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
from sklearn.model_selection import train_test_split
# to make this notebook's output stable across runs
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
np.random.seed(42)
```

```
5-2
```

```
3
```

Simple Linear Regression

Simple linear regression equation:

$y = ax + b$

a: slope

b: intercept

Generate linear-looking data with the equation:

$y = 3X + 4 + noise$

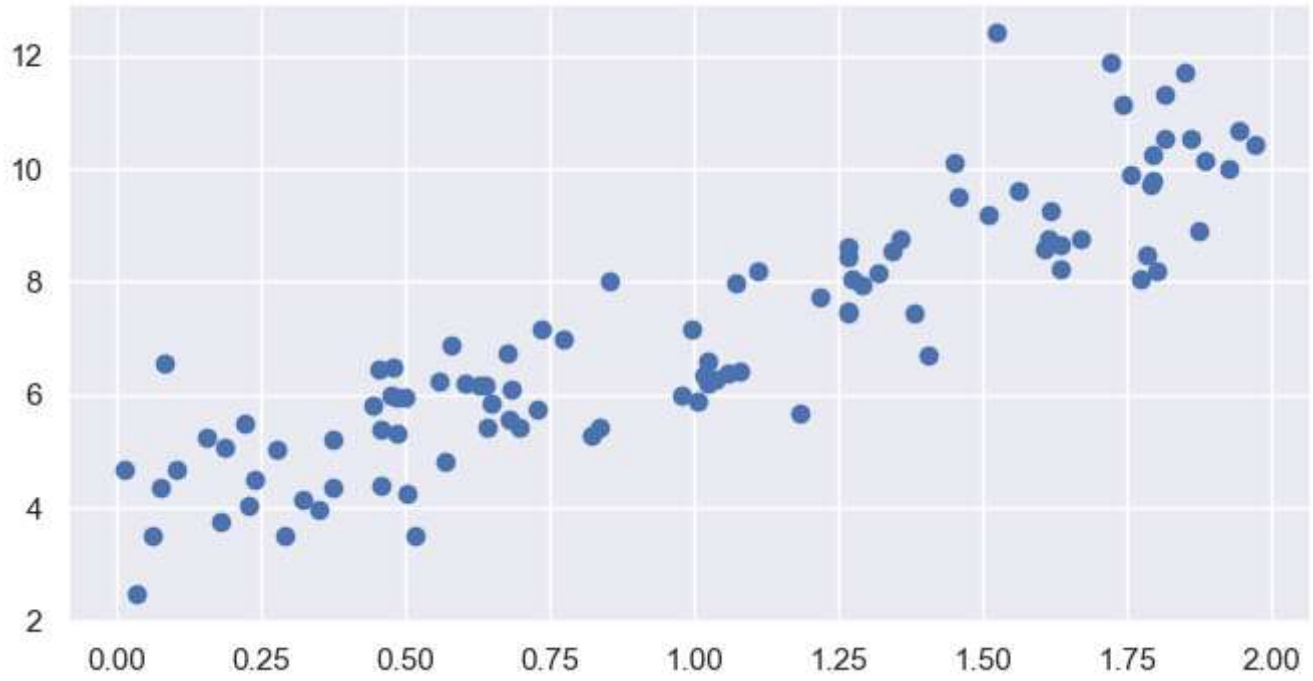
```
np.random.rand(100, 1)
```

```
array([[0.37454012],
       [0.95071431],
       [0.73199394],
```

```
[0.59865848],
[0.15601864],
[0.15599452],
[0.05808361],
[0.86617615],
[0.60111501],
[0.70807258],
[0.02058449],
[0.96990985],
[0.83244264],
[0.21233911],
[0.18182497],
[0.18340451],
[0.30424224],
[0.52475643],
[0.43194502],
[0.29122914],
[0.61185289],
[0.13949386],
[0.29214465],
[0.36636184],
[0.45606998],
[0.78517596],
[0.19967378],
[0.51423444],
[0.59241457],
[0.04645041],
[0.60754485],
[0.17052412],
[0.06505159],
[0.94888554],
[0.96563203],
[0.80839735],
[0.30461377],
[0.09767211],
[0.68423303],
[0.44015249],
[0.12203823],
[0.49517691],
[0.03438852],
[0.9093204 ],
[0.25877998],
[0.66252228],
[0.31171108],
[0.52006802],
[0.54671028],
[0.18485446],
[0.96958463],
[0.77513282],
[0.93949894],
[0.89482735],
[0.59789998],
[0.92187424],
[0.0884925 ],
[0.19598286],
```

```
X = 2*np.random.rand(100, 1)
```

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




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

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

100 rows × 1 columns

