Presentado por:

Giovanni Acuña Morales ***A01794007***

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)
```

```
#Nuestras librerias
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error,  make
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import power_transform
rt RepeatedKFold, RepeatedStratifiedKFold
rt cross_val_score
 StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV, train_test_split,
```

Se guardó correctamente

```
#La primer division del ejercicio se hace a partir de la linea 23, vamonos para haya
```

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)
```

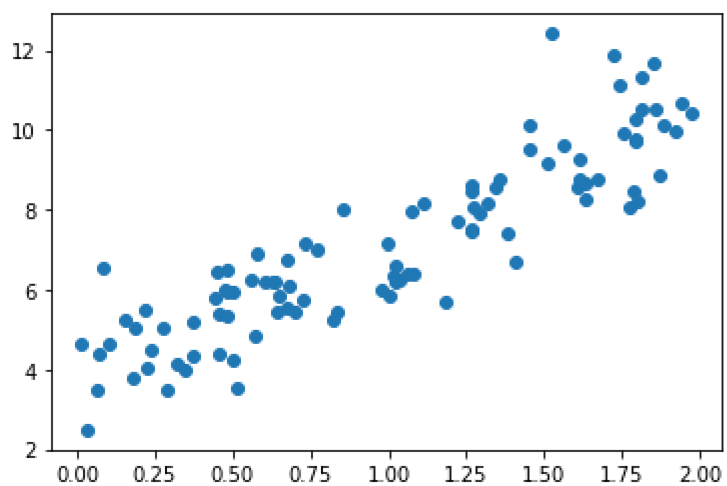
```
            [0.49517691],
            [0.03438852],
            [0.9093204 ],
            [0.25877998],
            [0.66252228],
            [0.31171108],
            [0.52006802],
            [0.54671028],
            [0.18485446],
            [0.96958463],
            [0.77513282],
            [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],
```

Se guardó correctamente

```
            [0.81546143],
            [0.70685734],
            [0.72900717]
```

```
       [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)
```

Se guardó correctamente

	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

```
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);
```

Se guardó correctamente

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
s and then train a linear model on the extended set of

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

to

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

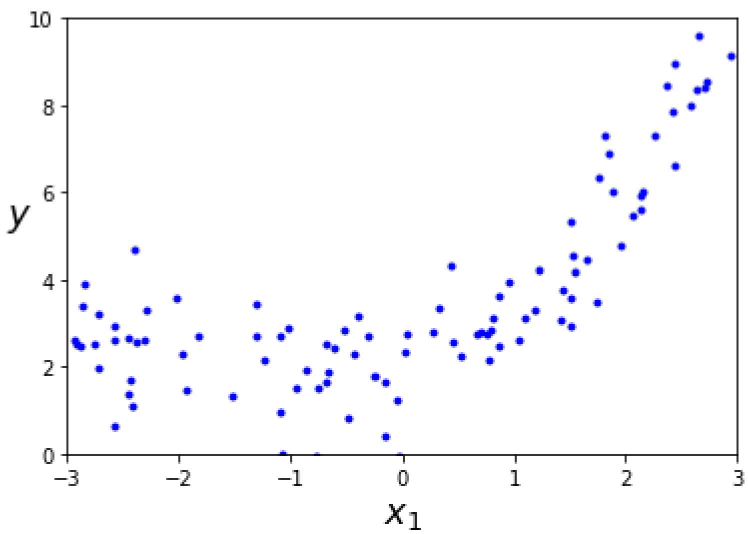
This is still a linear model, the linearity refers to the fact that the coefficients never multiply or divide each other.

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)
```

Se guardó correctamente

	0
0	8.529240
1	3.768929
2	3.354423
3	2.747935
4	0.808458
...	...
95	5.346771
96	6.338229

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

99	-0.072150

```
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],
                          74562893e+00],
                          26107913e+00],
                          38931206e+00],
       [ 2.94303085e+00,  8.66143060e+00],
       [-5.24293939e-01,  2.74884134e-01],
```

Se guardó correctamente

```
       [-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:

