CIENCIA Y ANALITICA DE DATOS

Actividad Semanal -- 7 Regresiones y K means

Notebook Regresión

Profesor Titular: Maria de la Paz Rico Fernandez

Profesor Tutor: Juan Miguel Meza Méndez

Alumno: Samuel Elías Flores González

Matrícula: A01793668

Fecha: 9/Noviembre/2022

Linear Models

- 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
# 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:

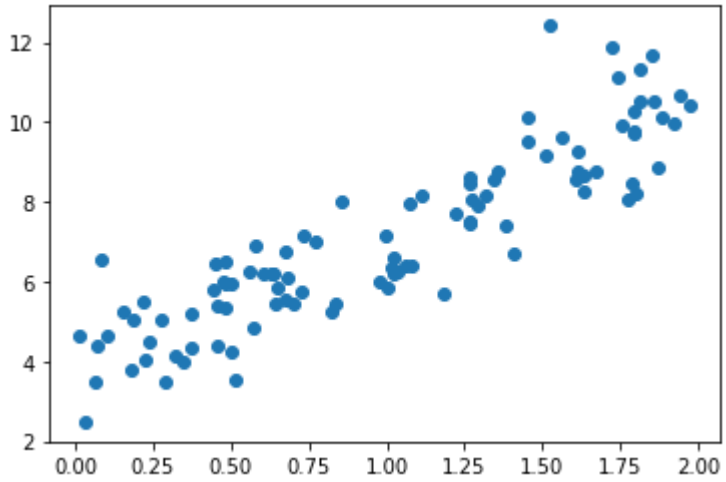
$y = 3X + 4 + noise$

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

```
[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

```
from sklearn.linear_model import LinearRegression

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

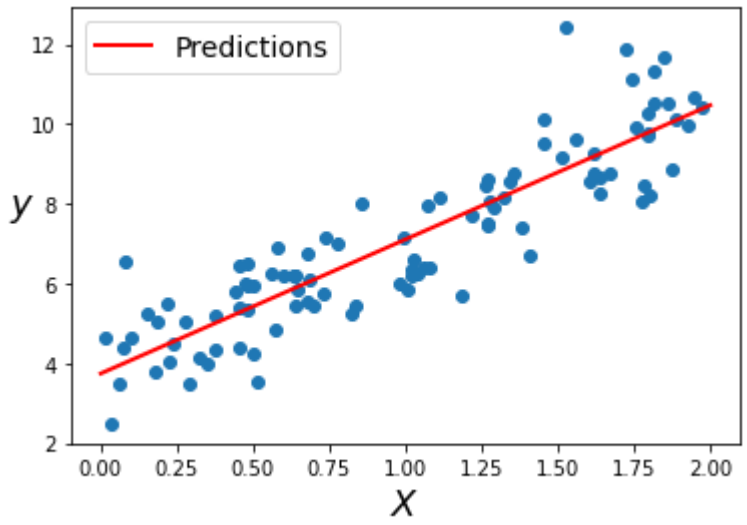
```
LinearRegression()
```

Plot the model's predictions:

98	8.061635

```
#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:

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

```
#|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

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

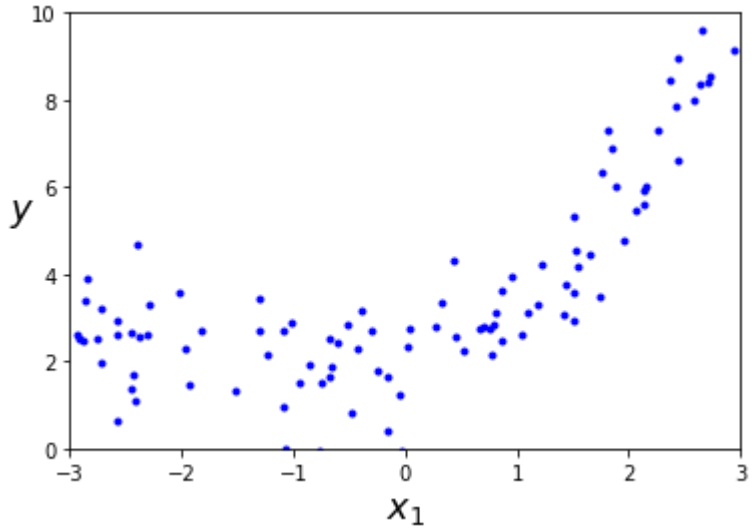
This is still a linear model, the linearity refers to the fact that the coefficients never multiply or divide each other.

To generate polynomial data we use the function:

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

```
# 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
97	3.488785

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
```

```
[ 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_
```

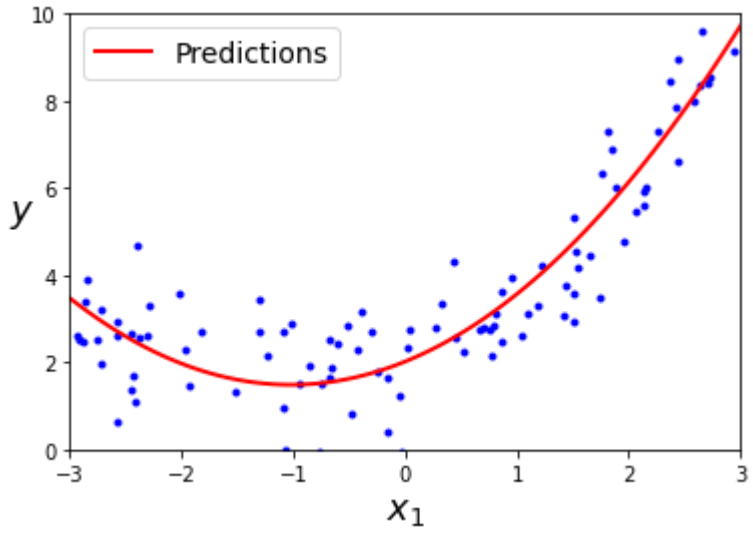
```
(array([[1.04271531, 0.50866711]]), array([2.01873554]))
```