```
from sklearn.linear_model import LinearRegression
```

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

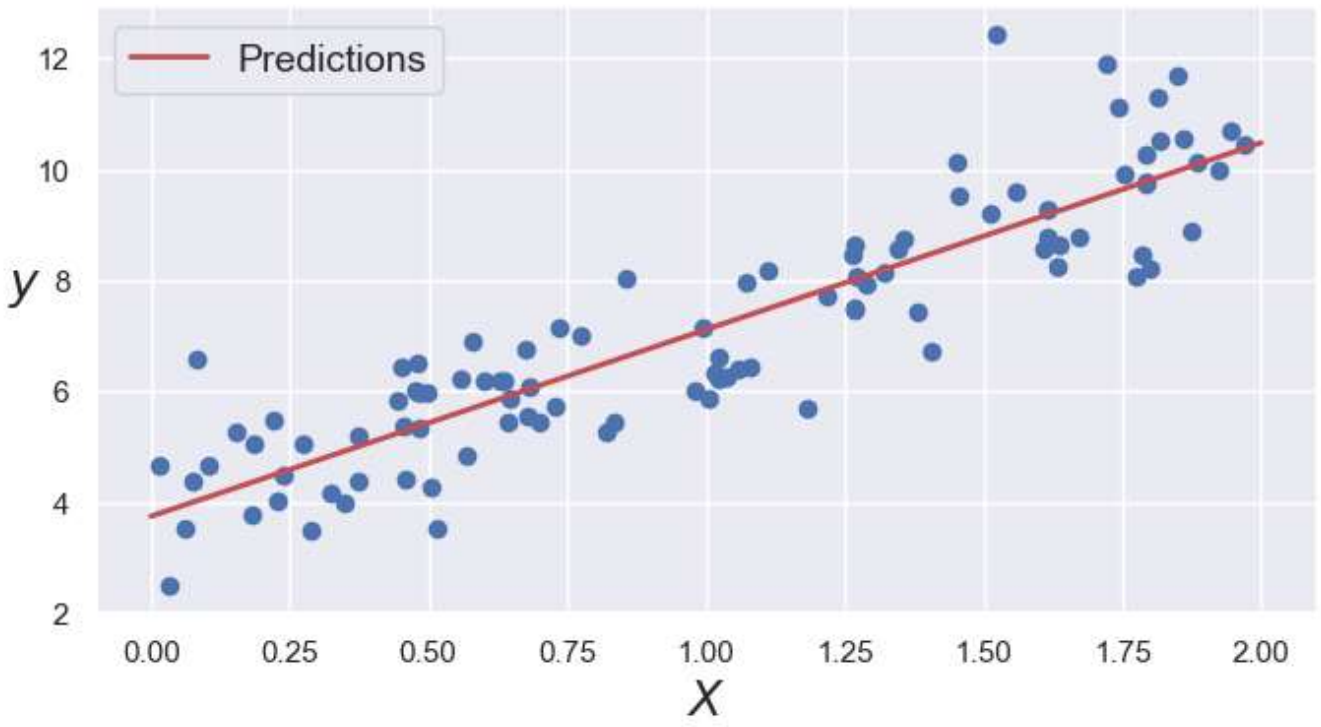
▾ LinearRegression

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:

$\hat{y} = 3.30X + 3.57$

```
#|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

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

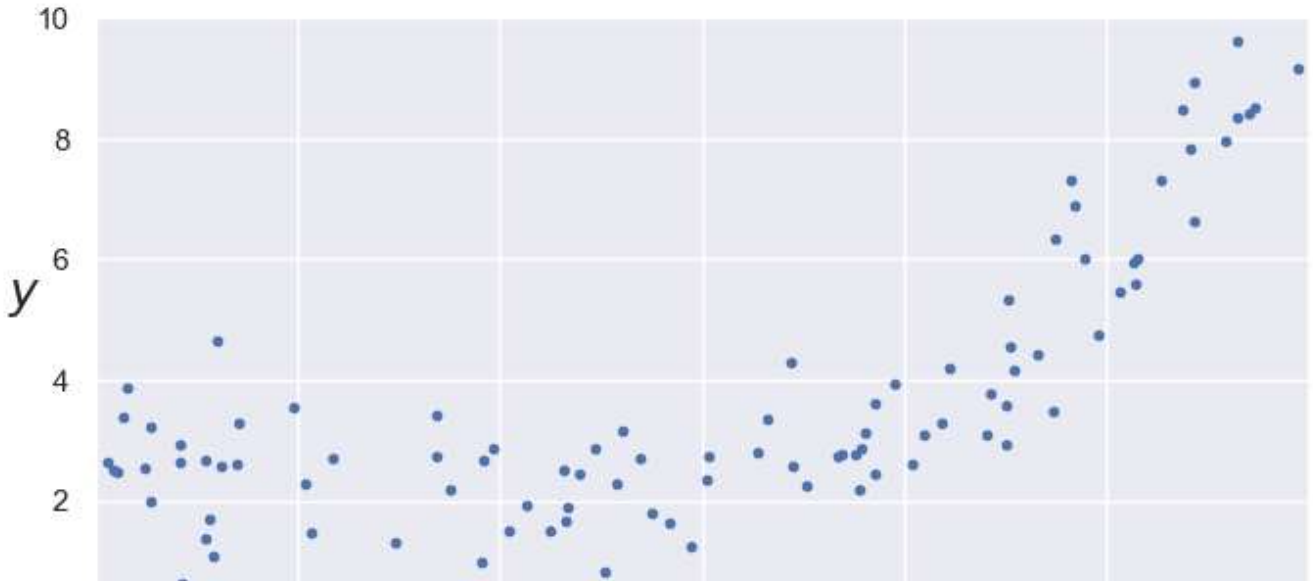
This is still a linear model, the linearity refers to the fact that the coefficients never multiply or divide each other.

