# Linear Models

## Ciencia y analítica de datos (Gpo 10)


**Alumno: Armando Bringas Corpus (A01200230)**

**Profesores: Dra. María de la Paz Rico Fernández, Mtra. Victoria Guerrero Orozco**


**Fecha: 9 de noviembre de 2022**

- 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.

```
from jupyterthemes import jtplot
jtplot.style(theme='monokai', context ='notebook', ticks =True, grid =False)
```

```
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.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
```

```
# to make this notebook's output stable across runs
np.random.seed(42)
```

## Simple Linear Regression

Simple linear regression equation:

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

Generate linear-looking data with the equation:

$y = 3X + 4 + noise$

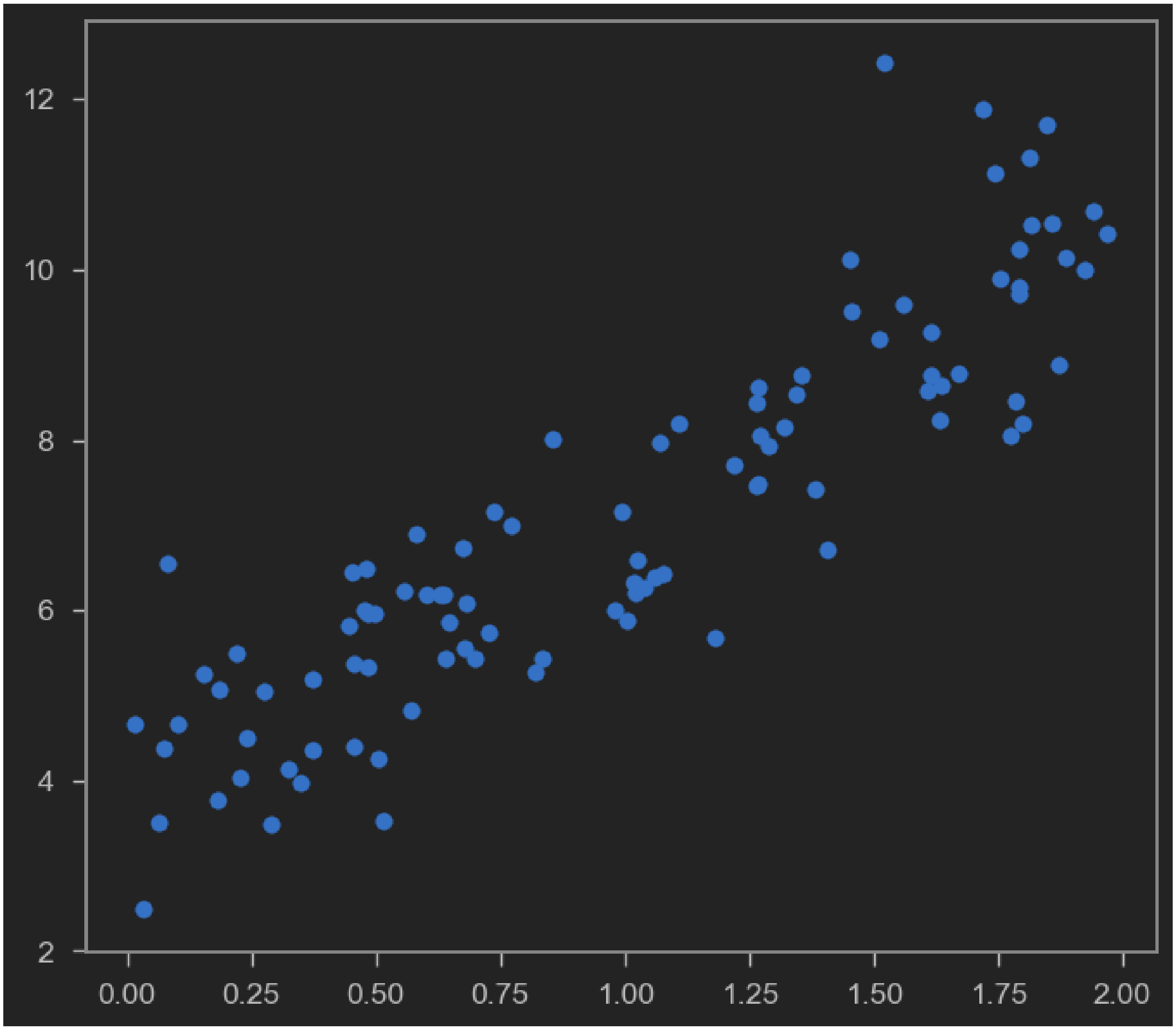
```
np.random.rand(100, 1)
```

```
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);
```




```
pd.DataFrame(y)
```

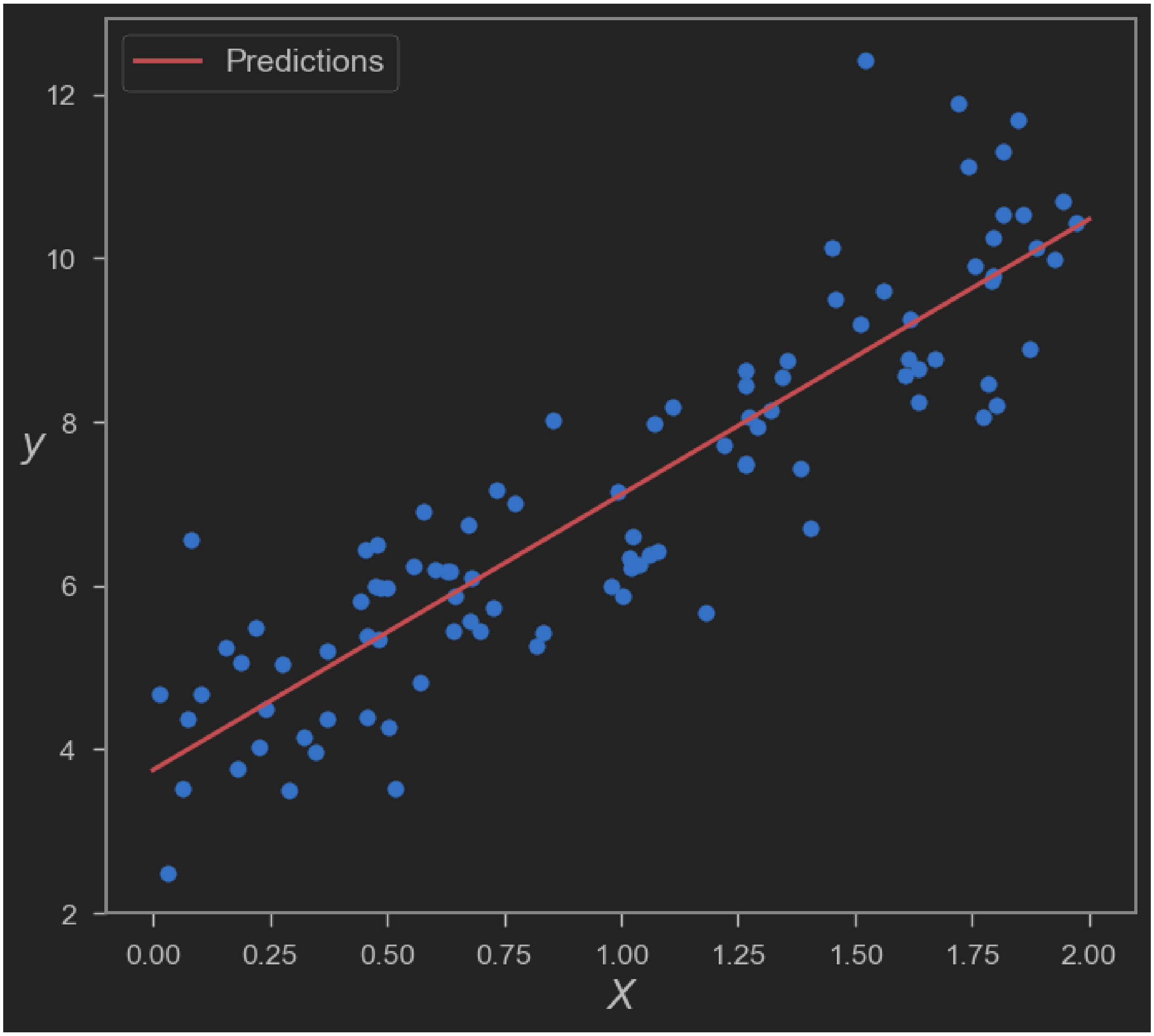
```
            0
0    3.508550
1    8.050716
2    6.179208
3    6.337073
4   11.311173
..        ...
95   5.441928
96  10.121188
97   9.787643
98   8.061635
99   9.597115

[100 rows x 1 columns]
```

```
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:

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

```
#|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.

