Maestría en Inteligencia Artificial Aplicada

Curso: Ciencia y analítica de datos

Tecnológico de Monterrey

Prof. Titular: María de la Paz Rico Fernández

Prof. Tutor: Bernardo Charles Canales

Actividad Semanal -- 7 Regresiones y K meansn

Alumno: Alberto Nieves Cisneros
Matrícula: A01793829 **Fecha**: 09/11/2022

Linear Models

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

In [1]:
```python
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)
```

In [2]:
```python
5-2
```

Out[2]: 3

Simple Linear Regression

Simple linear regression equation:

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

Generate linear-looking data with the equation:

$$y = 3X + 4 + noise$$

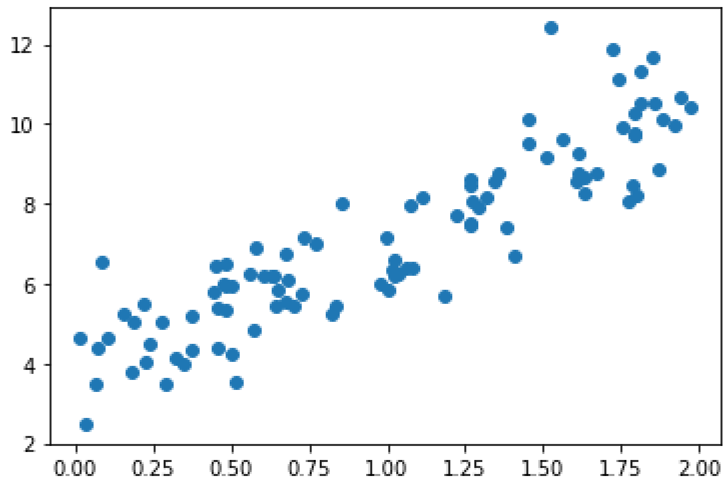
In [3]:
```python
np.random.rand(100, 1)
```

Out[3]:
```
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]])
```

In [4]:

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



In [5]:

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

Out[5]:

	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

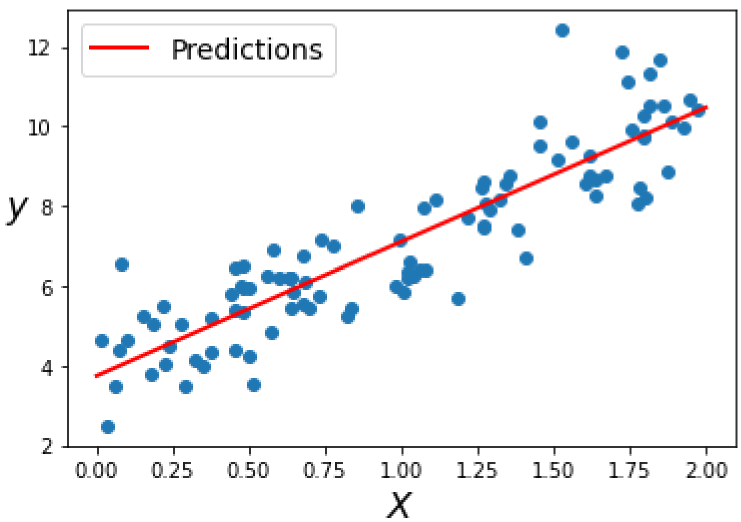
100 rows × 1 columns

In [6]:
```python
from sklearn.linear_model import LinearRegression

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

Out[6]:
```
▾ LinearRegression

LinearRegression()
```

Plot the model's predictions:

In [7]:
```python
# 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:

In [8]:
```python
X_new = np.array([[0], [2]])
X_new
```

Out[8]:
```
array([[0],
       [2]])
```

In [9]:
```python
X_new.shape
```

Out[9]:
```
(2, 1)
```

In [10]:
```
y_new = linear_reg.predict(X_new)
y_new
```

Out[10]:
```
array([[ 3.74406122],
       [10.47517611]])
```

In [11]:
```
 linear_reg.coef_, linear_reg.intercept_
```

Out[11]:
```
(array([[3.36555744]]), array([3.74406122]))
```

The model estimates:

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

In [12]:
```
#|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.

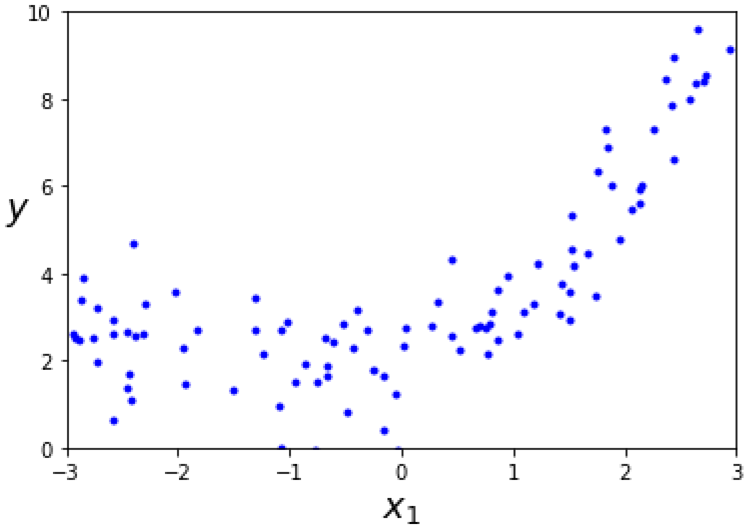
To generate polynomial data we use the function:

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

In [13]:
```
# 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)
```

In [14]:

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



In [15]:

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

Out[15]:

	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 × 1 columns

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

In [16]:
```
from sklearn.preprocessing import PolynomialFeatures