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

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)
```

97	3.488785

```
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],
```

`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_
```

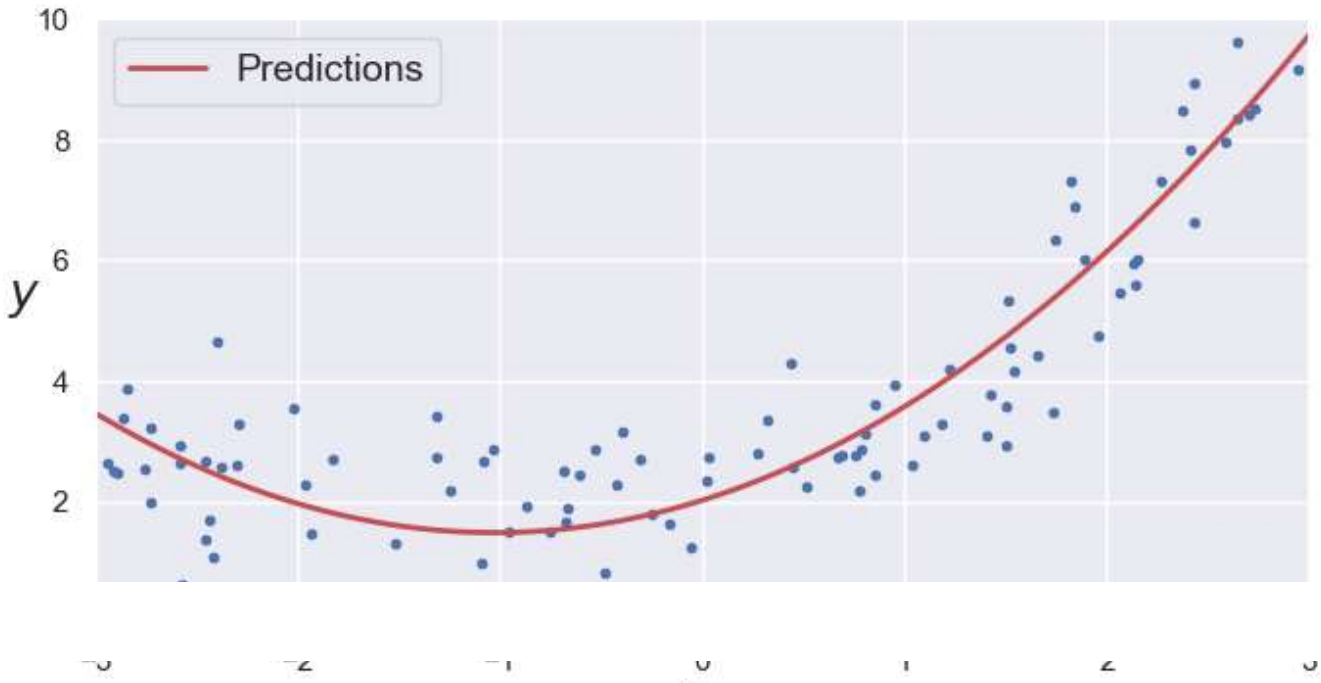
```
(array([[1.04271531, 0.50866711]]), array([2.01873554]))
```

The model estimates:

$$\hat{y} = 0.96X + 0.96X^2 + 2.19$$

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]);
```

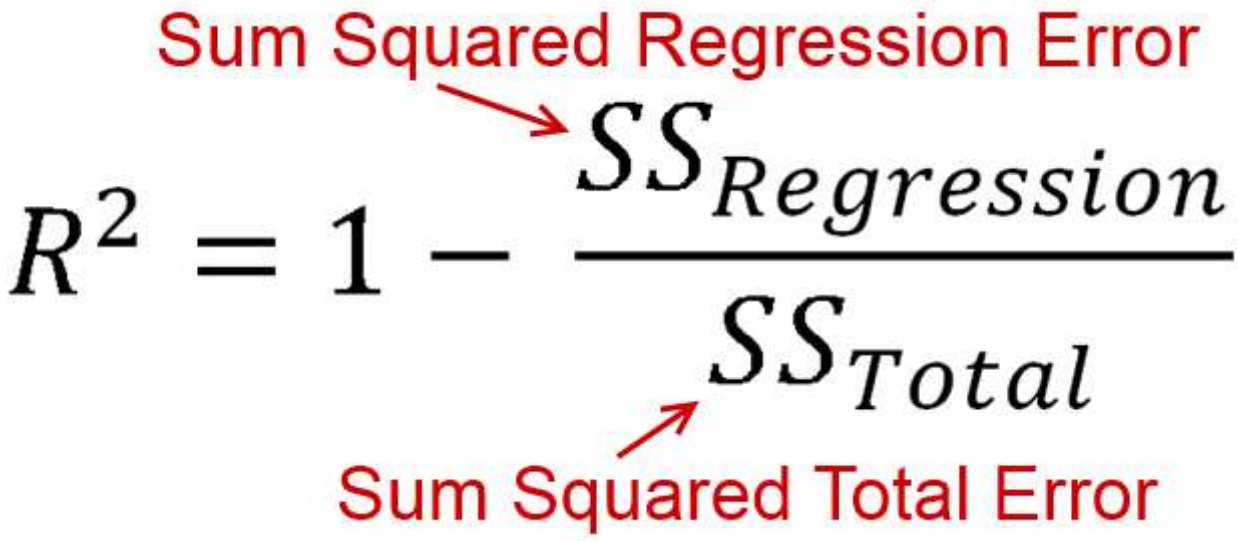


R square

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.

[ ] ↳ *93 celdas ocultas*

Ejercicio 2

Casas en King Country