Se guardó correctamente

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

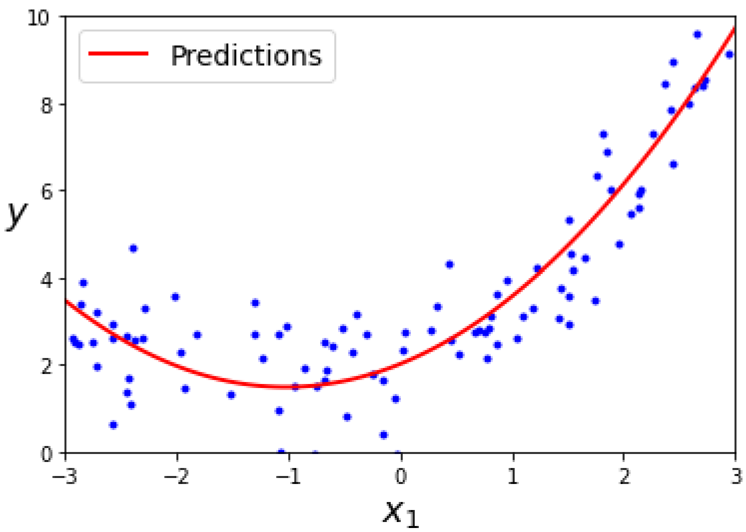
```
(array([[1.04271531, 0.50866711]]), array([2.01873554]))
```

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

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 ado, un modelo sesgado puede tener un valor alto de R^2.

Se guardó correctamente

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

Sum Squared Regression Error

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}}$$

Sum Squared Total Error

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

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

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.

Ejercicio 1. Costo en la industria de manufactura. Ahora realizaremos los ejercicios de regresión utilizando una parte para entrenar y otra para evaluar.

- Divide los datos del costo de manufactura. Utiliza la función train_test_split (viene el ejemplo al final del notebook).
- Regresión Lineal.
- Realiza la regresión lineal: modelo generado (ecuación), su visualización, sus errores y r cuadrada.

- Realiza la regresión polinomial completa, tu modelo generado (ecuación), su visualización, sus errores y r cuadrada.

- Realiza la regresión con Ridge y Lasso. Incluye la ecuación de tu modelo, visualización , errores y r cuadrada.
- Finalmente grafica :
- *MAE** (de los cuatro métodos)
- *R2** (de los cuatro métodos)
- Explica tus resultados, que método conviene más a la empresa, ¿por que?, ¿que porcentajes de entrenamiento y evaluación usaste?, ¿que error tienes?, ¿es bueno?, ¿cómo lo sabes?

```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/marypazrf/bdd/main/EconomiesOfSca
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
```

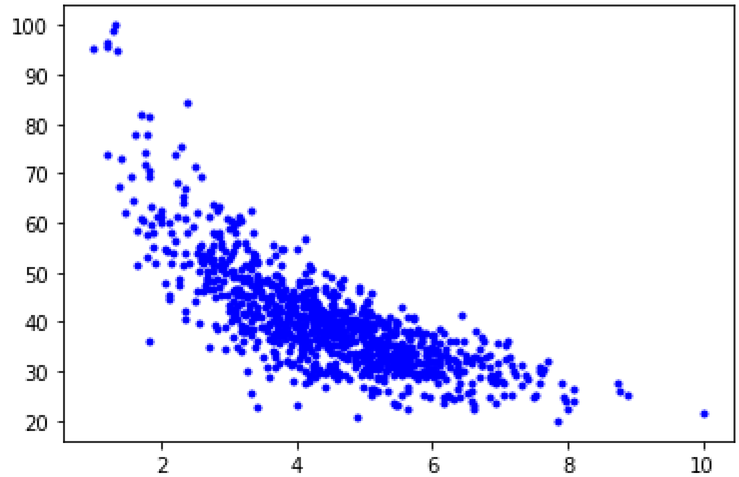
Se guardó correctamente

```
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 0x7fada5b3df10>]
```



