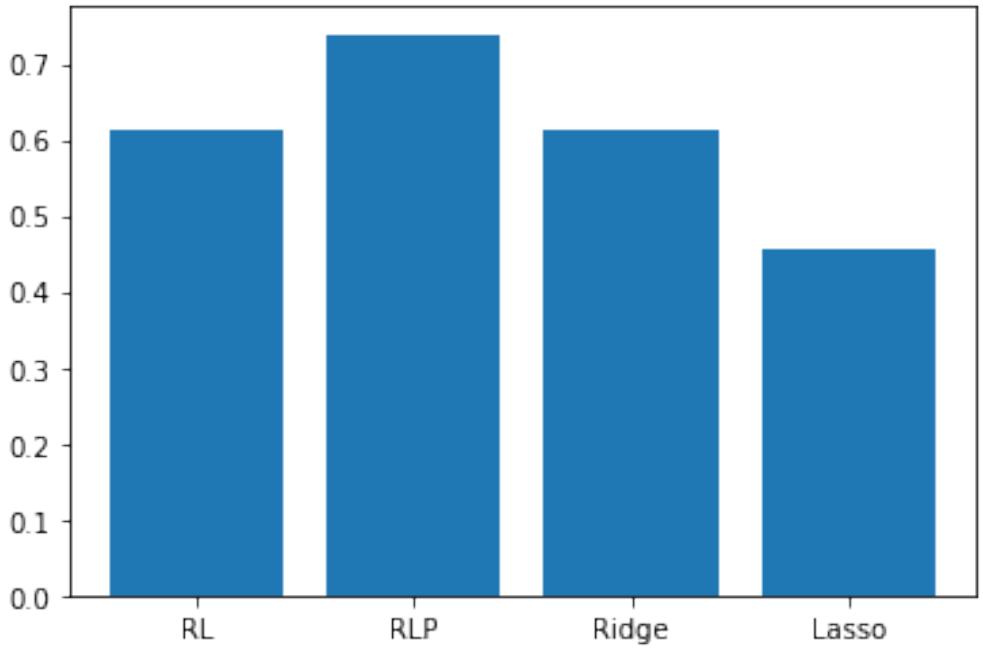

```
#Imprimimos las listas para mae y r cuadrada.
print(lista_para_mae)
print(lista_para_r2)
```

```
[5.013587781954963, 4.3833025759681075, 5.0162057389928325,
5.681207654677401]
[0.6116251549562579, 0.7385501224942537, 0.6113903530239646,
0.456505036516648]
```

```
#Realizamos las graficas de MAE de los modelos lineal, polinomial,
ridge y lasso.
nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')

plt.bar(nombres, lista_para_mae)
plt.show()
```




```
#Realizamos las graficas de r2 de los modelos lineal, polinomial,
ridge y lasso.
nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')

plt.bar(nombres, lista_para_r2)
plt.show()
```




¿Que método conviene más a la empresa? El método que más conviene para este caso es el de regresión lineal.

¿Por que? Presenta un error el cual es menor, así como una r cuadrada que es mayor, lo cual nos dice que este se ajusta de una manera adecuada y justa a los datos.

¿Que porcentajes de entrenamiento y evaluación usaste? 80% y 20%

¿Que error tienes? El error que se puede observar es de 0.0001 ¿Es bueno?

¿Cómo lo sabes? Porqué el error es menor a 0.01

#Ejercicio 2 Realiza la regresión polinomial de los siguientes datos:

```
df =
pd.read_csv('https://raw.githubusercontent.com/marypazrf/bdd/main/
kc_house_data.csv')
df.sample(10)

df.info()

df.describe()

df.drop('id', axis = 1, inplace = True)
df.drop('date', axis = 1, inplace = True)
df.drop('zipcode', axis = 1, inplace = True)
df.drop('lat', axis = 1, inplace = True)
df.drop('long', axis = 1, inplace = True)

plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, cmap='Dark2_r', linewidths = 2)
plt.show()
```

```
columns = df.columns.drop('price')

features = columns
label = ['price']

X = df[features]
y = df[label]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state = 101)

print(f'Numero total de registros en la bdd: {len(X)}')
print("*****"*10)
print(f'Numero total de registros en el training set: {len(X_train)}')
print(f'Tamaño de X_train: {X_train.shape}')
print("*****"*10)
print(f'Mumero total de registros en el test dataset: {len(X_test)}')
print(f'Tamaño del X_test: {X_test.shape}')

#tu codigo aquí
```