The model estimates:

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

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]);
```

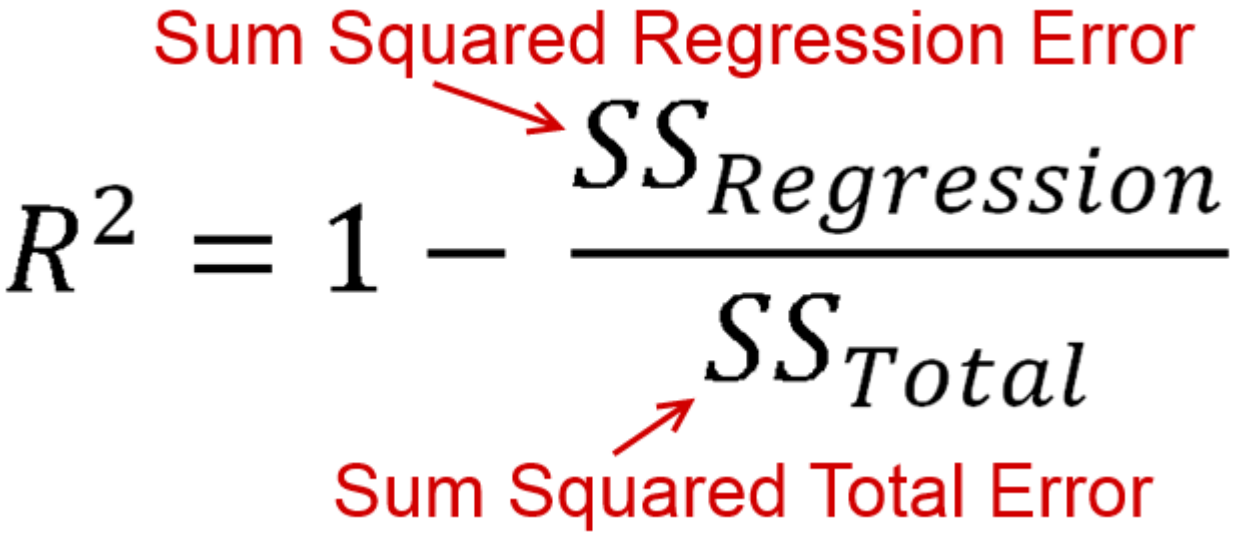


R square

R² 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, R² es una medida de ajuste para los modelos de regresión lineal.

R² no indica si un modelo de regresión se ajusta adecuadamente a sus datos. Un buen modelo puede tener un valor R² bajo. Por otro lado, un modelo sesgado puede tener un valor alto de R².

SSres + SSreg = SStot, R² = 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.

```
# Importacion de Librerias
import pandas as pd
import numpy as np

import seaborn as sbn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
```

```
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
968	7.065653	27.804027
212	3.372115	41.127212
416	4.194513	43.832711
677	5.068888	41.225741
550	4.604122	37.569764
764	5.389522	31.191501
386	4.104190	42.988730
339	3.942214	46.291435
82	2.665856	48.578425
487	4.399514	37.567914

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

```
len(X)
```

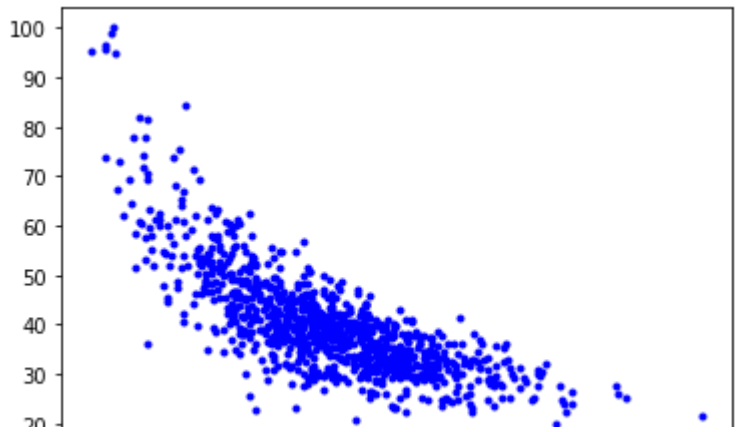
```
1000
```

```
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 0x7f3ded17da50>]
```




```
# Division del conjunto de datos en entrenamiento y prueba

Xtv, Xtest, Ytv, Ytest = train_test_split(X, y, test_size=0.20, random_state=101)


# Definiesto listas para graficar errores

R2scores = list()
MAEscores = list()
Modelos = list()
```