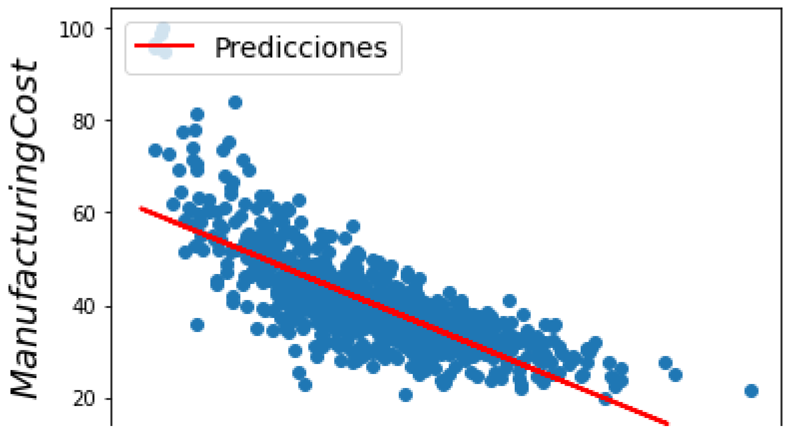
Divide los datos del costo de manufactura. Utiliza la función train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_sta
lista_para_mae =[]
lista_para_r2 =[]
```

- Regresión Lineal.
- Realiza la regresión lineal: modelo generado (ecuación), su visualización, sus errores y r cuadrada.

```
linear_reg = LinearRegression(fit_intercept=True)
linear_reg.fit(X_train, y_train)
X_para_regresion = X_test
y_para_regresion = linear_reg.predict(X_para_regresion)
plt.scatter(X_train, y_train)
plt.plot(X_para_regresion, y_para_regresion, "r-", linewidth=2, label="Predicciones")
plt.xlabel("$Number of Units$", fontsize=18)
plt.ylabel("$Manufacturing Cost$", rotation=90, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```

Se guardó correctamente




```
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)

print(f'El modelo contiene:\n\tY = {linear_reg.coef_[0]}\n\tX = {linear_reg.intercept
print(f'El Error Medio Absoluto (MAE) es : {metrics.mean_absolute_error(y_test,y_para
print(f'El Error Medio Cuadrado (RMSE) es : {np.sqrt(metrics.mean_squared_error(y_tes
print(f'La R cuadrada es r2_score: {r2_score(y_test,y_para_regresion)}')
```

```
El modelo contiene:
        Y = -5.988826991706113
        X = 66.83650741226988

El Error Medio Absoluto (MAE) es : 5.013587781954963

El Error Medio Cuadrado (RMSE) es : 7.108963321847682

La R cuadrada es r2_score: 0.6116251549562579
```

```
caracteristicas_para_poly = PolynomialFeatures(degree=2, include_bias=False)
X_polinomial = caracteristicas_para_poly.fit_transform(X_train)
print(f'Input: {caracteristicas_para_poly.n_input_features_}')
print(f'Ouput: {caracteristicas_para_poly.n_output_features_}')
print(f'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
```

Se guardó correctamente

```
t-packages/sklearn/utils/deprecation.py:103: FutureW
  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)
```

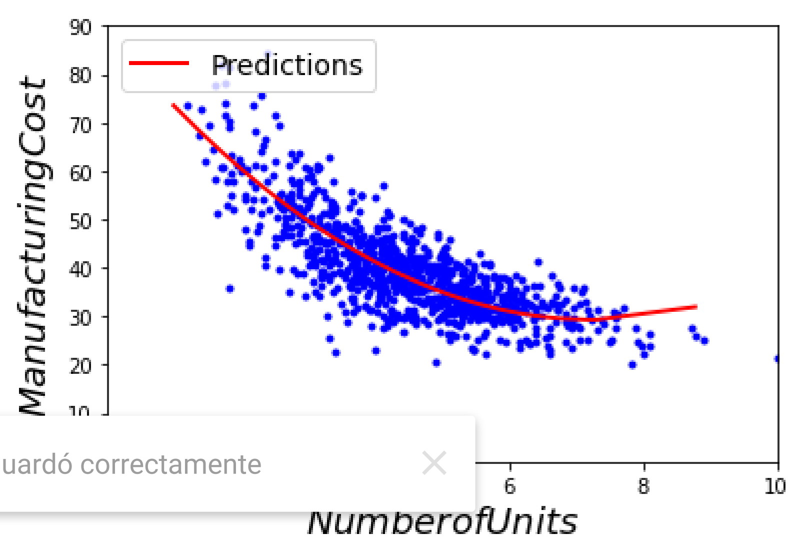
```
y_con_regresion_poli = regresion_lineal_poli.predict(X_polinomial_test)
y_con_regresion_poli.shape
```

```
(100,)
```

Finalmente grafica :