$$y = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + ...$$

to

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + ...$$

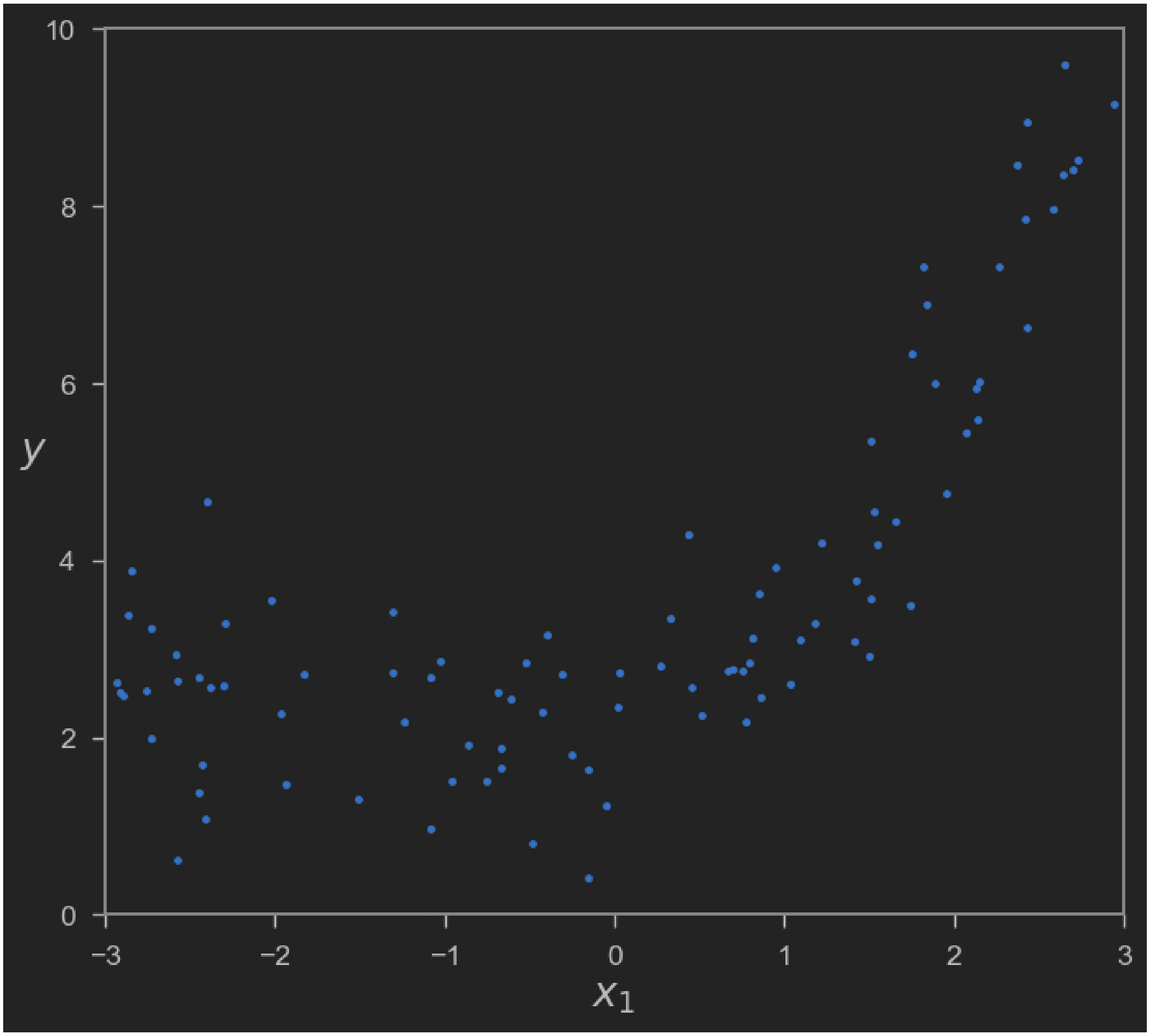
This is still a linear model, the linearity refers to the fact that the coefficients never multiply or divide each other.

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]);
```




```
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.

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

```python
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_
```

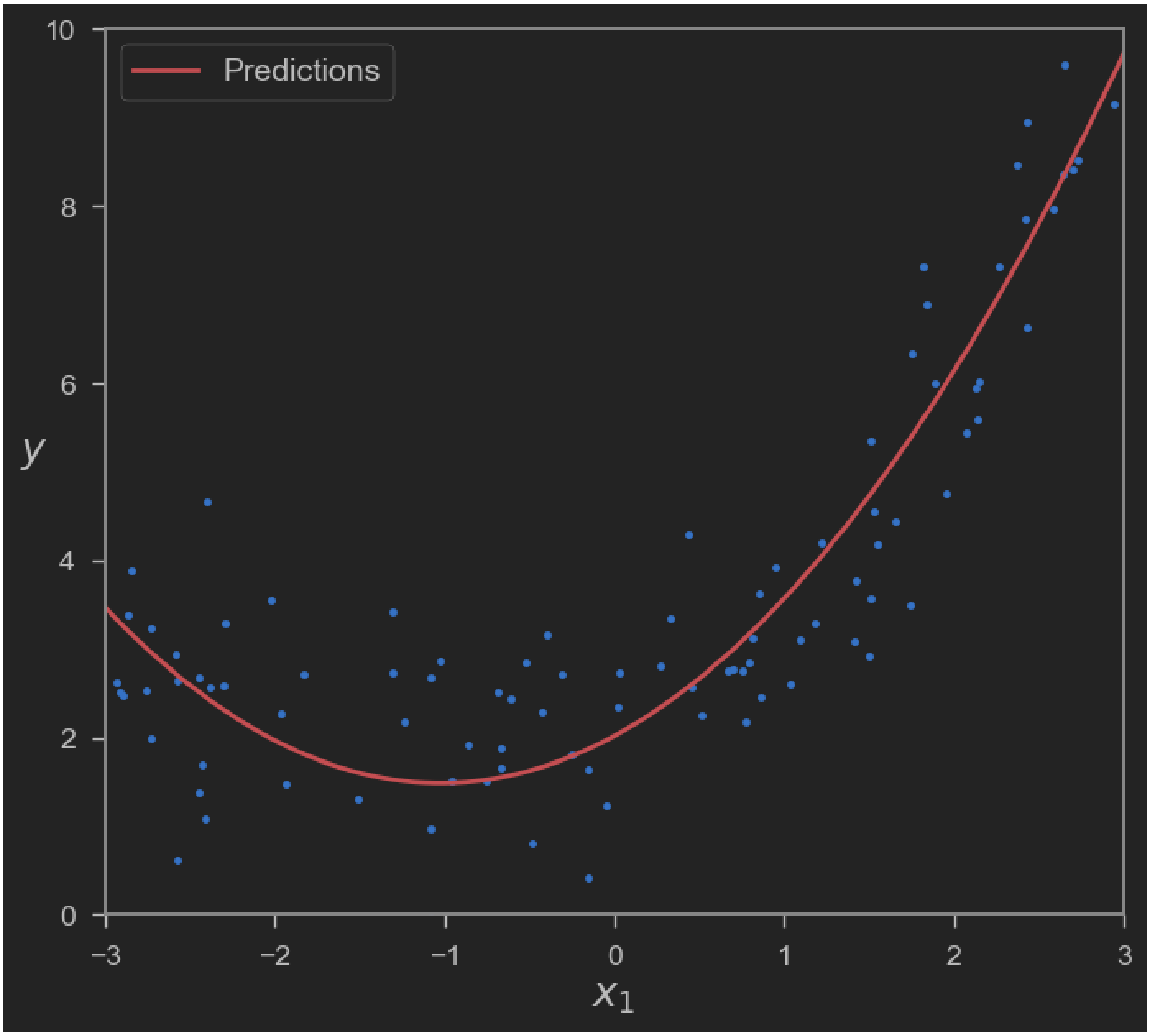
```
(array([[1.04271531, 0.50866711]]), array([2.01873554]))
```

The model estimates:

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

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 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

Sum Squared Regression Error

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

Sum Squared Total Error

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

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

## 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.

```
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']]
```

```python
y = df['Manufacturing Cost']
```

```python
len(X)
```

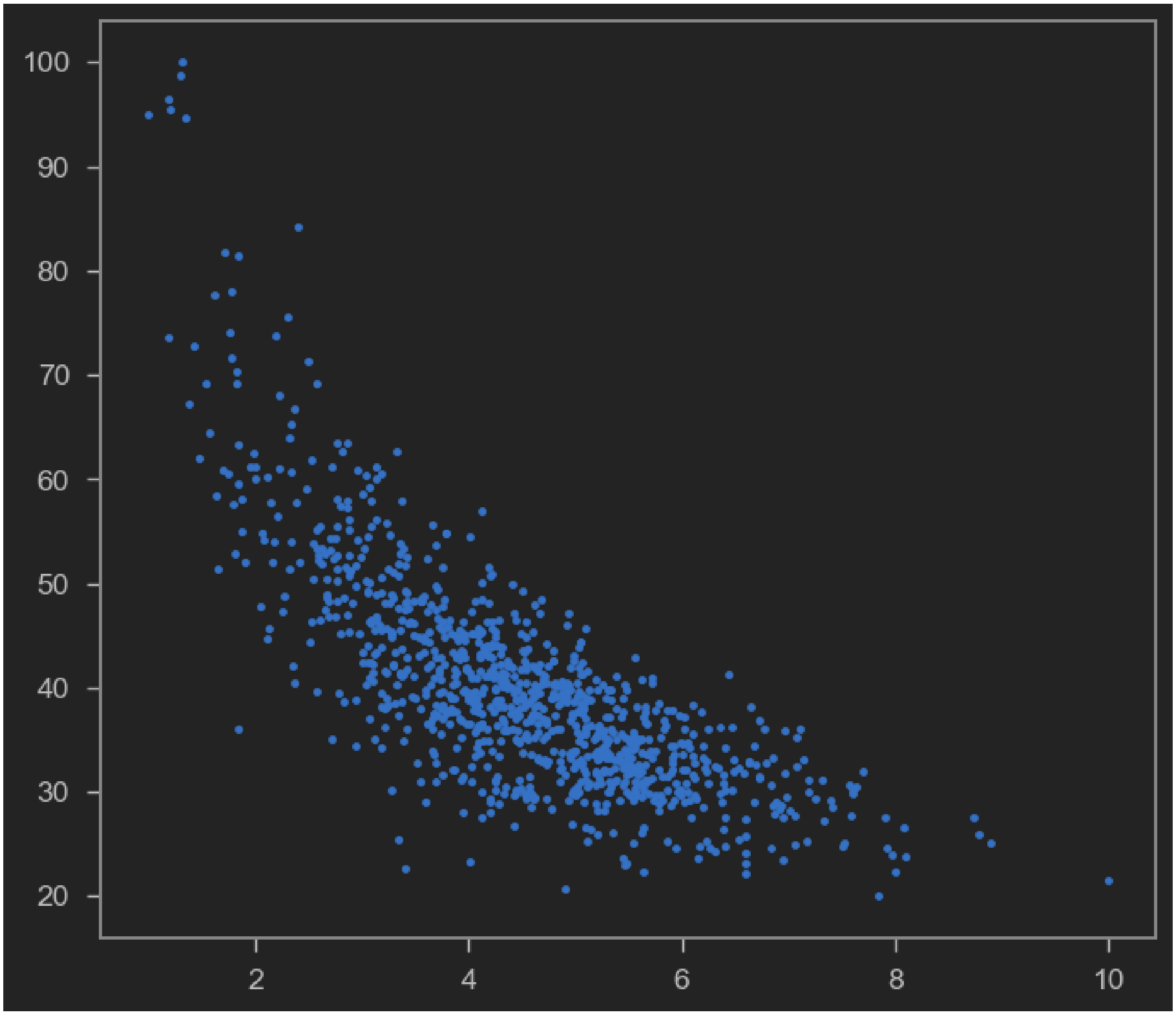
```
1000
```

```python
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>
```

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

```
[<matplotlib.lines.Line2D at 0x1f1b2cffd60>]
```




### Datos de entrenamiento y prueba, Funciones Predeterminadas

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

```
# Función para cálculo de errores MAE, RMSE y R-Squered
def calculo_errores(y, y_predict):
    """
    Funcion para calular los errores MAE, RMSE y R-squared
    y -> "y original"
    y -> "y predictora"
    """
    error_MAE =  metrics.mean_absolute_error(y, y_predict)
    error_RMSE = np.sqrt(metrics.mean_squared_error(y, y_predict))
    error_r2 = r2_score(y, y_predict)

    print('Error medio Absoluto (MAE):', error_MAE)
    print('Root Mean Squared Error:', error_RMSE)
    print('r2_score', error_r2)

    return error_MAE, error_RMSE, error_r2
```