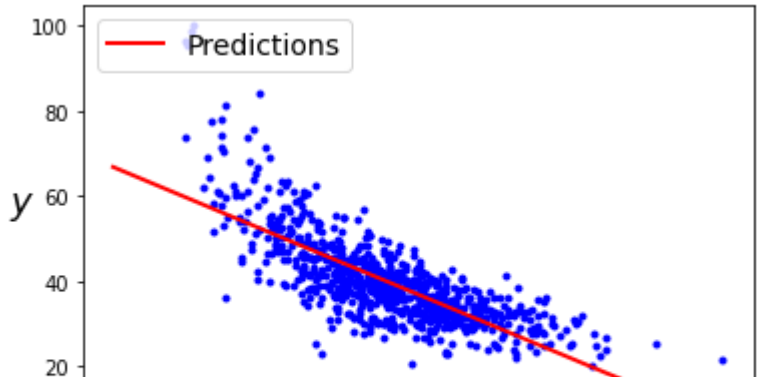
Regresion Lineal

```
# Definicion y entramiento del modelo con los datos de entrenamiento

linear_reg = LinearRegression(fit_intercept=True) #Definimos objeto
linear_reg.fit(Xtv, Ytv)
```

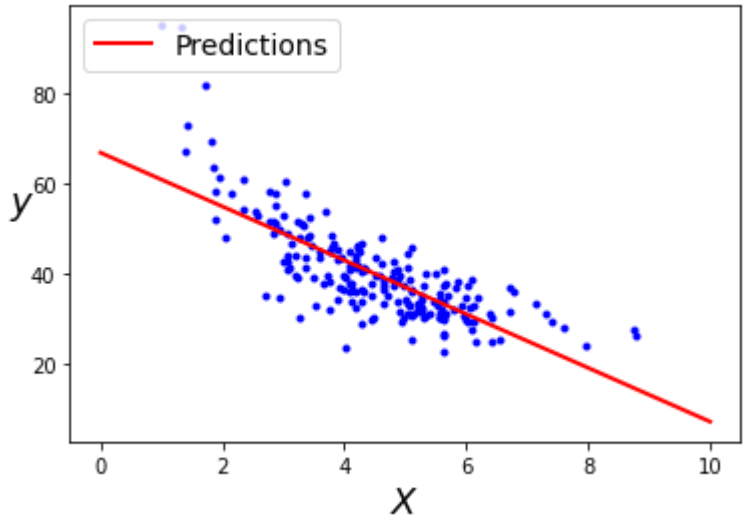
```
LinearRegression()
```

```
# Graficamos los datos de entrenamiento y predicciones

X_fit = np.linspace(0, 10, 100)
y_fit = linear_reg.predict(X_fit[:, np.newaxis]) #Best fit line

plt.plot(Xtv, Ytv, "b.")
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);
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
  "X does not have valid feature names, but"
```




```
# Graficamos los datos de prueba con su respectiva prediccion

plt.plot(Xtest, Ytest, "b.")
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);
```




```
# Obtencion de coeficientes de la ecuacion

linear_reg.coef_, linear_reg.intercept_
```

```
(array([-5.97912772]), 66.79997932683362)
```

Modelo estimado:

$$\hat{y} = -5.91X + 66.44$$

```
# Impresion de errores y R2

Yhat = linear_reg.predict(Xtest)
```

```
Modelos.append("LINEAR")
MAE = metrics.mean_absolute_error(Ytest, Yhat)
MAEscores.append(MAE)
R2 = r2_score(Ytest, Yhat)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:",R2)
```

```
Error Medio Absoluto (MAE): 5.033403667160277
Raiz del Error Cuadratico Medio (RMSE): 7.067709647692056
R2 Score: 0.595798333672179
```

Regresion Polinomial

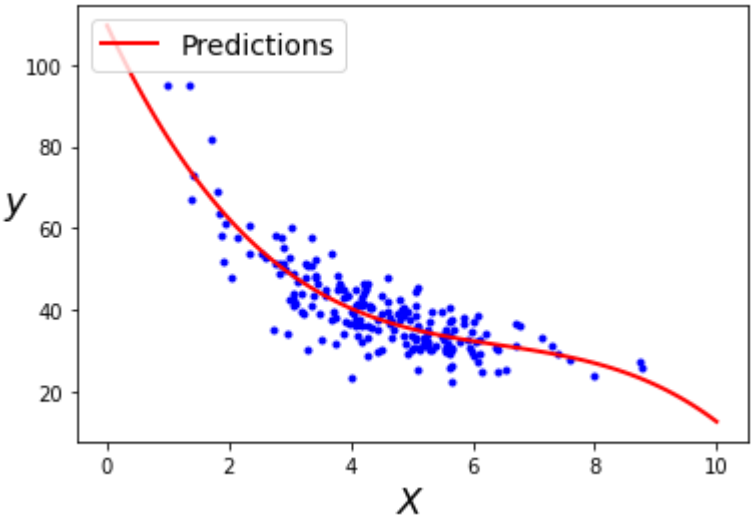
```
# Declaracion y entramiento del modelo con los datos de entrenamiento
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(Xtv)
lin_reg = LinearRegression(fit_intercept=True)
lin_reg.fit(X_poly, Ytv)
```

```
LinearRegression()
```

```
# Visualizacion de los datos de entrenamiento con su respectiva prediccion

X_new=np.linspace(0, 10, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)

plt.plot(Xtv, Ytv, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
  "X does not have valid feature names, but"
```


```
# Visualizacion del conjunto de prueba con su respectiva prediccion

plt.plot(Xtest, Ytest, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```




```
# Obtencion de coeficientes de la ecuacion

lin_reg.coef_, lin_reg.intercept_
```

```
(array([-31.83069901,   4.58905775,  -0.23734936]), 109.50904482461856)
```

Modelo estimado:

$$\hat{y} = -32.08X + 4.64X^2 - 0.24X^3 + 109.66$$

```
# Impresion de errores y R2
X_poly2 = poly_features.fit_transform(Xtest)
Yhat2 = lin_reg.predict(X_poly2)