- MAE* (de los cuatro métodos)
- R2* (de los cuatro métodos)

```
order = np.argsort(X_test.values.ravel())
sortedXPoly = X_test.values.ravel()[order]
sortedYPoly = y_test.values.ravel()[order]
sorted_predicPoly = y_con_regresion_poli[order]

plt.plot(X, y, "b.")
plt.plot(sortedXPoly, sorted_predicPoly, "r-", linewidth=2, label="Predictions")
plt.xlabel("$Number of Units$", fontsize=18)
plt.ylabel("$Manufacturing Cost$", rotation=90, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 10, 0, 90]);
```



Se guardó correctamente

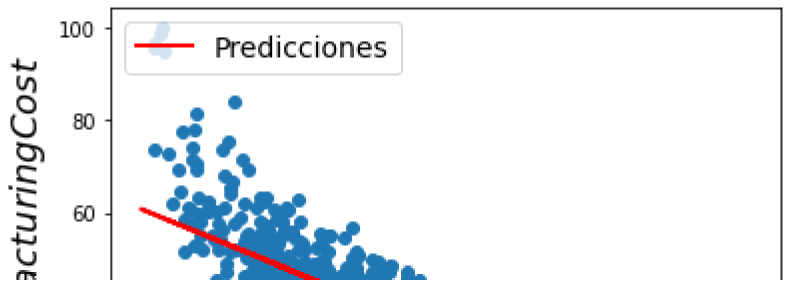
Generación de errores

```
from sklearn import metrics
from sklearn.metrics import r2_score
import random

print(f'El modelo tomado es: Y = {regresion_lineal_poli.coef_[1]} X^2 + {regresion_li

mae_regresion_lineal_multiple = metrics.mean_absolute_error(y_test,y_con_regresion_po
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)
print(f'Error medio Absoluto (MAE): {metrica_mae}\n')
print(f'Root Mean Squared Error: {np.sqrt(metrics.mean_squared_error(y_test, y_con_re
print(f'R2_score : {r2Score}')
```

```
El modelo tomado es: Y = 1.1313609537119216 X^2 + -16.406381017212386 X + 88.801
Error medio Absoluto (MAE): 4.3833025759681075

Root Mean Squared Error: 5.832771301068425

R2_score : 0.7385501224942536
```

Realiza la regresión con **Ridge** y **Lasso**. Incluye la ecuación de tu modelo, visualización , errores y r cuadrada.

```
mi_ridge = Ridge(alpha=5.0,fit_intercept=True)
mi_ridge.fit(X_train, y_train)
X_para_ridge = X_test
y_para_ridge = mi_ridge.predict(X_para_ridge)
plt.scatter(X_train, y_train)
plt.plot(X_para_ridge, y_para_ridge, "r-", linewidth=2, label="Predicciones")
plt.xlabel("$Number of Units$", fontsize=18)
plt.ylabel("$Manufacturing Cost$", rotation=90, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```



Metricas del **Ridge**

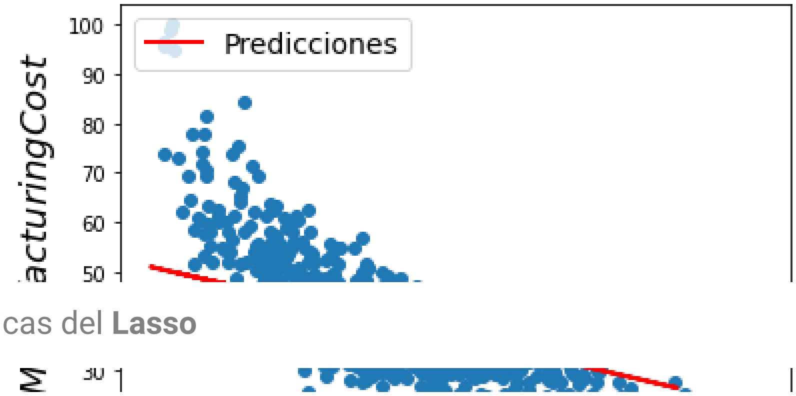
```
mae_ridge = metrics.mean_absolute_error(y_test,y_para_ridge)
lista_para_mae.append(mae_ridge)
r2_ridge= r2_score(y_test,y_para_ridge)
lista_para_r2.append(r2_ridge)

metrica_mae_ridge = metrics.mean_absolute_error(y_test, y_para_ridge)
r2Score = r2_score(y_test, y_para_ridge)
print(f'Error medio Absoluto (MAE): {metrica_mae_ridge}\n')
print(f'Root Mean Squared Error: {np.sqrt(metrics.mean_squared_error(y_test, y_para_r
print(f'R2_score: {r2Score}\n')
print(f'El modelo aplicado es: Y = {mi_ridge.coef_[0]} X + {mi_ridge.intercept_}')
```