```
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
2009	2472930270	20140905T000000	485000.0	3	2.50	3110	90
10935	2141500070	20140619T000000	450000.0	4	2.50	2400	76
21496	1283800110	20140506T000000	776000.0	4	2.50	3040	64
15805	3904901300	20150414T000000	468000.0	3	2.25	1470	55
9578	3204300625	20140903T000000	785950.0	4	3.00	2530	45
19889	1776460140	20140724T000000	395000.0	3	2.50	2130	50
5076	5100402764	20150415T000000	740000.0	3	1.00	1230	63
235	1762600320	20140610T000000	1025000.0	5	4.00	3760	280
3381	7854800090	20141107T000000	799950.0	3	3.00	2900	117
12893	1431600180	20150403T000000	335000.0	5	3.00	2660	77

```
df.info()
```

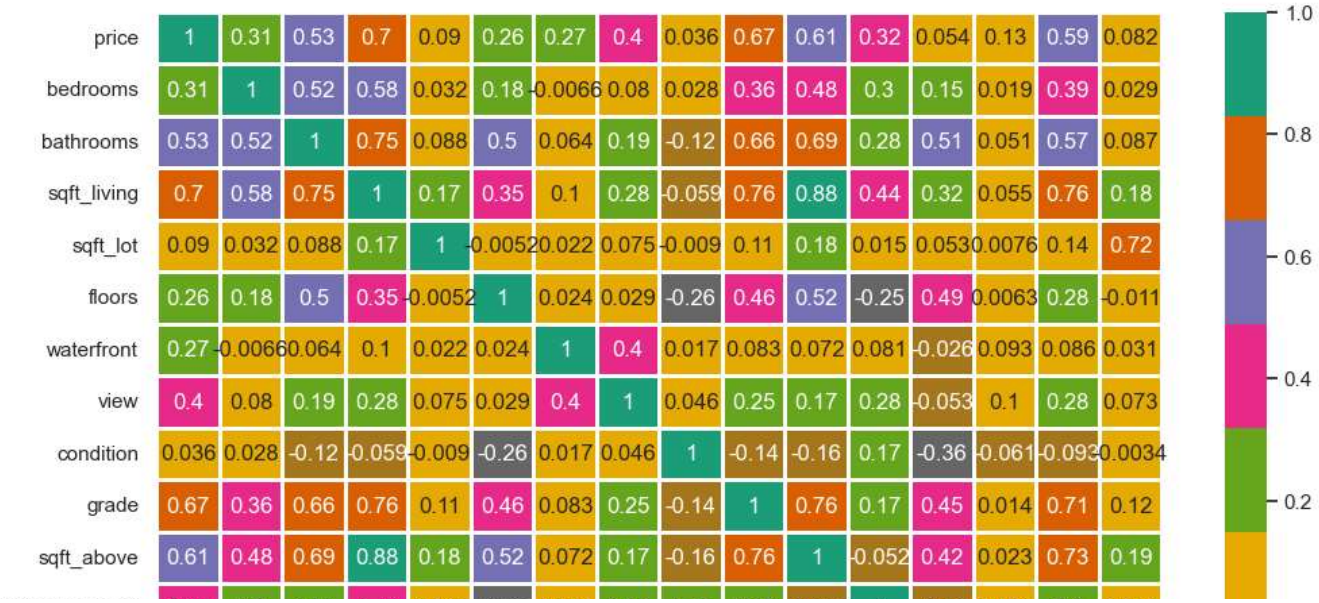
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          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))
sns.heatmap(df.corr(), annot=True, cmap='Dark2_r', linewidths = 2)
plt.show()
```

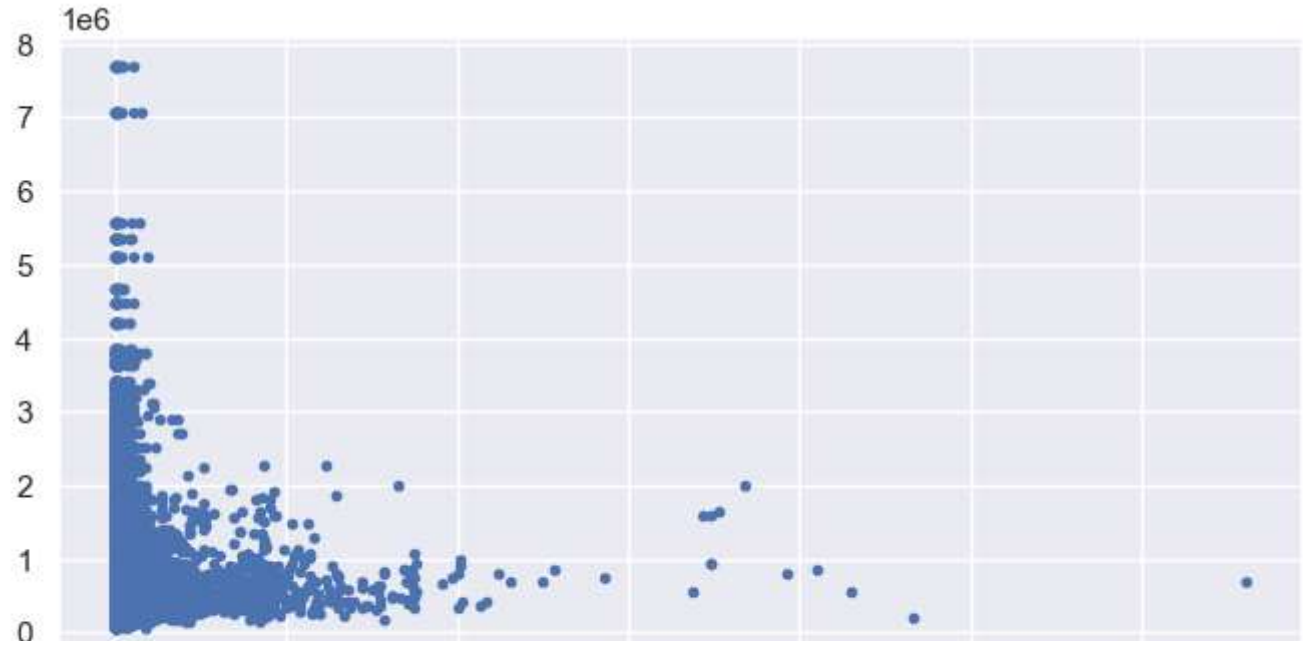