Modelos.append("POLYNOMIAL")
MAE = metrics.mean_absolute_error(Ytest, Yhat2)
MAEscores.append(MAE)
R2 = r2_score(Ytest, Yhat2)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
```

```
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:", R2)
```

```
Error Medio Absoluto (MAE): 4.339059416327446
Raiz del Error Cuadratico Medio (RMSE): 5.637445640824421
R2 Score: 0.7428388053898054
```

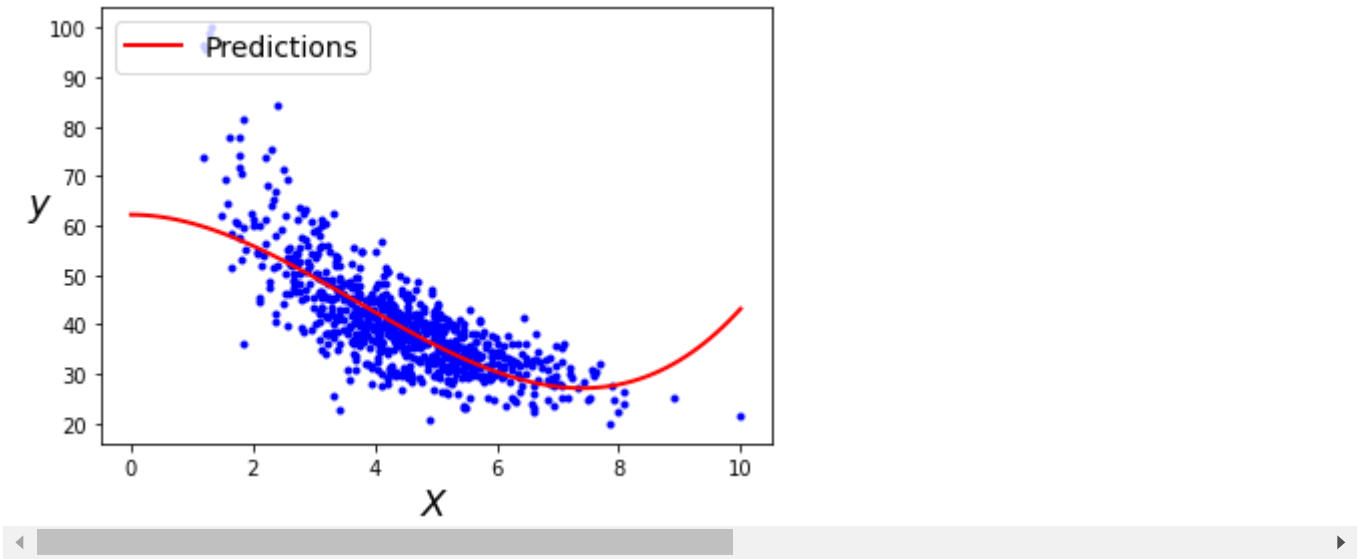
Regresion Lasso

```
# Declaracion y entramiento del modelo con los datos de entrenamiento
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(Xtv)
lin_reg = Lasso()
lin_reg.fit(X_poly, Ytv)
```

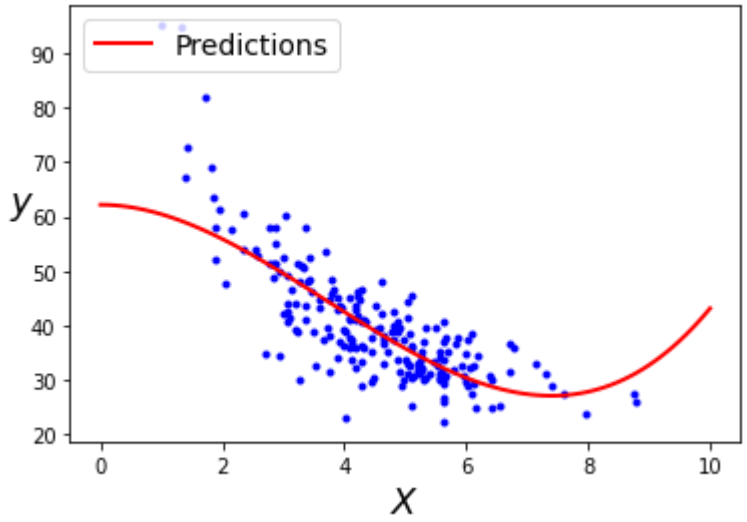
```
Lasso()
```

```
# Visualizacion de los datos de entrenamiento con su respectiva prediccion

X_new=np.linspace(0, 10, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)

plt.plot(Xtv, Ytv, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
  "X does not have valid feature names, but"
```



```
# Visualizacion del conjunto de prueba con su respectiva prediccion

plt.plot(Xtest, Ytest, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```




```
# Obtencion de coeficientes de la ecuacion

lin_reg.coef_, lin_reg.intercept_
```

```
(array([-0.        , -1.91976298,  0.17297538]), 62.19704917714074)
```

Modelo estimado:

$$\hat{y} = -1.90X^2 + 0.17X^3 + 61.97$$

```
# Impresion de errores y R2
X_poly2 = poly_features.fit_transform(Xtest)
Yhat2 = lin_reg.predict(X_poly2)

Modelos.append("LASSO")
MAE = metrics.mean_absolute_error(Ytest, Yhat2)
MAEscores.append(MAE)
R2 = r2_score(Ytest, Yhat2)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:", R2)
```

```
Error Medio Absoluto (MAE): 4.722951590074714
Raiz del Error Cuadratico Medio (RMSE): 6.805231614821702
R2 Score: 0.6252630463323101
```