```
Error medio Absoluto (MAE): 5.016205738992834

Root Mean Squared Error: 7.111111949820097

R2_score: 0.6113903530239646

El modelo aplicado es: Y = -5.97003397211605 X + 66.75243237759665
```

Regrsión de **Lasso**

```
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)
plt.scatter(X_train, y_train)
plt.plot(X_para_lasso, y_para_lasso, "r-", linewidth=2, label="Predicciones")
plt.xlabel("$Number of Units$", fontsize=18)
plt.ylabel("$Manufacturing Cost$", rotation=90, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```

Se guardó correctamente



Metricas del **Lasso**

```
mae_lasso = metrics.mean_absolute_error(y_test,y_para_lasso)
print(f'mae_lasso: {mae_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)
print(f'Error medio Absoluto (MAE): {metrica_mae_lasso}\n')
print(f'Root Mean Squared Error: {np.sqrt(metrics.mean_squared_error(y_test, y_para_l
print(f'R2_score : {r2Score}\n')
print(f'El modelo aplicado es: Y = {mi_lasso.coef_} X + {mi_lasso.intercept_}\n')
```

```
mae_lasso: 5.681207654677401
Error medio Absoluto (MAE): 5.681207654677401

Root Mean Squared Error: 8.409660991642687

R2_score : 0.456505036516648

El modelo aplicado es: Y = [-3.15572458] X + 54.16195119377412
```

Grafico **MAE**

```
nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')
plt.bar(nombres, lista_para_mae[:4])
plt.show()
```

Se guardó correctamente ×

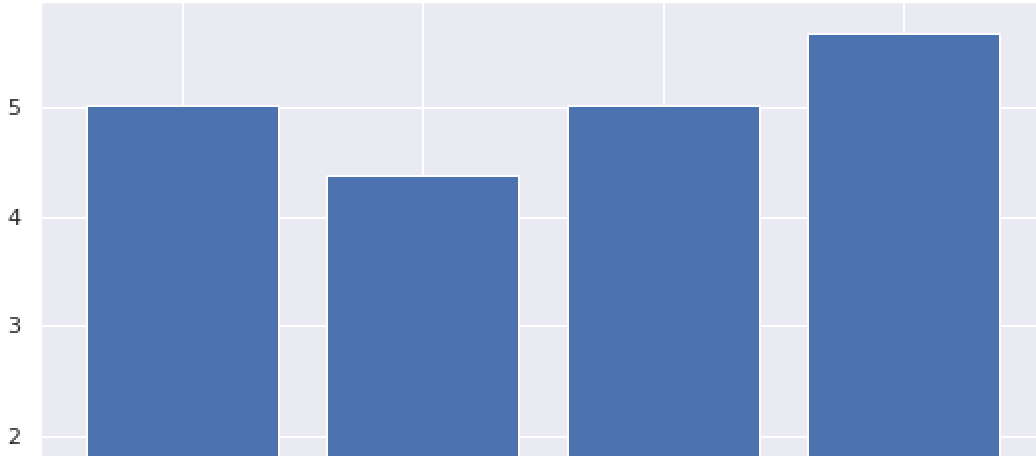
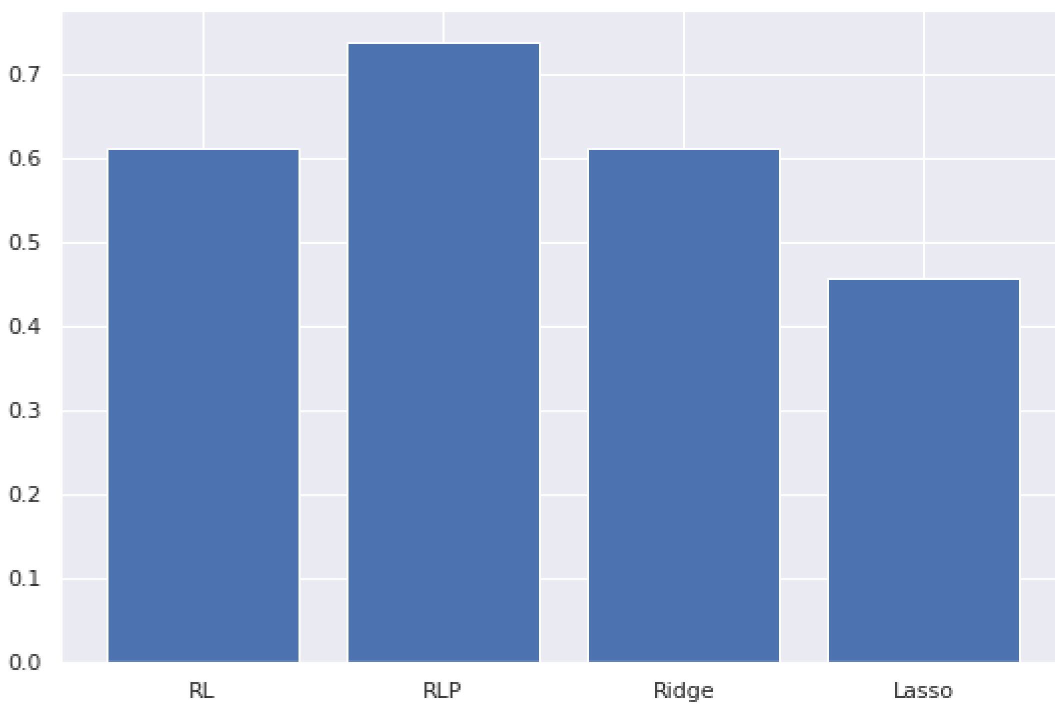