La siguiente función la tomamos del curso de IBM con algunas ligeramos modificaciones

```
def PollyPlot(xtrain, xtest, y_train, y_test, lr, poly_transform, title='Regression'):
    """
    training data
    testing data
    lr:  linear regression object
    poly_transform:  polynomial transformation object
    """
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    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))), linewidth=3, label='Predict
    plt.title(title, fontsize=20)
    plt.ylabel('Y')
    plt.xlabel('X')
    plt.legend()
```

## Regresión Lineal

### Modelo

```
lin_reg_model = LinearRegression(fit_intercept=True)
lin_reg_model.fit(X_train, y_train)
y_predict = lin_reg_model.predict(X)
```

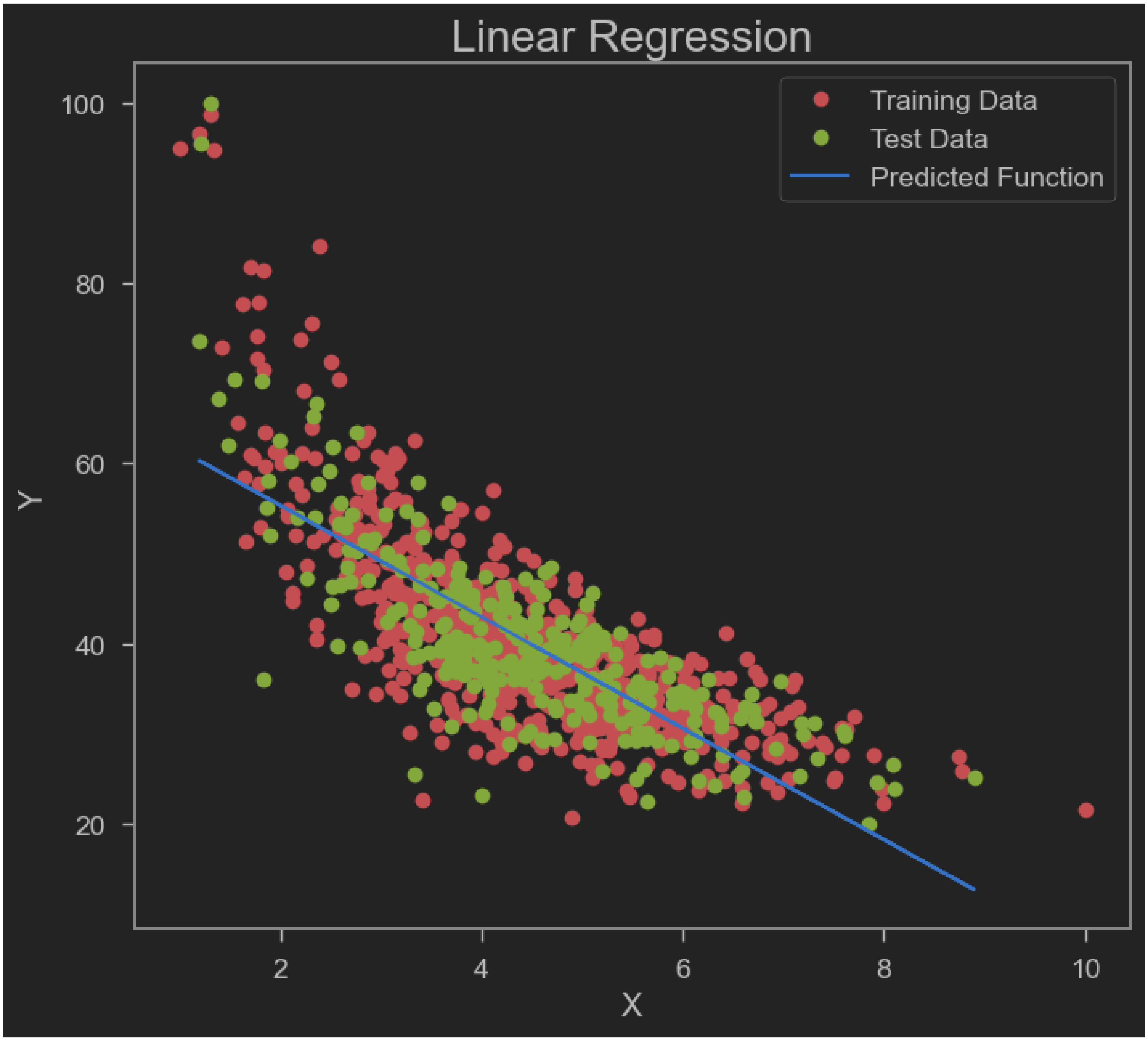
### Errores (MAE, RMSE, $R^2$)

```
lin_reg_MAE, lin_reg_RMSE, lin_reg_r2 = calculo_errores(y, y_predict)
```

```
Error medio Absoluto (MAE): 4.939597008496299
Root Mean Squared Error: 6.874192552027152
r2_score 0.5786435857137199
```

### Visualización

```
y_hat = lin_reg_model.predict(X_test)

plt.plot(X_train, y_train, 'ro', label='Training Data')
plt.plot(X_test, y_test, 'go', label='Test Data')
plt.plot(X_test, y_hat, label='Predicted Function')
plt.title('Linear Regression', fontsize=20)
plt.ylabel('Y')
plt.xlabel('X')
plt.legend()
plt.show()
```




### Regresión Polinomial

#### Modelo

```
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_poly = poly_features.fit_transform(X_train.values)
X_test_poly = poly_features.fit_transform(X_test.values)

poly_reg_model = LinearRegression(fit_intercept=True)
poly_reg_model.fit(X_train_poly, y_train)
y_predict = poly_reg_model.predict(X_poly)
```

#### Errores (MAE, RMSE, $R^2$)

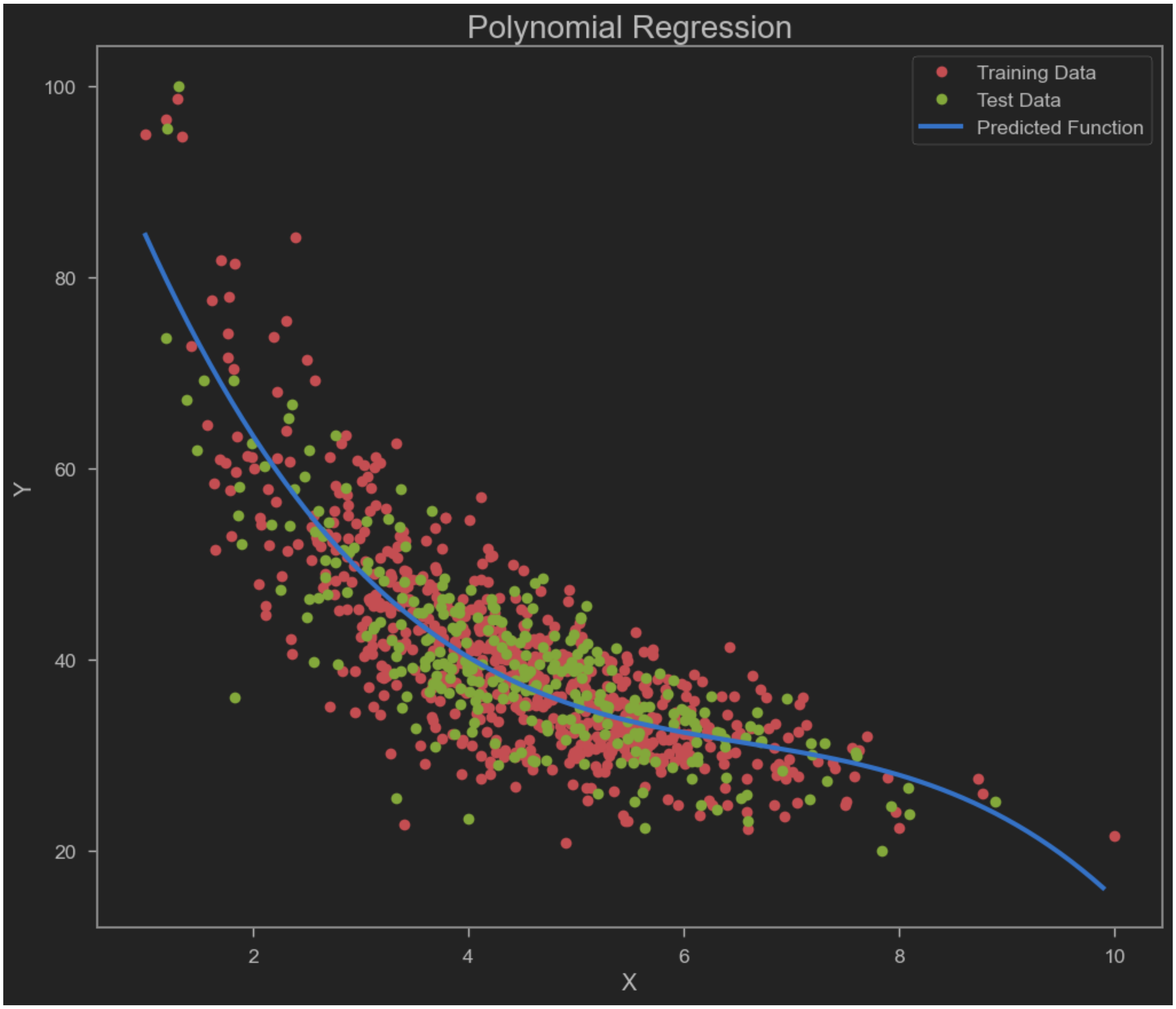
```
poly_reg_MAE, poly_reg_RMSE, poly_reg_r2 = calculo_errores(y, y_predict)
```

```
Error medio Absoluto (MAE): 4.491695145471314
Root Mean Squared Error: 5.931601464733989
```

```
r2_score 0.6862742608497481
```

### Visualización

```
PollyPlot(X_train, X_test, y_train, y_test, poly_reg_model, poly_features, 'Polynomial Regression')
```




## Regresión con Ridge

### Modelo

```
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_pr = poly_features.fit_transform(X_train)
X_test_pr = poly_features.fit_transform(X_test)

alpha = 0.005

ridge_model = Pipeline([
    ("poly_features", poly_features),
    ("scaler", StandardScaler()),
    ("ridge", Ridge(alpha=alpha, solver='cholesky', random_state=42))
```

```
])

ridge_model.fit(X_train_pr, y_train)

y_pred = ridge_model.predict(X_poly)
```