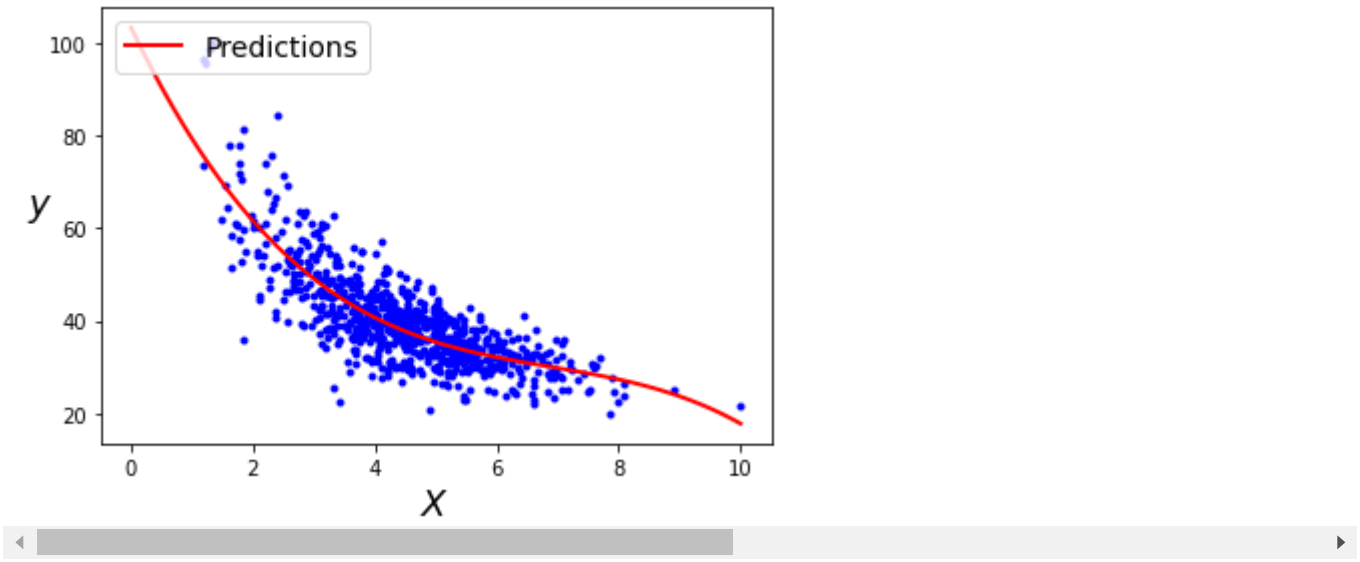
Regresion Ridge

```
# Declaracion y entramiento del modelo con los datos de entrenamiento
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(Xtv)
lin_reg = Ridge()
lin_reg.fit(X_poly, Ytv)
```

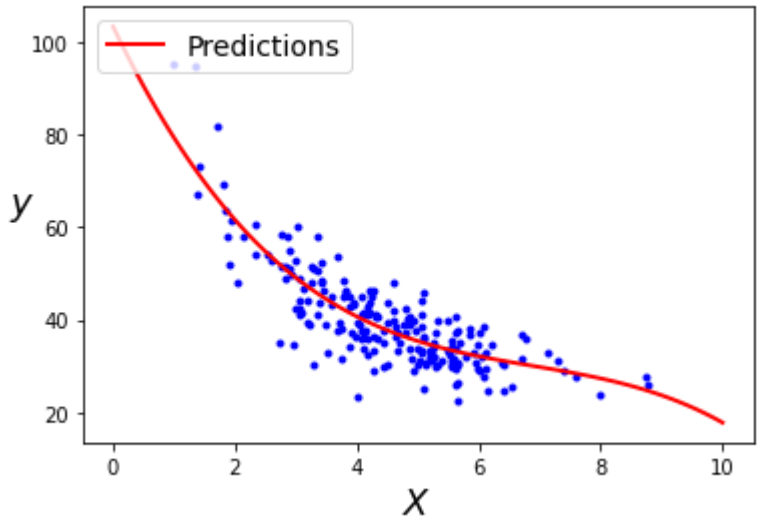
```
Ridge()
```

```
# Visualizacion de los datos de entrenamiento con su respectiva prediccion

X_new=np.linspace(0, 10, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)

plt.plot(Xtv, Ytv, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
  "X does not have valid feature names, but"
```




```
# Visualizacion del conjunto de prueba con su respectiva prediccion

plt.plot(Xtest, Ytest, "b.")
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```