```
columns = df.columns.drop('price')

features = columns
label = ['price']

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

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

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

```
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)
```

```
plt.plot(X_train,y_train,'b.')
plt.show()
```




```
linear_reg = LinearRegression(fit_intercept=True)
linear_reg.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
linear_reg.coef_,
```

```
(array([[-3.82008048e+04,  4.14661380e+04,  1.07992584e+02,
          1.71356997e-02,  3.16916913e+04,  5.52691023e+05,
          4.12493228e+04,  2.12221443e+04,  1.19493216e+05,
          4.77750272e+01,  6.02175567e+01, -3.55090216e+03,
          1.32602215e+01,  2.90059284e+01, -5.48132603e-01]]),)
```

```
linear_reg.intercept_
```

```
array([6151359.26274133])
```

The model estimates:

calculate the ecuation is complicated due to the 15 parameter on the X

```
y_pred = linear_reg.predict(X_train)
from sklearn import metrics
from sklearn.metrics import r2_score
linear_Mae_train=metrics.mean_absolute_error(y_train,y_pred)
linear_r2_train=r2_score(y_train,y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_train, y_pred))
```

```
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
print('r2_score', r2_score(y_train, y_pred))
```

```
Error medio Absoluto (MAE): 139269.32939115047
Root Mean Squared Error: 214234.8822754647
```

```
y_pred = linear_reg.predict(X_test)
from sklearn import metrics
from sklearn.metrics import r2_score
linear_Mae=metrics.mean_absolute_error(y_test,y_pred)
linear_r2=r2_score(y_test,y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('r2_score', r2_score(y_test, y_pred))
```

```
Error medio Absoluto (MAE): 137480.13882731603
Root Mean Squared Error: 232133.36762408566
r2_score 0.6579723205007496
```