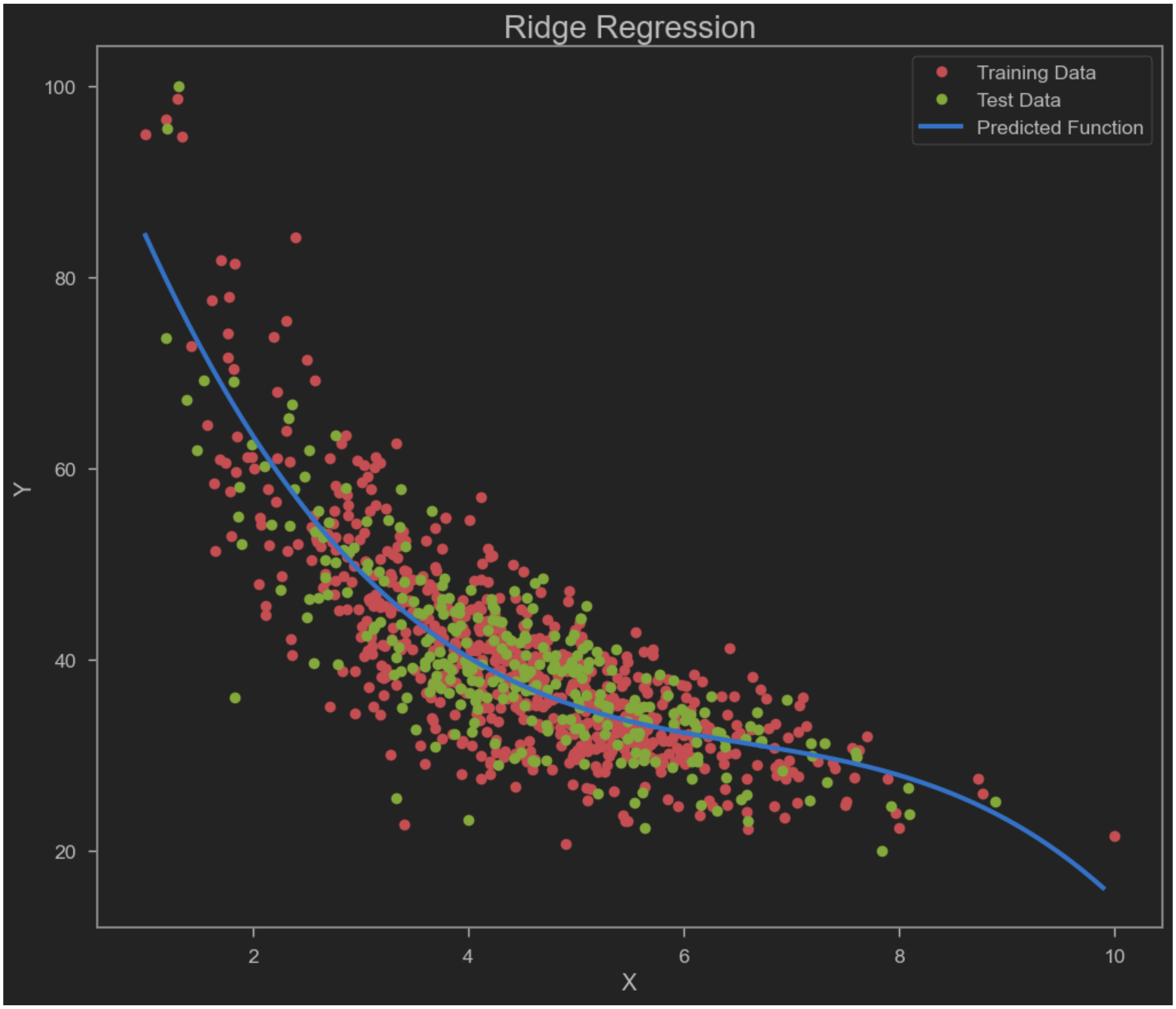
### Errores (MAE, RMSE, $R^2$)

```
ridge_reg_MAE, ridge_reg_RMSE, ridge_reg_r2 = calculo_errores(y, y_pred)
```

```
Error medio Absoluto (MAE): 4.439115075861926
Root Mean Squared Error: 5.862760553595349
r2_score 0.6935140728267526
```

### Visualización

```
reg_model_obj = Ridge(alpha=alpha)
reg_model_obj.fit(X_train_pr, y_train)

PollyPlot(X_train, X_test, y_train, y_test, reg_model_obj, poly_features, 'Ridge Regression')
```



## Regresión con Lasso

### Modelo

```
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_pr = poly_features.fit_transform(X_train)
X_test_pr = poly_features.fit_transform(X_test)

alpha = 0.5

lasso_model = Pipeline([
    ("poly_features", poly_features),
    ("scaler", StandardScaler()),
    ("lasso", Lasso(alpha=alpha, random_state=42))
])

lasso_model.fit(X_train_pr, y_train)

y_pred = lasso_model.predict(X_poly)
```

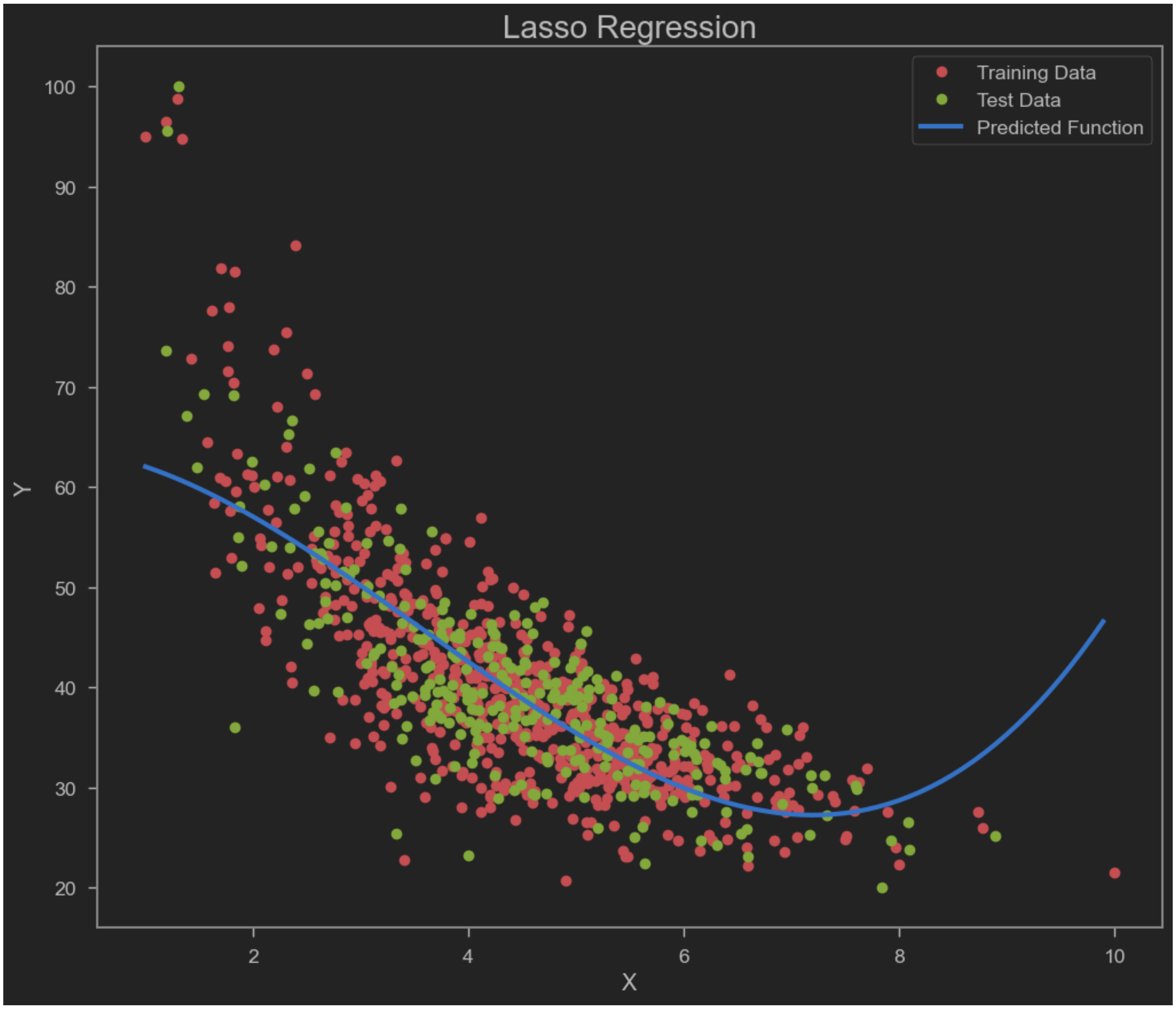
### Errores (MAE, RMSE, $R^2$)

```
lasso_reg_MAE, lasso_reg_RMSE, lasso_reg_r2 = calculo_errores(y, y_pred)
```

```
Error medio Absoluto (MAE): 4.71793606853768
Root Mean Squared Error: 6.6297462541716685
r2_score 0.6080776442471629
```

### Visualización

```
lasso_model_obj = Lasso(alpha=alpha, random_state=42)
lasso_model_obj.fit(X_train_pr, y_train)

PollyPlot(X_train, X_test, y_train, y_test, lasso_model_obj, poly_features, 'Lasso Regression')
```