```
# Obtencion de coeficientes de la ecuacion

lin_reg.coef_, lin_reg.intercept_
```

```
(array([-27.45935455,   3.66676159,  -0.17748537]), 103.23151933795558)
```

Modelo estimado:

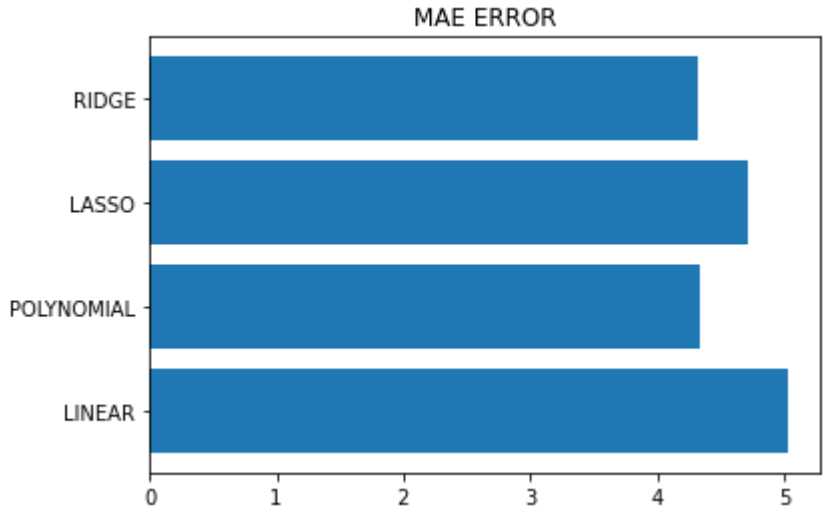
$$\hat{y} = -27.92X + 3.77X^2 - 0.18X^3 + 103.68$$

```
# Impresion de errores y R2
X_poly2 = poly_features.fit_transform(Xtest)
Yhat2 = lin_reg.predict(X_poly2)

Modelos.append("RIDGE")
MAE = metrics.mean_absolute_error(Ytest, Yhat2)
MAEscores.append(MAE)
R2 = r2_score(Ytest, Yhat2)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:", R2)
```

```
Error Medio Absoluto (MAE): 4.326794629433284
Raiz del Error Cuadratico Medio (RMSE): 5.695015174630914
R2 Score: 0.7375597327395196
```

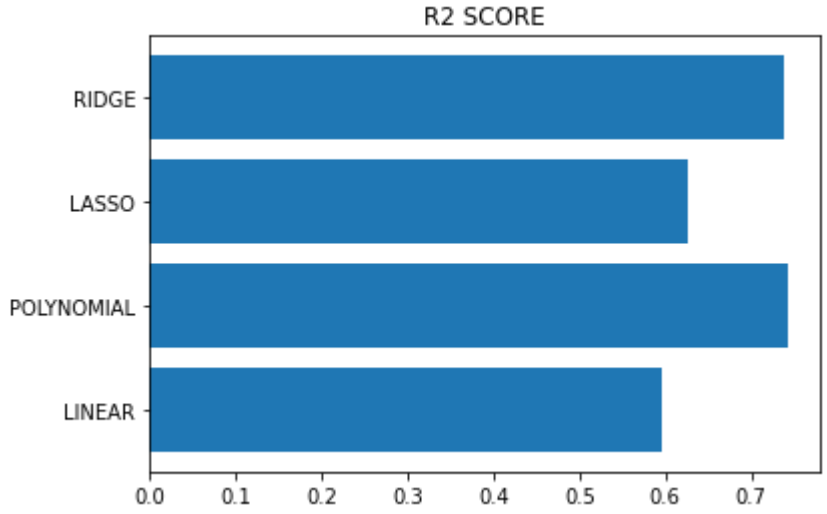
Grafica MAE Scores

```
plt.barh(Modelos, MAEscores)
plt.title("MAE ERROR")
plt.show()
```



Grafica RMSE Scores

```
plt.barh(Modelos, R2scores)
plt.title("R2 SCORE")
plt.show()
```



Resultados

Realizando pruebas con diferentes grados polinomiales, podemos observar que el el modelo se comporto de una mejor manera cuando se hizo uso de funciones polinomiales de tercer grado.

¿Que metodo conviene mas a la empresa?, ¿Por que?

El método que mejor se ajusta al problema planteado es Ridge. Nos podemos dar cuenta de ello al observar sus graficas, donde la linea de predicción se adapta o ajusta mejor al conjunto de datos.

Además de obtener mejores valores de error.

¿Que porcentajes de entrenamiento y evaluacion usaste?

Conjunto de datos de entrenamiento: 80%

Conjunto de datos de prueba: 20%

¿Que error tienes?, ¿Es bueno?, ¿Como lo sabes?

El error que presentan los modelos se encuentran entre los valores de 4-5, sin embargo los dos modelos que mejor error tienen son ridge y polinomial con 4.2 y 4.3 respectivamente. Estos valores pueden considerarse buenos debido a que es un valor de error relativamente bajo.

Para el caso de R2 Ridge y polinomial presentan 0.72 y 0.73 respectivamente, lo que indican que el modelo es bueno. a medida que este valor se acerca a 1, el modelo se comportará de una mejor manera, es decir no esta ni sobre entrenado ni subentrendo

Ejercicio 2

Realiza la regresión polinomial de los siguientes datos:

```
# Importacion de Librerias

# Tratamiento de los datos
import pandas as pd
import numpy as np
# Graficos
import seaborn as sbn
import matplotlib.pyplot as plt
# Preprocesado y Modelado
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures


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

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_l
5954	7852020250	20140602T000000	725995.0	4	2.50	3190	78
8610	6392002020	20150324T000000	559000.0	3	1.75	1700	65
7650	626049058	20150504T000000	275000.0	5	2.50	2570	172
5683	2202500255	20150305T000000	335000.0	3	2.00	1210	99
20773	7304301231	20140617T000000	345000.0	3	2.50	1680	22
6959	723000114	20140505T000000	1395000.0	5	3.50	4010	85
10784	4104900340	20150204T000000	710000.0	4	2.50	3220	186
21529	2487200490	20140623T000000	670000.0	3	2.50	3310	53
12319	2386000070	20141029T000000	795127.0	4	3.25	4360	911
19948	293070090	20140711T000000	859990.0	4	2.75	3520	55