Grafico de r cuadrada

```
nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')
plt.bar(nombres, lista_para_r2)
plt.show()
```

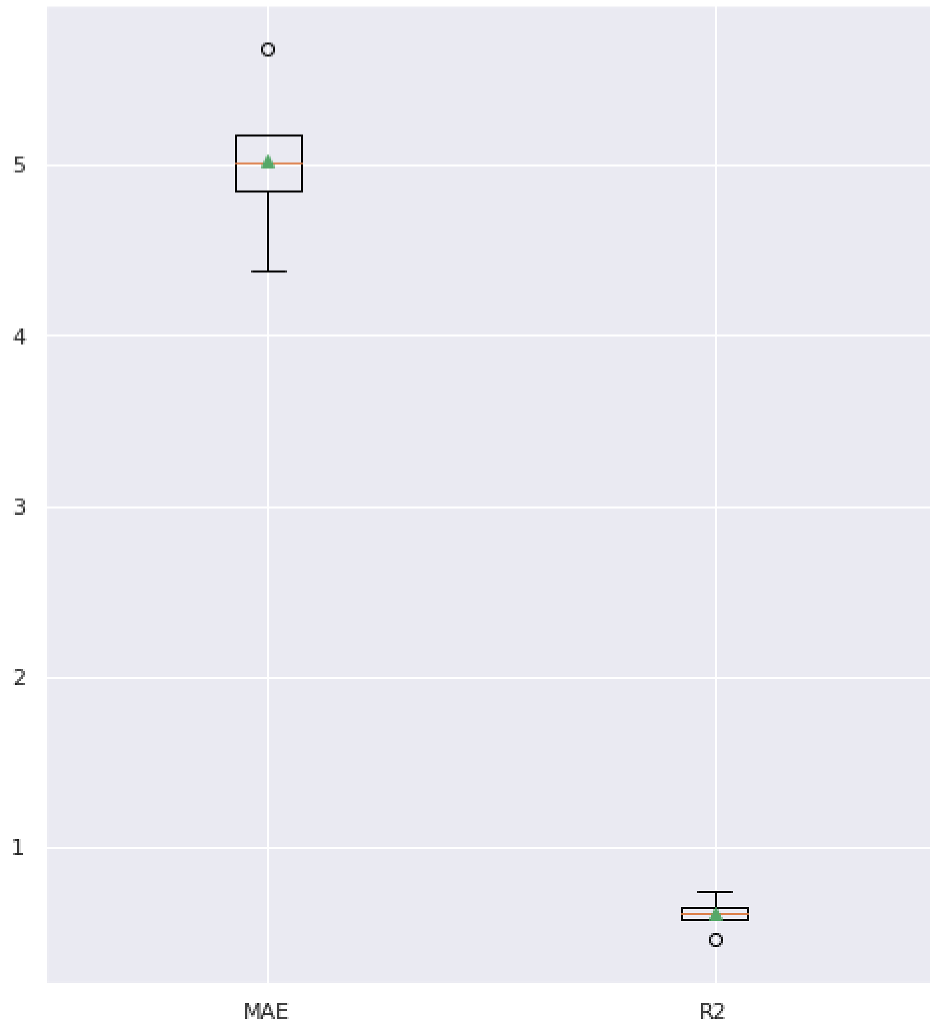


Grafica del **MAE** Box Plot

Se guardó correctamente

```
nombres.append('MAE')
nombres.append('R2')
```

```
#grafica del MAE (de los cuatro métodos)
sns.set(rc={'figure.figsize':(8,9)})
error_list = list()
error_list.append(lista_para_mae)
error_list.append(lista_para_r2)

plt.boxplot(error_list, labels=nombres, showmeans=True)
plt.show()
```



Ejercicio 2

Realiza la regresión polinomial de los siguientes datos:

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

Se guardó correctamente

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_
18388	2041000025	20141203T000000	474000.0	2	1.00	1090	
8011	2206700215	20140822T000000	375000.0	4	2.00	2070	
11884	5631501323	20140805T000000	309500.0	3	1.50	1340	1
134	2767602356	20150126T000000	675000.0	4	3.50	2140	
19910	7853360990	20150102T000000	430000.0	3	2.50	1950	
7096	2114300290	20140929T000000	411500.0	5	3.00	2420	