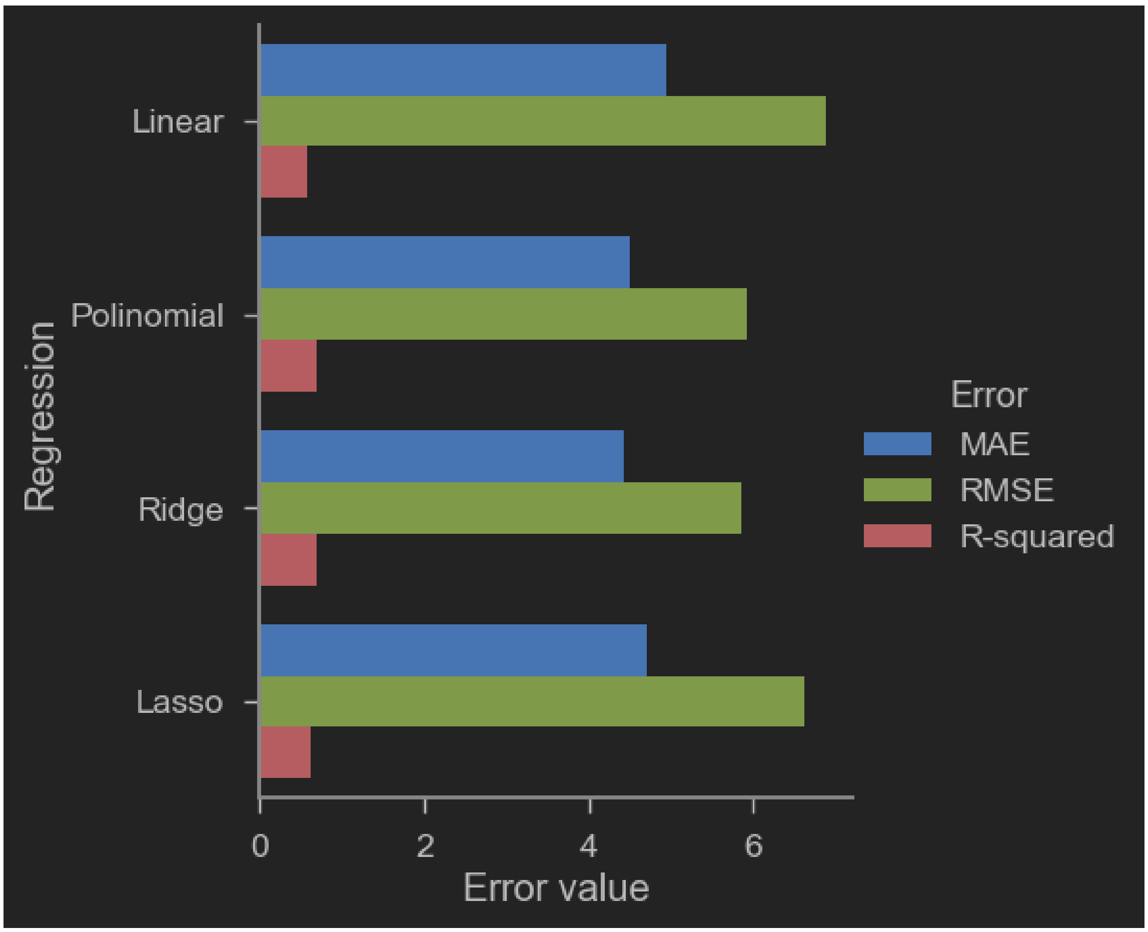

### Comparación de MAE, RMSE y $R^2$ de los cuatro métodos

```
error_comparison = {
    'Method' : ['Linear', 'Polinomial', 'Ridge', 'Lasso'],
    'MAE' : [lin_reg_MAE, poly_reg_MAE, ridge_reg_MAE, lasso_reg_MAE],
    'RMSE' : [lin_reg_RMSE, poly_reg_RMSE, ridge_reg_RMSE, lasso_reg_RMSE],
    'R-squared' : [lin_reg_r2, poly_reg_r2, ridge_reg_r2, lasso_reg_r2],
}

error_comparison = {
    'Error' : ['MAE', 'RMSE', 'R-squared'],
    'Linear' : [lin_reg_MAE, lin_reg_RMSE, lin_reg_r2],
    'Polinomial' : [poly_reg_MAE, poly_reg_RMSE, poly_reg_r2],
    'Ridge' : [ridge_reg_MAE, ridge_reg_RMSE, ridge_reg_r2],
    'Lasso' : [lasso_reg_MAE, lasso_reg_RMSE, lasso_reg_r2]
}

error_comparison_df = pd.DataFrame(error_comparison)
error_comparison_df
```

```
       Error    Linear  Polinomial     Ridge     Lasso
0        MAE  4.939597    4.491695  4.439115  4.717936
1       RMSE  6.874193    5.931601  5.862761  6.629746
2  R-squared  0.578644    0.686274  0.693514  0.608078
```

'Catplot' para visualizar los errores de los métodos

```
# Aplicamos un 'melt' que es un 'unpivot' de los datos para que podamos graficarlos
error_comparison_melt_df = pd.melt(
    error_comparison_df,
    id_vars="Error",
    var_name="Regression",
    value_name="Error value")

sns.catplot(data=error_comparison_melt_df, x="Error value", y="Regression", hue='Error', kind="bar")
plt.show()
```




## Explicación de los resultados

Sugerimos ir por el de Ridge ya que es el que cuenta con el menor error ya que se aprecia que podría tener un mejor ajuste, es muy similar su ajuste con respecto al de Regresión Polinomial, ambos tienen un margen de error muy cercano por lo que podemos apreciar en la gráfica de 'catplot'. Sin embargo, para poder determinar de forma más contundente el modelo, recomendamos que se haga un evaluación de puntaje de validación cruzada (cross validation).

Los porcentajes de entrenamiento son de 70% y de prueba de 30% que son con los que generalmente se comienzan, sin embargo, recomendamos hacer un análisis de 'underfitting' y 'overfitting' para analizar si los resultados no están subajustados o sobreajustados y de esta manera hacer una redefinición de los porcentajes en caso de ser necesario.

Los valores de error son aceptables para este modelo, tenemos un MAE and RMSE dentro de la escala de valores que esperamos, por otro lado, el valor de $R^2 > 0$, que nos indica que no hubo algún problema con el ajuste, en ridge y polinomial $R^2 \approx 0.7$, el modelo ideal, que ajustará perfectamente tendría una $R^2 = 1$ lo cual nos habla de que nuestro error es aceptable.

## 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)
```

```
               id             date      price  bedrooms  bathrooms  \
5954   7852020250  20140602T000000   725995.0         4       2.50   
8610   6392002020  20150324T000000   559000.0         3       1.75   
7650    626049058  20150504T000000   275000.0         5       2.50   
5683   2202500255  20150305T000000   335000.0         3       2.00   
20772  1972200428  20140625T000000   563500.0         3       2.50   
6959    723000114  20140505T000000  1395000.0         5       3.50   
10784  4104900340  20150204T000000   710000.0         4       2.50   
21528  3416600750  20150217T000000   585000.0         3       2.50   
12319  2386000070  20141029T000000   795127.0         4       3.25   
19947  1776460110  20141223T000000   395000.0         4       2.75   

       sqft_living  sqft_lot  floors  waterfront  view  ...  grade  \
5954          3190      7869     2.0           0     2  ...      9   
8610          1700      6500     1.0           0     0  ...      8   
7650          2570     17234     1.0           0     0  ...      7   
5683          1210      9926     1.0           0     0  ...      7   
20772         1400      1312     3.5           0     0  ...      8   
6959          4010      8510     2.0           0     1  ...      9   
10784         3220     18618     2.0           0     1  ...     10   
21528         1750      1381     3.0           0     0  ...      8   
12319         4360     91158     1.0           0     0  ...     10   
19947         2280      5013     2.0           0     0  ...      8   

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
5954         3190              0      2001             0    98065  47.5317   
8610         1700              0      1967             0    98115  47.6837   
7650         1300           1270      1959             0    98133  47.7753   
5683         1210              0      1954          2015    98006  47.5731   
20772        1400              0      2007             0    98103  47.6534   
6959         2850           1160      1971             0    98105  47.6578   
10784        3220              0      1991             0    98056  47.5326   
21528        1750              0      2008             0    98122  47.6021   
12319        3360           1000      1993             0    98053  47.6398   
19947        2280              0      2009             0    98019  47.7333   

          long  sqft_living15  sqft_lot15  
5954  -121.866           2630        6739  
8610  -122.284           1880        6000  
7650  -122.355           1760        7969  
```

```
5683  -122.135          1690        9737
20772 -122.355          1350        1312
6959  -122.286          2610        6128
10784 -122.181          2650       11896
21528 -122.294          1940        4800
12319 -121.985          3540       90940
19947 -121.976          2130        5121

[10 rows x 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
```

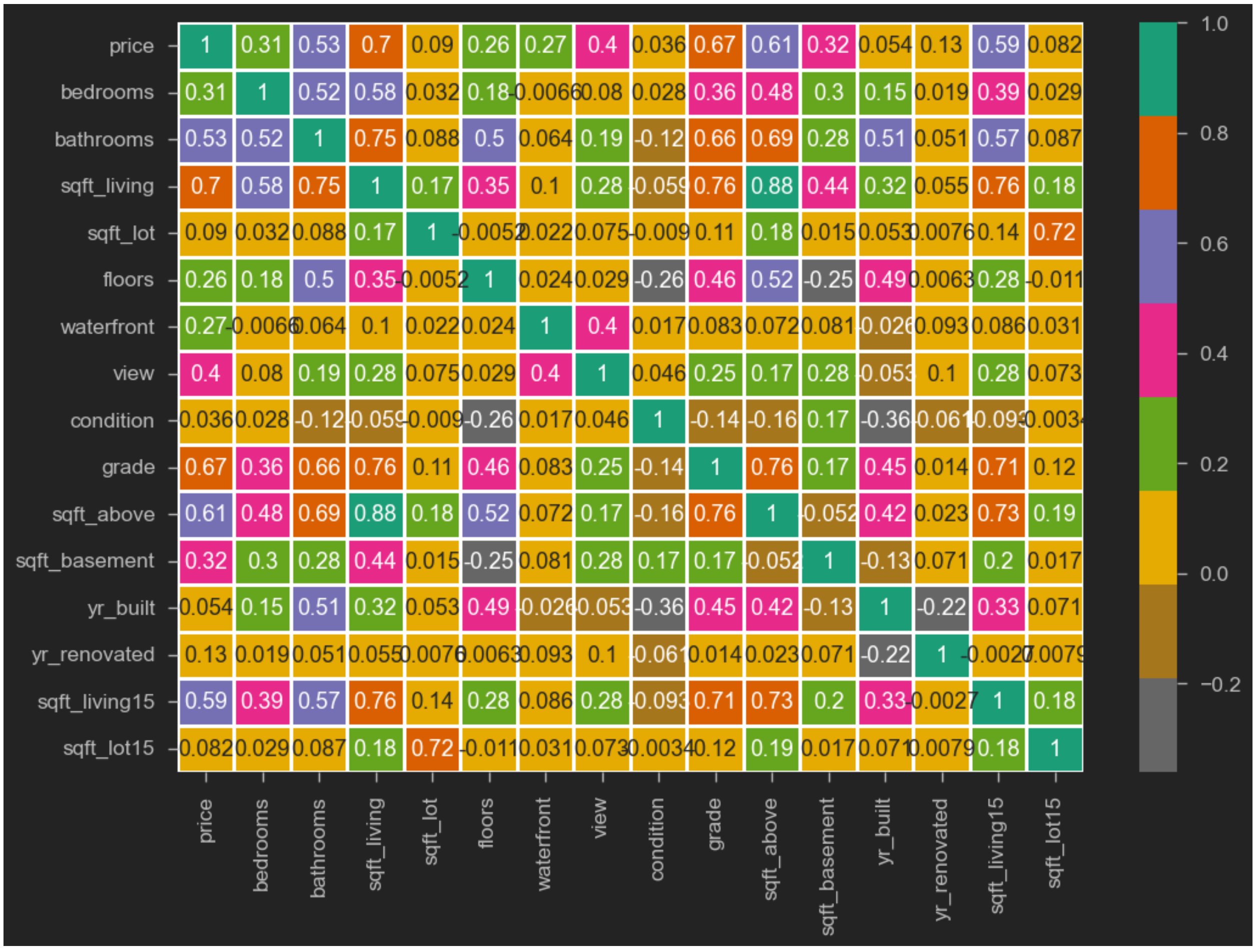
```
df.describe()
```

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

           sqft_lot        floors    waterfront          view     condition  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000  21613.000000
mean   1.510697e+04      1.494309      0.007542      0.234303      3.409430
```

```
std    4.142051e+04     0.539989     0.086517     0.766318     0.650743
min    5.200000e+02     1.000000     0.000000     0.000000     1.000000
25%    5.040000e+03     1.000000     0.000000     0.000000     3.000000
50%    7.618000e+03     1.500000     0.000000     0.000000     3.000000
75%    1.068800e+04     2.000000     0.000000     0.000000     4.000000
max    1.651359e+06     3.500000     1.000000     4.000000     5.000000

              grade    sqft_above  sqft_basement      yr_built  yr_renovated  \
count  21613.000000  21613.000000   21613.000000  21613.000000  21613.000000
mean       7.656873   1788.390691     291.509045   1971.005136     84.402258
std        1.175459    828.090978     442.575043     29.373411    401.679240
min        1.000000    290.000000       0.000000   1900.000000      0.000000
25%        7.000000   1190.000000       0.000000   1951.000000      0.000000
50%        7.000000   1560.000000       0.000000   1975.000000      0.000000
75%        8.000000   2210.000000     560.000000   1997.000000      0.000000
max       13.000000   9410.000000    4820.000000   2015.000000   2015.000000

            zipcode           lat          long  sqft_living15     sqft_lot15
count  21613.000000  21613.000000  21613.000000   21613.000000   21613.000000
mean   98077.939805     47.560053   -122.213896    1986.552492   12768.455652
std       53.505026      0.138564      0.140828     685.391304   27304.179631
min    98001.000000     47.155900   -122.519000     399.000000     651.000000
25%    98033.000000     47.471000   -122.328000    1490.000000    5100.000000
50%    98065.000000     47.571800   -122.230000    1840.000000    7620.000000
75%    98118.000000     47.678000   -122.125000    2360.000000   10083.000000
max    98199.000000     47.777600   -121.315000    6210.000000  871200.000000
```

```python
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)
```