10 rows × 21 columns

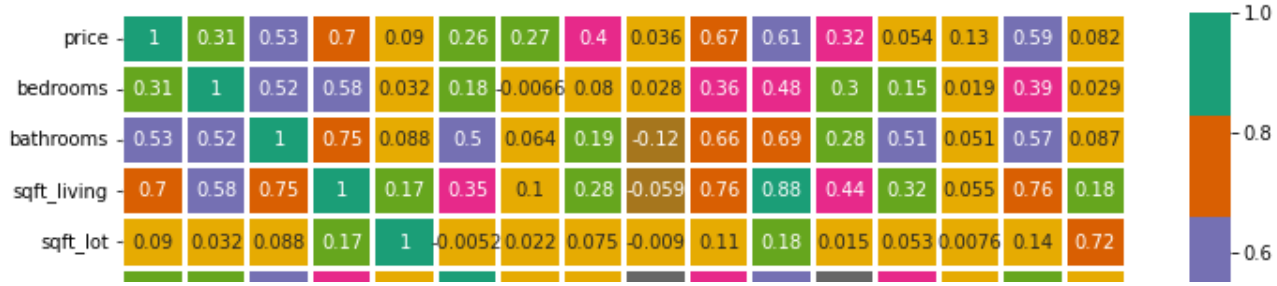
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
df.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06

```
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))
sbn.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]
```


```
# Division del conjunto de datos en entrenamiento y prueba

Xtv, Xtest, Ytv, Ytest = train_test_split(X, y, test_size=0.20, random_state=101)
```

```
# Definiesto listas para graficar errores

R2scores = list()
MAEscores = list()
Modelos = list()
```

Regresion Lineal

```
# Declaracion y entramiento del modelo con los datos de entrenamiento

linear_reg = LinearRegression(fit_intercept=True)
linear_reg.fit(Xtv, Ytv)
```

```
LinearRegression()
```

Modelo estimado:

```
# Obtencion de coeficientes de la ecuacion

linear_reg.coef_, linear_reg.intercept_
```

```
(array([[-3.71022526e+04,  4.01418845e+04,  1.09169906e+02,
          2.64378621e-02,  2.95338523e+04,  5.52048615e+05,
          4.22004055e+04,  2.21281704e+04,  1.21117318e+05,
          5.02818538e+01,  5.88880527e+01, -3.52812693e+03,
```

```
                1.04703799e+01,  2.60075557e+01, -5.66471373e-01]]),
      array([6093360.93532648]))
```

```
# Impresion de errores y R2

Yhat = linear_reg.predict(Xtest)

Modelos.append("LINEAR")
MAE = metrics.mean_absolute_error(Ytest, Yhat)
MAEscores.append(MAE)
R2 = r2_score(Ytest, Yhat)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:",R2)
```

```
Error Medio Absoluto (MAE): 136332.19435816712
Raiz del Error Cuadratico Medio (RMSE): 213832.57523968935
R2 Score: 0.6648546558345027
```

Regresion Polinomial

```
# Declaracion y entramiento del modelo con los datos de entrenamiento
lin_reg = LinearRegression(fit_intercept=True)
lin_reg.fit(Xtv, Ytv)
```

```
LinearRegression()
```

Modelo estimado:

```
# Obtencion de coeficientes de la ecuacion

lin_reg.coef_, lin_reg.intercept_
```

```
(array([[-3.71022526e+04,  4.01418845e+04,  1.09169906e+02,
          2.64378621e-02,  2.95338523e+04,  5.52048615e+05,
          4.22004055e+04,  2.21281704e+04,  1.21117318e+05,
          5.02818538e+01,  5.88880527e+01, -3.52812693e+03,
          1.04703799e+01,  2.60075557e+01, -5.66471373e-01]]),
 array([6093360.93532648]))
```

```
# Impresion de errores y R2
Yhat2 = lin_reg.predict(Xtest)

Modelos.append("POLYNOMIAL")
MAE = metrics.mean_absolute_error(Ytest, Yhat2)
MAEscores.append(MAE)
```

```
R2 = r2_score(Ytest, Yhat2)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:", R2)
```

```
Error Medio Absoluto (MAE): 136332.19435816712
Raiz del Error Cuadratico Medio (RMSE): 213832.57523968935
R2 Score: 0.6648546558345027
```

Regresion Lasso

```
# Declaracion y entramiento del modelo con los datos de entrenamiento
lin_reg = Lasso()
lin_reg.fit(Xtv, Ytv)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:648:
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
Lasso()
```

Modelo estimado:

```
# Obtencion de coeficientes de la ecuacion

lin_reg.coef_, lin_reg.intercept_
```

```
(array([-3.71009851e+04,  4.01376842e+04,  2.97310842e+02,  2.64300933e-02,
         2.95294081e+04,  5.51902073e+05,  4.22058660e+04,  2.21254807e+04,
         1.21116062e+05, -1.37854556e+02, -1.29251177e+02, -3.52807746e+03,
         1.04730018e+01,  2.60070042e+01, -5.66472918e-01]),
 array([6093285.72783688]))
```

```
# Impresion de errores y R2
Yhat2 = lin_reg.predict(Xtest)

Modelos.append("LASSO")
MAE = metrics.mean_absolute_error(Ytest, Yhat2)
MAEscores.append(MAE)
R2 = r2_score(Ytest, Yhat2)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:", R2)
```

```
Error Medio Absoluto (MAE): 136332.42567085347
Raiz del Error Cuadratico Medio (RMSE): 213833.2347796031
```

```
R2 Score: 0.6648525884032703
```