Polinomial

```
from sklearn.preprocessing import PolynomialFeatures

poly_transform = PolynomialFeatures(degree=3, include_bias = False)
polyX = poly_transform.fit_transform(X_train)
polyX
```

```
array([[5.00000000e+00, 2.75000000e+00, 3.75000000e+03, ...,
        7.06230720e+10, 1.64221278e+11, 3.81867106e+11],
       [4.00000000e+00, 4.50000000e+00, 5.25000000e+03, ...,
        3.02200000e+11, 5.84478976e+12, 1.13042910e+14],
       [3.00000000e+00, 2.50000000e+00, 2.88000000e+03, ...,
        9.21194624e+10, 3.27210820e+11, 1.16226168e+12],
       ...,
       [3.00000000e+00, 2.25000000e+00, 1.78000000e+03, ...,
        1.73857625e+10, 8.77923438e+10, 4.43322266e+11],
       [2.00000000e+00, 1.00000000e+00, 1.15000000e+03, ...,
        9.24639408e+09, 2.73030912e+10, 8.06215680e+10],
       [3.00000000e+00, 1.00000000e+00, 1.45000000e+03, ...,
        8.37158400e+09, 6.23039040e+10, 4.63684824e+11]])
```

```
poly_regression = LinearRegression(fit_intercept=True)
poly_regression.fit(polyX, y_train)
poly_regression.intercept_
```

```
array([22303152.33754259])
```

```
poly_regression.coef_
```

```
array([[-2.61095885e+02,  1.94799681e+01,  7.36856487e+01,
         3.44355280e-03, -5.53722248e+01, -1.19147437e+01,
        -1.33611969e+00,  3.99586651e-01,  9.01854617e-02,
         2.05705202e+00,  9.66689402e-01, -5.41798467e-02,
        -1.44814417e+00, -1.19376403e+00, -1.85682858e+01,
         1.40245749e+00, -1.92334700e-01, -1.15860409e+01,
        -6.07776034e+00,  1.75337027e-01,  4.60366656e-01,
         1.08006405e-01,  1.16516393e+00,  1.13251999e+00,
        -1.46933663e+01,  3.40151878e+00,  3.72667095e+02,
         3.97643611e+00, -1.40007329e+01, -2.50779267e+01,
        -2.71001582e-01, -1.28754891e+02, -5.90497228e+01,
        -4.46752199e-01, -6.42874131e-02, -6.94370006e-02,
        -3.77644274e-02, -3.22683502e-01, -1.03608388e+02,
        -2.51332572e+01, -1.06388577e+02, -2.50039286e+00,
        -1.03861608e+02,  8.64936649e+01, -3.72865019e-01,
         2.88501059e-02, -3.33200797e+02, -2.07333618e+01,
        -5.29072204e+01,  1.32423533e+02,  1.09155914e+02,
         2.43391925e-01, -6.16240594e-01,  6.51661078e-01,
        -2.85187069e-01,  1.12887463e+00, -1.60921618e-01,
         2.61114802e-05,  6.83838823e+01, -2.61762159e+01,
         1.37028413e+01,  2.37503096e+01,  1.37215658e+01,
         2.17234311e-02,  7.11838146e-03, -9.48383562e-02,
        -2.07244403e-01, -7.80321445e-02,  3.96794379e-04,
        -5.74458455e-01, -6.20566386e-02, -1.48102823e-01,
        -4.31534566e-01, -1.13898394e+00, -2.85452337e+02,
        -4.77483963e+01, -3.07142584e+02, -3.25894721e+00,
        -3.08216941e+02, -9.55655484e+01, -1.66048855e-02,
        -1.67775924e-01, -1.10289063e-01, -1.67119032e-01,
        -1.81992610e+01, -2.53408813e+00, -1.61576940e+01,
         1.11336231e-01, -1.97411899e+01, -1.13881776e+02,
        -1.25739482e-01, -1.17851571e-01, -1.52525494e-01,
        -4.09392847e+01, -1.19679841e+01, -4.95884729e+01,
        -4.40641268e+00, -5.97054817e+01, -3.02732485e+01,
         4.91217619e-01,  6.02823228e-01,  1.08994799e+02,
         2.34287414e+01,  1.29268000e+02, -2.01907919e+00,
         1.23223374e+02,  1.30823864e+01,  6.66832888e-01,
         8.68033783e+01,  2.23525260e+01,  8.27554974e+01,
         7.22223896e-01,  9.19756606e+01, -1.12010511e+01,
        -1.81688240e+00,  2.06027390e+00,  1.35167042e+00,
        -1.80012464e+00,  1.06768171e+00, -4.02725853e-03,
        -2.67650589e+00, -7.00026515e-01,  1.51497230e+00,
         6.12014935e-02, -1.56902518e-01, -1.90010081e+01,
         4.60100541e+01,  1.77398867e+00,  1.35998102e-01,
        -4.51871918e+01,  2.50698892e-01, -2.17008292e-02,
        -2.05260212e+00,  1.98390680e-01, -1.06332518e-03,
         1.08528458e+02, -1.97390319e+00,  5.89026342e+00,
         4.58464822e-01,  9.28325869e-01, -6.80504857e-01,
        -2.26933356e+00,  5.09389744e+00,  2.14931491e+00,
        -3.76538117e+00,  9.65564461e+00, -5.50379923e+00,
         7.96470345e+00, -5.26079571e+00,  5.03805383e-01,
        -8.47402141e-01, -4.83835384e+00,  3.31955159e-01,
        -1.01645000e+00, -1.43482645e-01,  2.68086907e-01,
        -1.13256458e+00,  3.08273086e-01, -1.63814234e+01,
         1.15430695e+01, -3.88960385e+01, -3.56451563e+01,
         6.75912274e+01,  8.97922892e-01,  5.56879460e-03,
        -8.01025986e-04, -2.28004132e+01,  3.18070750e+01,
```

```
7.59340907e+00,  3.24210410e-01, -7.12998416e-02,
```

Making the ecuation is complicated due to the amount of x components

```
polyX = poly_transform.fit_transform(X_train)
polyX
y_pred = poly_regression.predict(polyX)
from sklearn import metrics
from sklearn.metrics import r2_score
Poly_Mae_train=metrics.mean_absolute_error(y_train,y_pred)
Poly_r2_train=r2_score(y_train,y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_train, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
print('r2_score',r2_score(y_train, y_pred ))
```

```
Error medio Absoluto (MAE): 141530.9591213192
Root Mean Squared Error: 213038.105726498
r2_score 0.6567866116524821
```

```
polyX = poly_transform.fit_transform(X_test)
polyX
y_pred = poly_regression.predict(polyX)
```