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



***Tuvimos que reducir el número de atributos a utilizar para lograr la convergencia de los modelos, si utilizábamos todos los atributos teníamos un error de desbordamiento de memoria, los atributos para estos modelos fueron seleccionados con base al 'heatmap', aquellos que tuvieran una mayor correlación con la 'Y' en este caso: 'sqft_living', 'grade', 'sqft_above', 'sqft_living15'***

```python
#columns = df.columns.drop('price')
#features = columns
features = ['sqft_living', 'grade', 'sqft_above', 'sqft_living15']
label = ['price']

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)

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: 15129
Tamaño de X_train: (15129, 4)
*************************************************
Mumero total de registros en el test dataset: 6484
Tamaño del X_test: (6484, 4)
```

## Regresión Lineal

### Modelo

```
lin_reg_model = LinearRegression(fit_intercept=True)
lin_reg_model.fit(X_train, y_train)
y_predict = lin_reg_model.predict(X)
```

### Errores (MAE, RMSE, R-squared)

```
lin_reg_MAE, lin_reg_RMSE, lin_reg_r2 = calculo_errores(y, y_predict)
```

```
Error medio Absoluto (MAE): 162006.60695814595
Root Mean Squared Error: 248483.81686359108
r2_score 0.5418758377891657
```

## Regresión Polinomial

### Modelo

```
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_poly = poly_features.fit_transform(X_train.values)
X_test_poly = poly_features.fit_transform(X_test.values)

poly_reg_model = LinearRegression(fit_intercept=True)
poly_reg_model.fit(X_train_poly, y_train)
y_predict = poly_reg_model.predict(X_poly)
```

### Errores (MAE, RMSE, $R^2$)

```
poly_reg_MAE, poly_reg_RMSE, poly_reg_r2 = calculo_errores(y, y_predict)
```

```
Error medio Absoluto (MAE): 147040.24271547626
Root Mean Squared Error: 232498.00190120665
r2_score 0.5989251510287183
```

## Ridge

### Modelo

```
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_pr = poly_features.fit_transform(X_train)
X_test_pr = poly_features.fit_transform(X_test)

alpha = 0.1

ridge_model = Pipeline([
    ("poly_features", poly_features),
```

```
    ("scaler", StandardScaler()),
    ("ridge", Ridge(alpha=alpha, solver='cholesky', random_state=42))
])

ridge_model.fit(X_train_pr, y_train)

y_pred = ridge_model.predict(X_poly)
```

**Error**

```
ridge_reg_MAE, ridge_reg_RMSE, ridge_reg_r2 = calculo_errores(y, y_pred)
```

```
Error medio Absoluto (MAE): 150725.9130162672
Root Mean Squared Error: 912340.9941843494
r2_score -5.175915677094912
```

## Lasso

### Modelo

```
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_pr = poly_features.fit_transform(X_train)
X_test_pr = poly_features.fit_transform(X_test)

alpha = 15

lasso_model = Pipeline([
    ("poly_features", poly_features),
    ("scaler", StandardScaler()),
    ("lasso", Lasso(alpha=alpha, tol=1e-2, random_state=42))
])

lasso_model.fit(X_train_pr, y_train)

y_pred = lasso_model.predict(X_poly)
```

```
C:\Users\bring\anaconda3\envs\tec\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:648: Co
  model = cd_fast.enet_coordinate_descent(
```

**Error**

```
lasso_reg_MAE, lasso_reg_RMSE, lasso_reg_r2 = calculo_errores(y, y_pred)
```

```
Error medio Absoluto (MAE): 147842.20914431222
Root Mean Squared Error: 412469.7475706219
r2_score -0.2623249920317994
```

## Visualización de los Errores en los cuatro métodos

```
error_comparison = {
    'Method' : ['Linear', 'Polinomial', 'Ridge', 'Lasso'],
    'MAE' : [lin_reg_MAE, poly_reg_MAE, ridge_reg_MAE, lasso_reg_MAE],
    'RMSE' : [lin_reg_RMSE, poly_reg_RMSE, ridge_reg_RMSE, lasso_reg_RMSE],
    'R-squared' : [lin_reg_r2, poly_reg_r2, ridge_reg_r2, lasso_reg_r2],
}
```

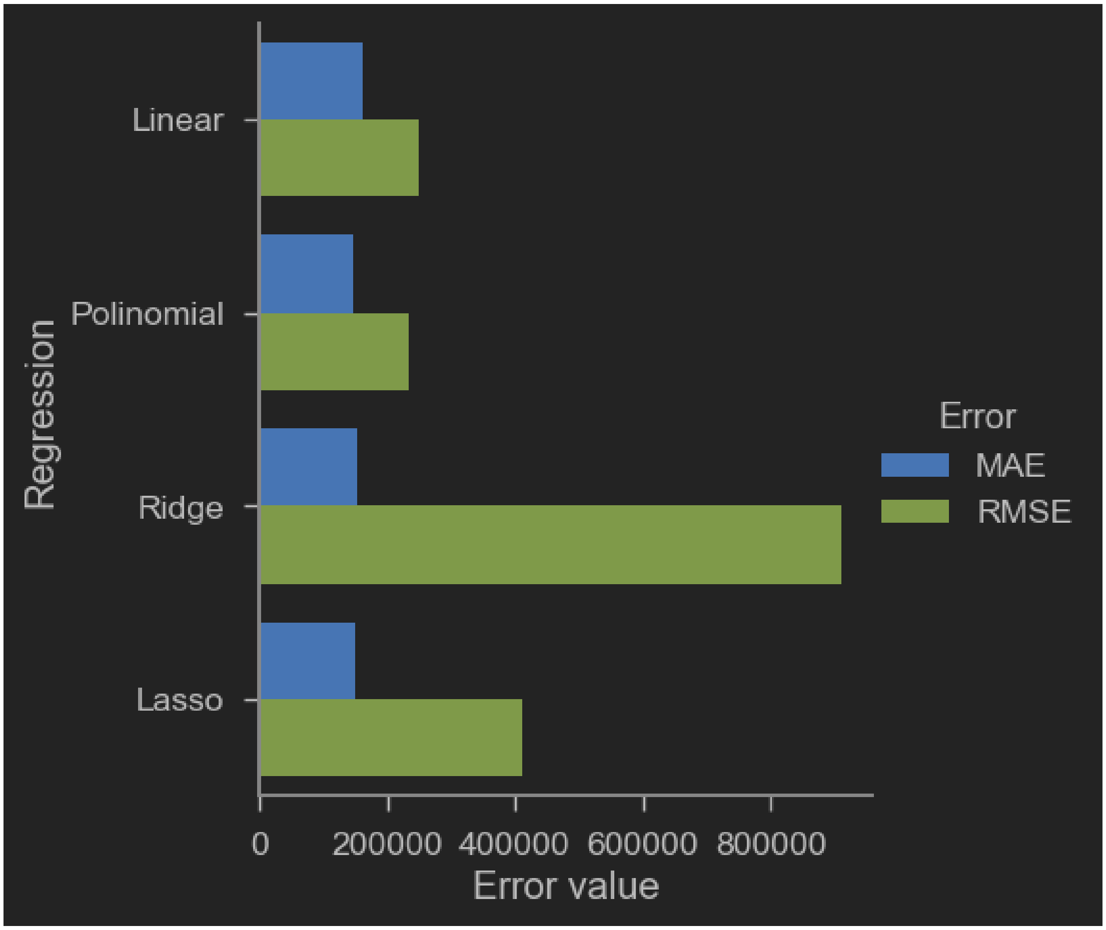
```python
error_comparison = {
    'Error' : ['MAE', 'RMSE', 'R-squared'],
    'Linear' : [lin_reg_MAE, lin_reg_RMSE, lin_reg_r2],
    'Polinomial' : [poly_reg_MAE, poly_reg_RMSE, poly_reg_r2],
    'Ridge' : [ridge_reg_MAE, ridge_reg_RMSE, ridge_reg_r2],
    'Lasso' : [lasso_reg_MAE, lasso_reg_RMSE, lasso_reg_r2]
}

error_comparison_df = pd.DataFrame(error_comparison)
error_comparison_df
```

| | Error | Linear | Polinomial | Ridge | Lasso |
|---|---|---|---|---|---|
| 0 | MAE | 162006.606958 | 147040.242715 | 150725.913016 | 147842.209144 |
| 1 | RMSE | 248483.816864 | 232498.001901 | 912340.994184 | 412469.747571 |
| 2 | R-squared | 0.541876 | 0.598925 | -5.175916 | -0.262325 |

```python
error_comparison_df1 = error_comparison_df.drop([2], axis=0, inplace=False)
error_comparison_df1.reset_index()
```

| | index | Error | Linear | Polinomial | Ridge | Lasso |
|---|---|---|---|---|---|---|
| 0 | 0 | MAE | 162006.606958 | 147040.242715 | 150725.913016 | 147842.209144 |
| 1 | 1 | RMSE | 248483.816864 | 232498.001901 | 912340.994184 | 412469.747571 |