Regresion Ridge

```
# Declaracion y entramiento del modelo con los datos de entrenamiento
lin_reg = Ridge()
lin_reg.fit(Xtv, Ytv)
```

```
Ridge()
```

Modelo estimado:

```
# Obtencion de coeficientes de la ecuacion

lin_reg.coef_, lin_reg.intercept_
```

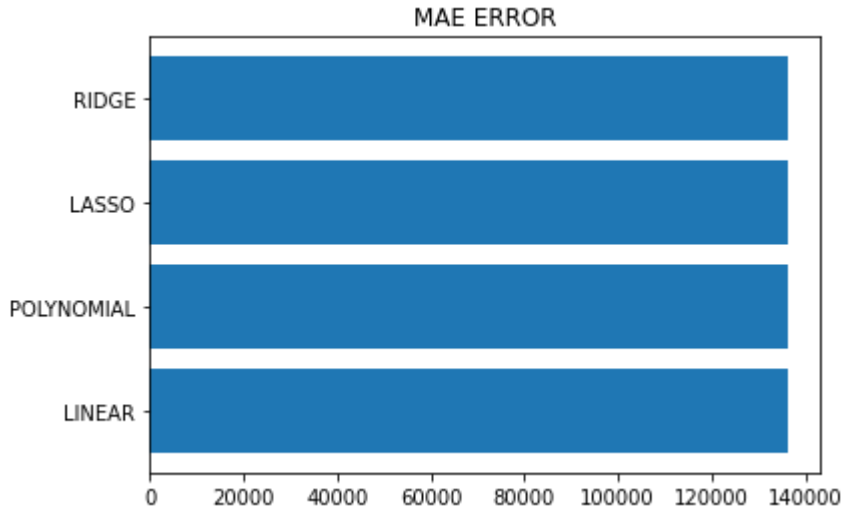
```
(array([[-3.71220406e+04,  4.01268952e+04,  1.09190469e+02,
          2.61948704e-02,  2.95324372e+04,  5.47178618e+05,
          4.24249301e+04,  2.21356344e+04,  1.21095755e+05,
          5.03177099e+01,  5.88734822e+01, -3.52773641e+03,
          1.05349857e+01,  2.59847053e+01, -5.66259176e-01]]),
 array([6092753.91682682]))
```

```
# Impresion de errores y R2
Yhat2 = lin_reg.predict(Xtest)

Modelos.append("RIDGE")
MAE = metrics.mean_absolute_error(Ytest, Yhat2)
MAEscores.append(MAE)
R2 = r2_score(Ytest, Yhat2)
R2scores.append(R2)

print("Error Medio Absoluto (MAE):", MAE)
print("Raiz del Error Cuadratico Medio (RMSE):", np.sqrt(metrics.mean_squared_error(Ytest, Yh
print("R2 Score:", R2)
```

```
Error Medio Absoluto (MAE): 136338.19942052223
Raiz del Error Cuadratico Medio (RMSE): 213852.21135950755
R2 Score: 0.6647931006080388
```

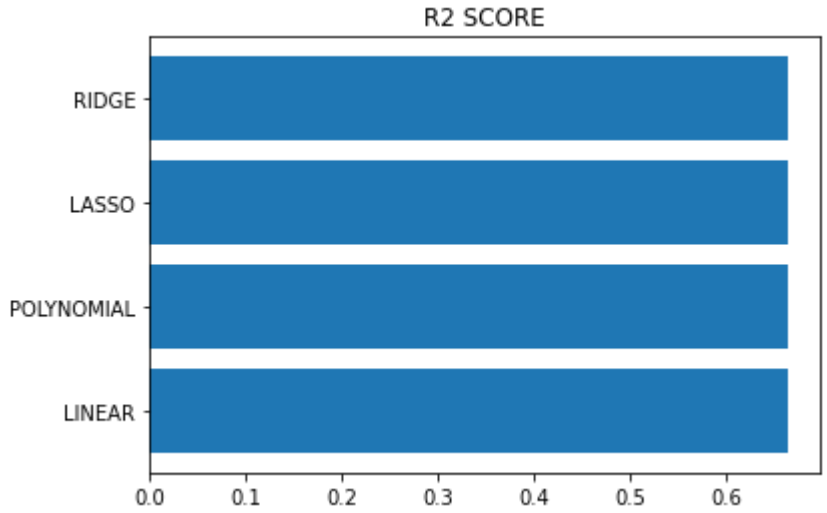
Grafica MAE Scores

```
plt.barh(Modelos, MAEscores)
plt.title("MAE ERROR")
plt.show()
```



Grafica RMSE Scores

```
plt.barh(Modelos, R2scores)
plt.title("R2 SCORE")
plt.show()
```



Resultados

Los datos sorpresivamente obtuvieron valores de error muy altos por lo cual son valores no esperados. Esto podría deberse a que no fueron transformados los datos ni normalizados. las diferentes variables de entrada al presentar diferentes rangos en sus magnitudes, pueden llegar a ocacionar ruido en los modelos.

¿Que metodo conviene mas a la empresa?, ¿Por que?

Ninguno de los modelos es apto para considerarse conveniente para la empresa, debido al error tan alto que presentan asi como su R2 bajo.

¿Que porcentajes de entrenamiento y evaluacion usaste?

Conjunto de datos de entrenamiento: 80%

Conjunto de datos de prueba: 20%

¿Que error tienes?, ¿Es bueno?, ¿Como lo sabes?

El error de todos los modelos mostrados para este ejercicio es malo, ya que son valores muy elevados. posiblemente requieran de transformación alguna