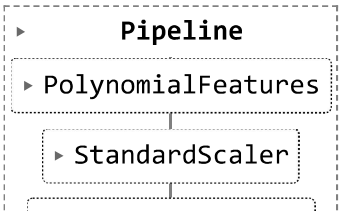
```
from sklearn import metrics
from sklearn.metrics import r2_score
Poly_Mae=metrics.mean_absolute_error(y_test,y_pred)
Poly_r2=r2_score(y_test,y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('r2_score',r2_score(y_test, y_pred ))
```

```
Error medio Absoluto (MAE): 149110.84568448795
Root Mean Squared Error: 267502.30518914235
r2_score 0.5458062351742023
```

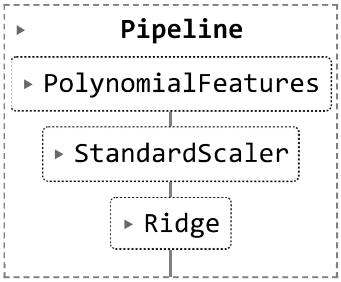
Ridge

```
X_train2=X_train.to_numpy().reshape(-1,1)
```

```
model = Pipeline([("poly_features", PolynomialFeatures(degree=2, include_bias=False)),
                  ("scaler", StandardScaler()),
                  ("linear_reg", LinearRegression())])
model.fit(X_train, y_train)
```




```
from sklearn.linear_model import Ridge

model_ridge = Pipeline([("poly_features", PolynomialFeatures(degree=5, include_bias=False)),
                  ("scaler", StandardScaler()),
                  ("ridge", Ridge(alpha = 5, solver = 'cholesky', random_state = 42))])
model_ridge.fit(X_train, y_train)
```




```
y_pred = model.predict(X_train)
from sklearn import metrics
from sklearn.metrics import r2_score

Ridge_Mae_train = metrics.mean_absolute_error(y_train, y_pred)
Ridge_r2_train = r2_score(y_train, y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_train, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
print('r2_score', r2_score(y_train, y_pred))
```

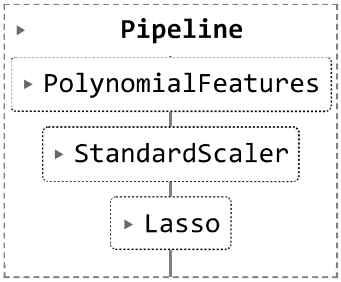
```
Error medio Absoluto (MAE): 123196.96131401205
Root Mean Squared Error: 182926.38521672378
r2_score 0.7469523304447205
```

```
y_pred = model.predict(X_test)


from sklearn import metrics
from sklearn.metrics import r2_score
Ridge_Mae=metrics.mean_absolute_error(y_test,y_pred)
Ridge_r2=r2_score(y_test,y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('r2_score',r2_score(y_test, y_pred ))
```

```
Error medio Absoluto (MAE): 121317.06614912253
Root Mean Squared Error: 186260.82904512723
r2_score 0.7797939871361215
```

Lasso

```
from sklearn.linear_model import Lasso

model_lasso = Pipeline([("poly_features", PolynomialFeatures(degree=5, include_bias=False)),
                        ("scaler", StandardScaler()),
                        ("lasso", Lasso(alpha = 1, random_state = 42,tol=0.1,max_iter=1000))]


model_lasso.fit(X_train, y_train)
```




```
y_pred = model_lasso.predict(X_train)
from sklearn import metrics
from sklearn.metrics import r2_score

Lasso_MAE_train = metrics.mean_absolute_error(y_train, y_pred)
Lasso_r2_train= r2_score(y_train, y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_train, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_train, y_pred)))
print('r2_score', r2_score(y_train, y_pred))
```

```
Error medio Absoluto (MAE): 113884.60002386104
Root Mean Squared Error: 162623.20298801802
r2_score 0.8000070629856828
```

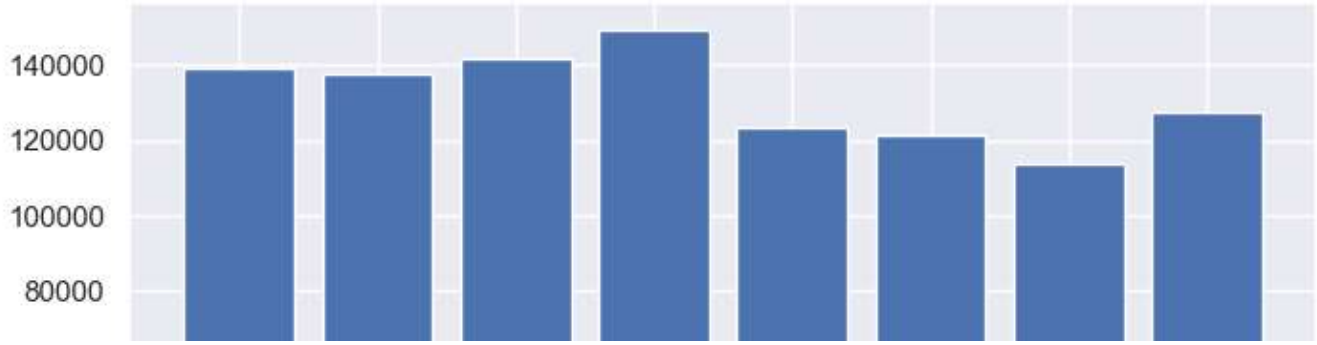
```
y_pred = model_lasso.predict(X_test)


from sklearn import metrics
from sklearn.metrics import r2_score
Lasso_MAE=metrics.mean_absolute_error(y_test,y_pred)
Lasso_r2=r2_score(y_test,y_pred)
print('Error medio Absoluto (MAE):', metrics.mean_absolute_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('r2_score',r2_score(y_test, y_pred ))
```

```
Error medio Absoluto (MAE): 127385.8598537924
```

```
Root Mean Squared Error: 254950.23977556525
r2_score 0.5874306511507237
```