21461	7787920230	20150408T000000	518000.0	5	2.50	2890	1
11742	1994200260	20140819T000000	869900.0	6	4.50	2750	
12739	9264911210	20150226T000000	320000.0	5	3.00	2970	
21376	1282300995	20150222T000000	365000.0	3	2.25	1310	

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)
```



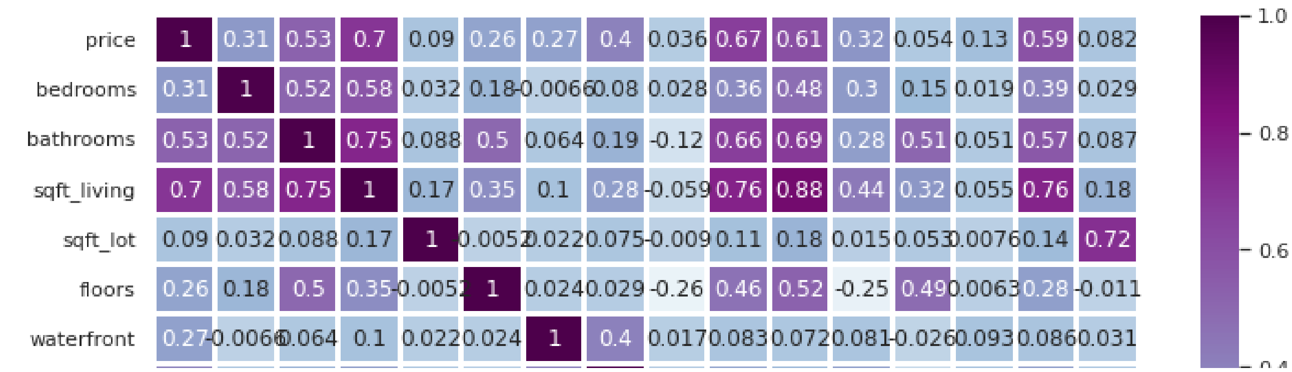
```
df.describe()
```

	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))
sns.heatmap(df.corr(), annot=True, cmap='BuPu', linewidths = 2)
plt.show()
```

Se guardó correctamente




```
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_sta

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}')
```

```
Numero total de registros en la bdd: 21613
**************************************************
Numero total de registros en el training set: 19451
Tamaño de X_train: (19451, 15)
**************************************************
Mumero total de registros en el test dataset: 2162
Tamaño del X_test: (2162, 15)
```

Se guardó correctamente

✓ 0 s se ejecutó 22:36

Se guardó correctamente