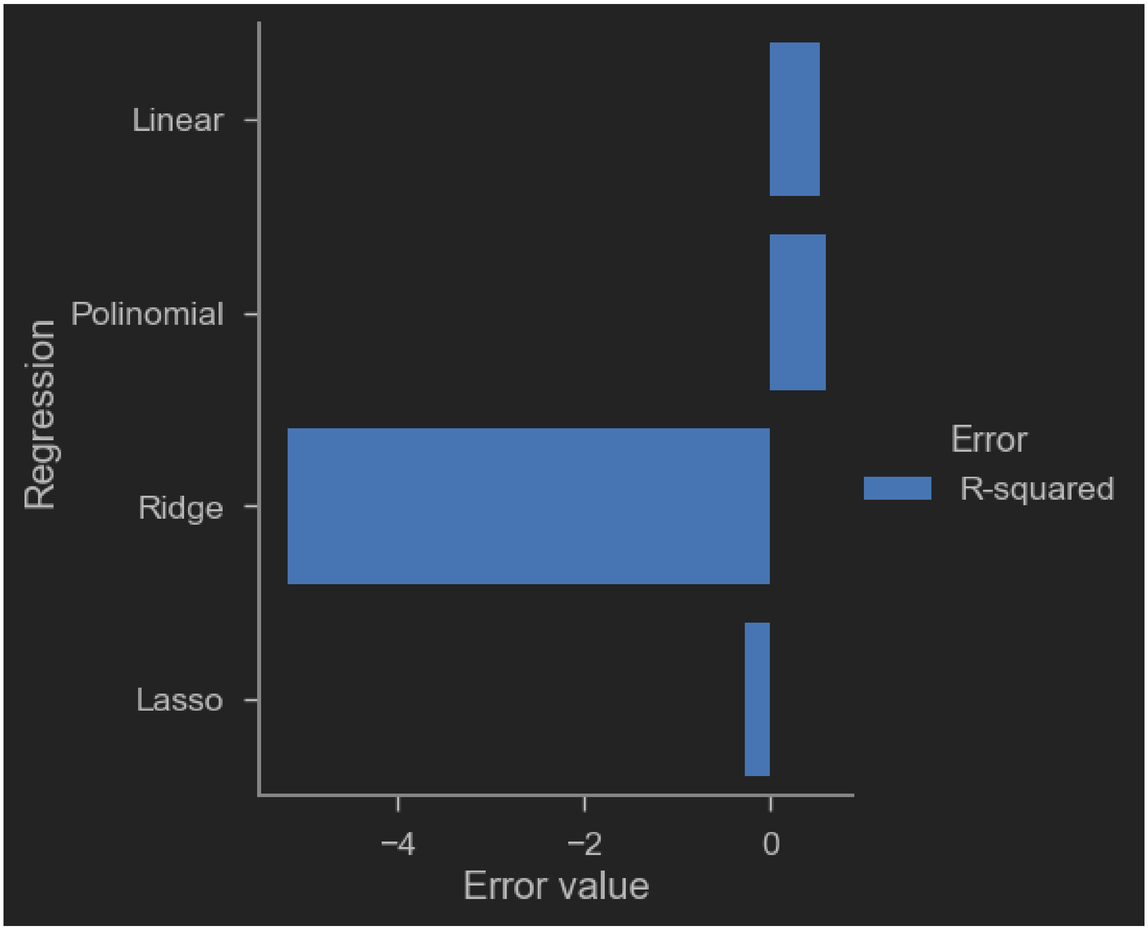
```python
# Aplicamos un 'melt' que es un 'unpivot' de los datos para que podamos graficarlos
error_comparison_melt_df1 = pd.melt(
    error_comparison_df1,
    id_vars="Error",
    var_name="Regression",
    value_name="Error value")

sns.catplot(data=error_comparison_melt_df1, x="Error value", y="Regression", hue='Error', kind="bar")
plt.show()
```




```
error_comparison_df2 = error_comparison_df.drop([0,1], axis=0, inplace=False)
error_comparison_df2.reset_index()
```

```
   index      Error    Linear  Polinomial     Ridge     Lasso
0      2  R-squared  0.541876    0.598925 -5.175916 -0.262325
```

```
# Aplicamos un 'melt' que es un 'unpivot' de los datos para que podamos graficarlos
error_comparison_melt_df2 = pd.melt(
    error_comparison_df2,
    id_vars="Error",
    var_name="Regression",
    value_name="Error value")

sns.catplot(data=error_comparison_melt_df2, x="Error value", y="Regression", hue='Error', kind="bar")
plt.show()
```




Como podemos observar con esta gráfica tenemos valores de $R^2 < 0$ para Ridge y Lasso, lo cual nos habla que el modelo no ajusto de forma adecuada con los datos, por lo tanto, no es un modelo válido. Particularmente lo podemos ver en Ridge donde no se logró la convergencia del modelo. Procederemos a realizar la repetición del ejercicio decrementando la cantidad de datos de entrenamiento y prueba y ajustando los parámetros en los modelos.

### Repetición del ejercio modificando los porcentajes de datos de entrenamiento y prueba

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, train_size = 0.35, random_s

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: 7564
Tamaño de X_train: (7564, 4)
**************************************************
Mumero total de registros en el test dataset: 3242
```

```
Tamaño del X_test: (3242, 4)
```

**Regresión Lineal**

```python
lin_reg_model = LinearRegression(fit_intercept=True)
lin_reg_model.fit(X_train, y_train)
y_predict = lin_reg_model.predict(X)

lin_reg_MAE, lin_reg_RMSE, lin_reg_r2 = calculo_errores(y, y_predict)
```

```
Error medio Absoluto (MAE): 164778.3916447209
Root Mean Squared Error: 248835.91020031332
r2_score 0.540576624447485
```

**Regresión Polinomial**

```python
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_poly = poly_features.fit_transform(X_train.values)
X_test_poly = poly_features.fit_transform(X_test.values)

poly_reg_model = LinearRegression(fit_intercept=True)
poly_reg_model.fit(X_train_poly, y_train)
y_predict = poly_reg_model.predict(X_poly)

poly_reg_MAE, poly_reg_RMSE, poly_reg_r2 = calculo_errores(y, y_predict)
```

```
Error medio Absoluto (MAE): 148054.32929226995
Root Mean Squared Error: 226183.7142848149
r2_score 0.6204144757159564
```

**Ridge**

```python
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_pr = poly_features.fit_transform(X_train)
X_test_pr = poly_features.fit_transform(X_test)

alpha = 10

ridge_model = Pipeline([
    ("poly_features", poly_features),
    ("scaler", StandardScaler()),
    ("ridge", Ridge(alpha=alpha, solver='cholesky', random_state=42))
])

ridge_model.fit(X_train_pr, y_train)

y_pred = ridge_model.predict(X_poly)

ridge_reg_MAE, ridge_reg_RMSE, ridge_reg_r2 = calculo_errores(y, y_pred)
```

```
Error medio Absoluto (MAE): 147416.11076243696
Root Mean Squared Error: 233653.59188662685
r2_score 0.5949283003652879
```

**Lasso**

```
poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X.values)
X_train_pr = poly_features.fit_transform(X_train)
X_test_pr = poly_features.fit_transform(X_test)

alpha = 10

lasso_model = Pipeline([
    ("poly_features", poly_features),
    ("scaler", StandardScaler()),
    ("lasso", Lasso(alpha=alpha, tol=1e-2, random_state=42))
])

lasso_model.fit(X_train_pr, y_train)

y_pred = lasso_model.predict(X_poly)

lasso_reg_MAE, lasso_reg_RMSE, lasso_reg_r2 = calculo_errores(y, y_pred)
```

```
C:\Users\bring\anaconda3\envs\tec\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:648: Co
  model = cd_fast.enet_coordinate_descent(
```

```
Error medio Absoluto (MAE): 147837.0988253276
Root Mean Squared Error: 241074.03642916318
r2_score 0.5687909589890132
```

**Visualización de los Errores en los cuatro métodos**