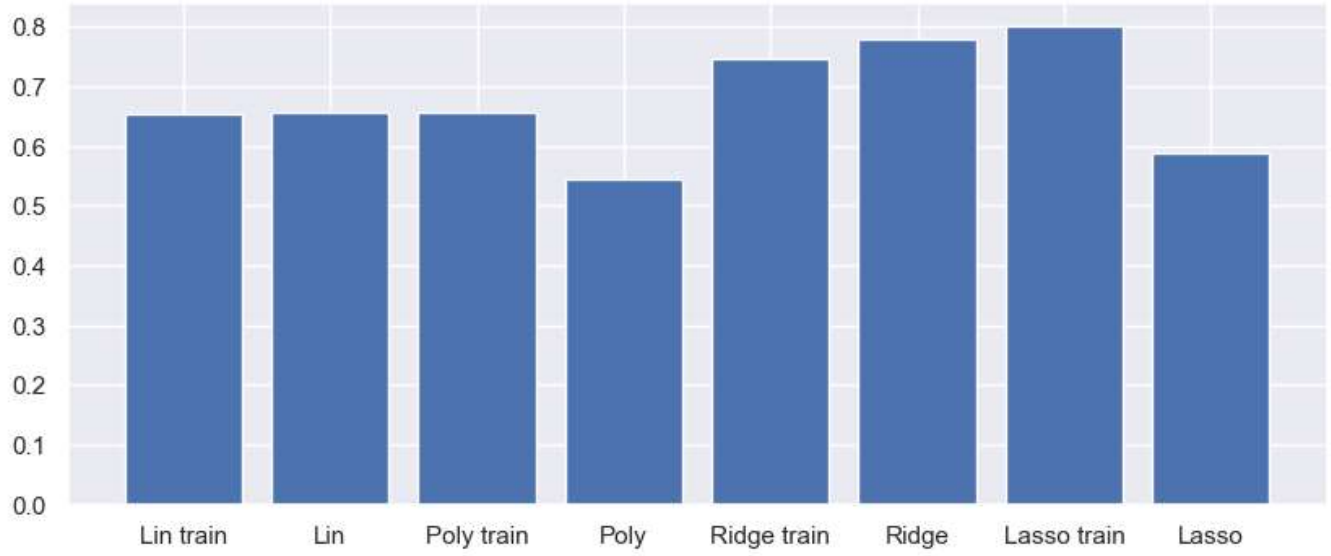
```
def diagramas(metricas):
    sumt=list()
    diagrama=metricas
    sumt.append(diagrama)
    return sumt


Metricas = list()
Metricas.append('Lin train')
Metricas.append('Lin')
Metricas.append('Poly train')
Metricas.append('Poly')
Metricas.append('Ridge train')
Metricas.append('Ridge')
Metricas.append('Lasso train')
Metricas.append('Lasso')


MAE = list()
MAE.append(linear_Mae_train)
MAE.append(linear_Mae)
MAE.append(Poly_Mae_train)
MAE.append(Poly_Mae)
MAE.append(Ridge_Mae_train)
MAE.append(Ridge_Mae)
MAE.append(Lasso_MAE_train)
MAE.append(Lasso_MAE)


fig, ax = plt.subplots(figsize =(8, 4))
print("MAE")
print(MAE)
Metricas=list(Metricas)
MAE=list(MAE)
plt.bar(Metricas,MAE)


plt.show()
```

```
MAE
[139269.32939115047, 137480.13882731603, 141530.9591213192, 149110.84568448795, 123196.9
```




```
R2 = list()
R2.append(linear_r2_train)
R2.append(linear_r2)
R2.append(Poly_r2_train)
R2.append(Poly_r2)
R2.append(Ridge_r2_train)
R2.append(Ridge_r2)
R2.append(Lasso_r2_train)
R2.append(Lasso_r2)
fig, ax = plt.subplots(figsize=(10, 4))
print("r2")
print(R2)
R2 = list(R2)
plt.bar(Metricas, R2)

plt.show()
```

```
r2
[0.6529196653133833, 0.6579723205007496, 0.6567866116524821, 0.5458062351742023, 0.74695
```



En este sistema podemos ver que el entrenamiento y el training si estan funcioando para el caso ridge donde el training funcioana ya que el test nodarroja una R2 de casi 80% por eso mismo yo me quedaria con el ridge m aunque cuando vemos el Lasso tiene un trainig de 80 al avelauar ya con los datos de tes vemos que el modelo esta subentrenado