```
error_comparison = {
    'Method' : ['Linear', 'Polinomial', 'Ridge', 'Lasso'],
    'MAE' : [lin_reg_MAE, poly_reg_MAE, ridge_reg_MAE, lasso_reg_MAE],
    'RMSE' : [lin_reg_RMSE, poly_reg_RMSE, ridge_reg_RMSE, lasso_reg_RMSE],
    'R-squared' : [lin_reg_r2, poly_reg_r2, ridge_reg_r2, lasso_reg_r2],
}

error_comparison = {
    'Error' : ['MAE', 'RMSE', 'R-squared'],
    'Linear' : [lin_reg_MAE, lin_reg_RMSE, lin_reg_r2],
    'Polinomial' : [poly_reg_MAE, poly_reg_RMSE, poly_reg_r2],
    'Ridge' : [ridge_reg_MAE, ridge_reg_RMSE, ridge_reg_r2],
    'Lasso' : [lasso_reg_MAE, lasso_reg_RMSE, lasso_reg_r2]
}

error_comparison_df = pd.DataFrame(error_comparison)
error_comparison_df
```

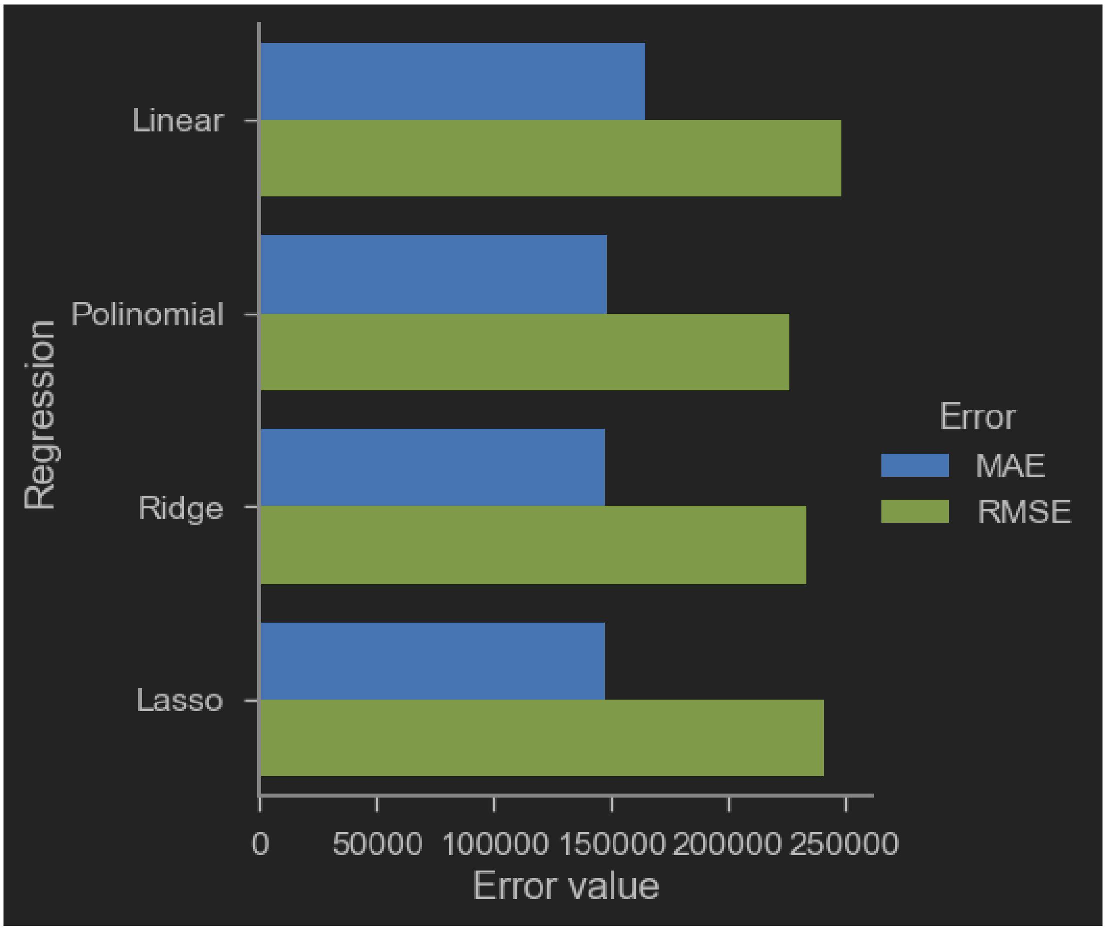
| | Error | Linear | Polinomial | Ridge | Lasso |
|---|---|---|---|---|---|
| 0 | MAE | 164778.391645 | 148054.329292 | 147416.110762 | 147837.098825 |
| 1 | RMSE | 248835.910200 | 226183.714285 | 233653.591887 | 241074.036429 |
| 2 | R-squared | 0.540577 | 0.620414 | 0.594928 | 0.568791 |

```
error_comparison_df1 = error_comparison_df.drop([2], axis=0, inplace=False)
error_comparison_df1.reset_index()
```

| index | Error | Linear | Polinomial | Ridge | Lasso |
|---|---|---|---|---|---|

```
0      0   MAE  164778.391645  148054.329292  147416.110762  147837.098825
1      1  RMSE  248835.910200  226183.714285  233653.591887  241074.036429
```

```
# Aplicamos un 'melt' que es un 'unpivot' de los datos para que podamos graficarlos
error_comparison_melt_df1 = pd.melt(
    error_comparison_df1,
    id_vars="Error",
    var_name="Regression",
    value_name="Error value")

sns.catplot(data=error_comparison_melt_df1, x="Error value", y="Regression", hue='Error', kind="bar")
plt.show()
```

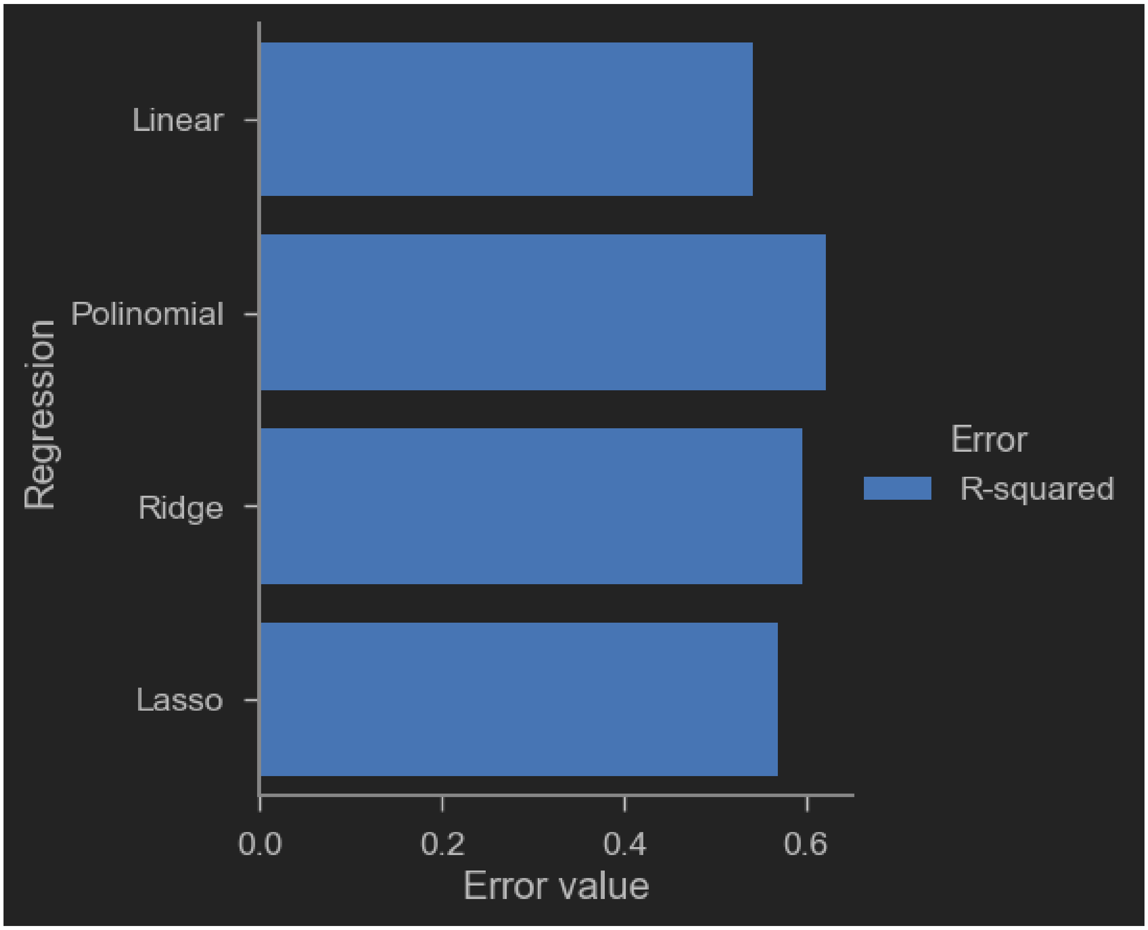



```
error_comparison_df2 = error_comparison_df.drop([0,1], axis=0, inplace=False)
error_comparison_df2.reset_index()
```

```
   index      Error    Linear  Polinomial     Ridge     Lasso
0      2  R-squared  0.540577    0.620414  0.594928  0.568791
```

```
# Aplicamos un 'melt' que es un 'unpivot' de los datos para que podamos graficarlos
error_comparison_melt_df2 = pd.melt(
    error_comparison_df2,
    id_vars="Error",
    var_name="Regression",
    value_name="Error value")
```

```
sns.catplot(data=error_comparison_melt_df2, x="Error value", y="Regression", hue='Error', kind="bar")
plt.show()
```




### Explicación de Resultados

Primero sugerimos descartar el método de Lasso por el 'Warning' que nos sale de que no se logra la convergencia del modelo, intentamos aumentar el grado de ajuste en el modelo pero tenemos un error de desbordamiento de memoria. Por lo tanto, decidimos irnos por el modelo de Regresión Polinomial ya que es el que cuenta con el menor error con respecto a los otros modelos. Sin embargo, para poder determinar de forma más contundente el modelo, recomendamos que se haga un evaluación de puntaje de validación cruzada (cross validation).

Los porcentajes de entrenamiento fueron de 70% y 30% pero sobre el 50% de los datos totales para lograr conseguir la convergencia de los modelos, sin embargo, como habíamos comentado en la primera parte recomendamos hacer un análisis de 'underfitting' y 'overfitting' para analizar si los resultados no están subajustados o sobreajustados y de esta manera hacer una redefinición de los porcentajes en caso de ser necesario.

Los valores de error hay que revisarlos ya que se tienen muy altos para MAE y RMSE, podría deberse por la multidimensionalidad del modelo, aunque también consideramos revisar la precencia de outliers. Por otro lado, el valor de $R^2 > 0$, que nos indica que no hubo algún problema con el ajuste, en ridge y polinomial

$R^2 \approx 0.6$, el modelo ideal, que ajustará perfectamente tendría una $R^2 = 1$ lo cual nos habla de que nuestro error podría ser aceptable.

## Conclusiones

Con respecto al Ejercicio 1 no se presentaron complicaciones importantes para poder lograr la convergencia de los modelos y su ajuste para obtener la menor cantidad de error. Sin embargo en los modelos de regresión múltiple tuvimos problema para lograr la convergencia de los modelos, esto se puede deber al tamaño del 'dataset' y la gran multidimensional del modelo debido a la cantidad de atributos que se tenían.

Para poder hacerle frente a este desafío propusimos disminuir la cantidad de atributos, seleccionar sólo aquellos que consideramos los más importantes y de igual manera disminuir la cantidad de datos de entrenamiento y validación para evitar el desbordamiento de la memoria. Sin embargo, pese a que conseguimos que convergieran en la mayoría de los modelos y calcular el error, consideramos necesario hacer un análisis más exhaustivo como un PCA para explorar otras técnicas de reducción de complejidad y así explorar si se alcanza un mejor ajuste de los modelos y su modificación de parámetros para reducir el error.