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

In [17]:
```
X_poly
```

Out[17]:
```
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:

In [18]:

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

```

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

In [19]:
```
X_poly[0]
```

Out[19]:
```
array([2.72919168, 7.44848725])
```

Fit the model to this extended training data:

In [20]:
```
lin_reg = LinearRegression(fit_intercept=True)
lin_reg.fit(X_poly, y)
lin_reg.coef_, lin_reg.intercept_
```

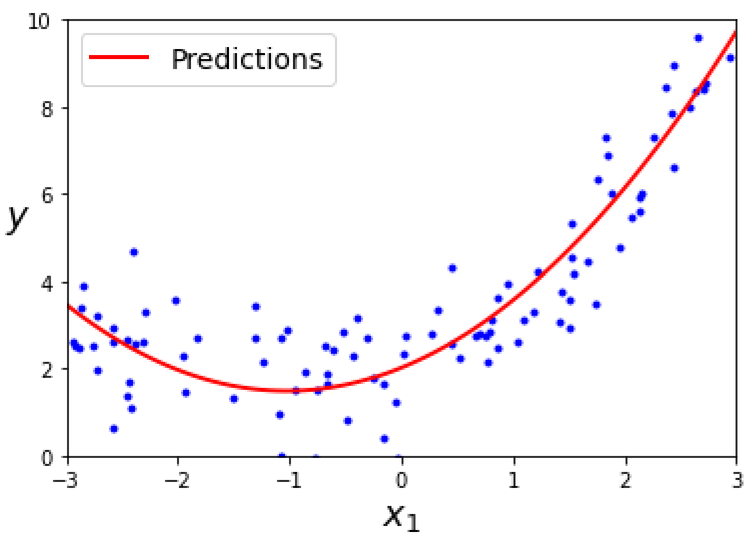
Out[20]:
```
(array([[1.04271531, 0.50866711]]), array([2.01873554]))
```

The model estimates:

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

Plot the data and the predictions:

In [21]:
```
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]);
```



In [ ]:
```

```

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.

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

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

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

Ejercicio 1

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

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

Out[22]:

	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

In [23]:
```
X = df[['Number of Units']]
y = df['Manufacturing Cost']
```

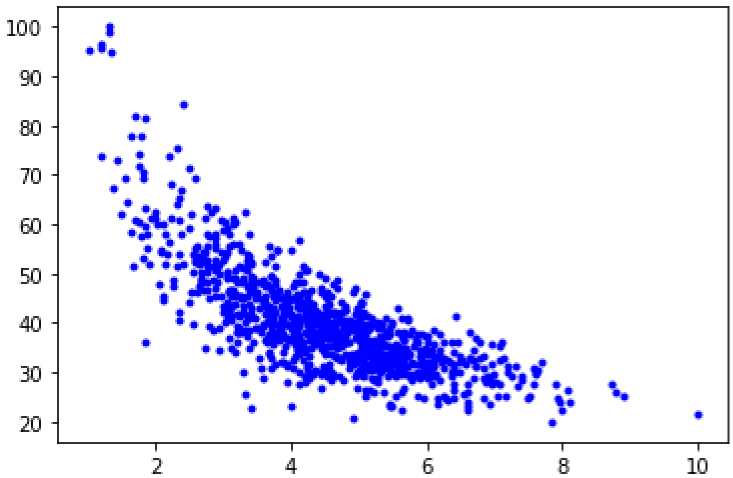
In [24]:
```
len(X)
```

Out[24]:
```
1000
```

In [25]:
```
y.describe
```

Out[25]:
```
<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>
```

In [26]:
```
plt.plot(X,y,'b.')
```
Out[26]: [<matplotlib.lines.Line2D at 0x16064713190>]



In [27]:
```
#Dividimos los datos
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r

print(f'Numero total de registros en la base de datos: {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 base de datos: 1000
**************************************************
Numero total de registros en el training set: 800
Tamaño de X_train: (800, 1)
**************************************************
Mumero total de registros en el test dataset: 200
Tamaño del X_test: (200, 1)
```

In [28]:
```
#Modelo lineal
from sklearn.linear_model import LinearRegression                #Llamamos la

linear_reg = LinearRegression(fit_intercept=True)                #Definimos e
linear_reg.fit(X_train, y_train)                                 #Ajustamos e
```
Out[28]:
```
▾ LinearRegression
LinearRegression()
```

In [29]:

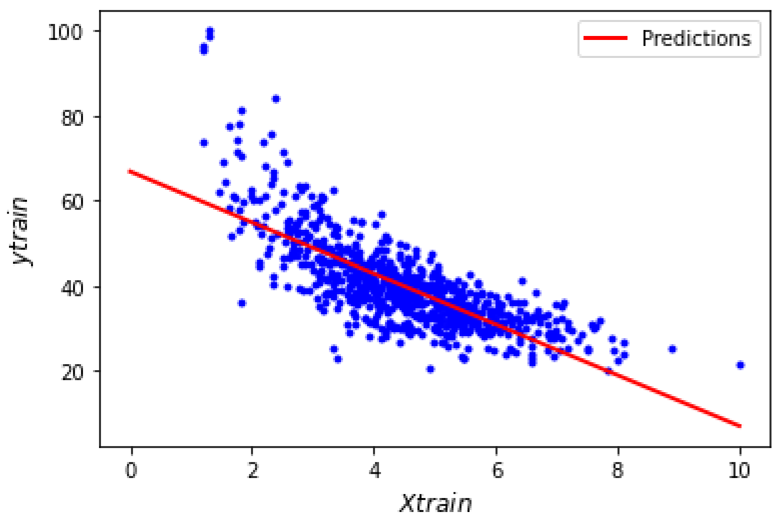
```
linear_reg.coef_, linear_reg.intercept_          #Recuperamos el coeficie
print(f"La ecuación del modelo es: {linear_reg.intercept_:.2f} + {np.array2s
```

```
La ecuación del modelo es: 66.80 + -5.9791x
```

In [30]:

```
#gráfico de Regresión Lineal
X_fit = np.linspace(0, 10, 200)
y_fit = linear_reg.predict(X_fit[:, np.newaxis])

plt.plot(X_train, y_train, "b.")
plt.plot(X_fit, y_fit, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X train$", fontsize=12)
plt.ylabel("$y train$", rotation=90, fontsize=12)
plt.legend(loc="upper right", fontsize=10);
```

```
C:\Users\sergi\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```



In [31]:

```
#Calculamos los errores en base a conjunto de prueba
y_pred = linear_reg.predict(X_test)
lin_errors = np.abs(y_test - y_pred)
lin_errors
```

Out[31]:

```
545     0.323742
298     0.552918
109     8.282181
837     2.264868
194     3.980632
          ...
68      2.271456
449    12.299120
715     2.493250
793     1.175750
688     0.358293
Name: Manufacturing Cost, Length: 200, dtype: float64
```

In [32]:
```python
#Calculamos la R2, la cual podemos obtener del objeto del regresor
lin_r2 = linear_reg.score(X_test,y_test)
print(f"La R2 lineal es: {lin_r2:.4f}")
```

```
La R2 lineal es: 0.5958
```

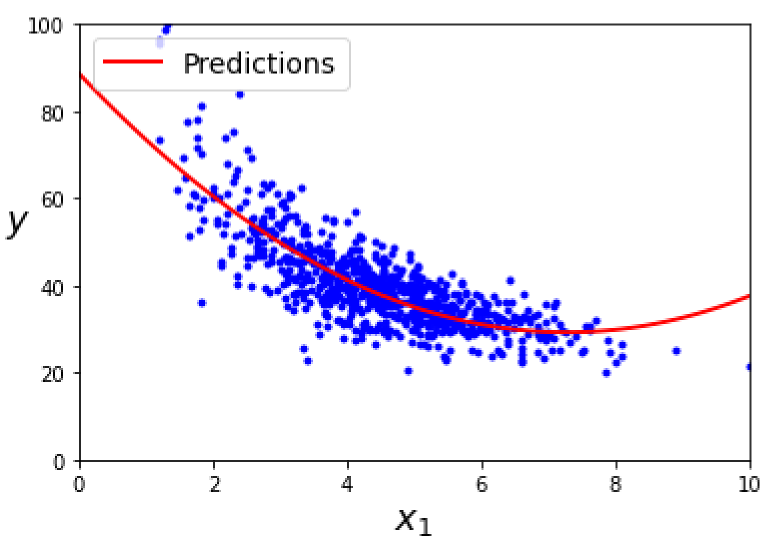
In [33]:
```python
#polinomial - añadimos el cuadrado de la variable existente
from sklearn.preprocessing import PolynomialFeatures

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

In [34]:
```python
poli_reg = LinearRegression(fit_intercept=True)
poli_reg.fit(X_poly, y_train)
print(f"La ecuación del modelo es: {poli_reg.intercept_:.4f} + {np.array2str
```

```
La ecuación del modelo es: 88.6610 + -16.3251x + 1.1219x^2
```

In [35]:
```python
#regresión polinomica
X_new=np.linspace(0, 10, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = poli_reg.predict(X_new_poly)
plt.plot(X_train, y_train, "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([0, 10, 0, 100]);
```

```
C:\Users\sergi\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X
does not have valid feature names, but PolynomialFeatures was fitted with featu
re names
  warnings.warn(
```



In [36]:

```
#Calculamos los errores en base a conjunto de prueba
X_test_poly = poly_features.transform(X_test)
y_pred = poli_reg.predict(X_test_poly)
poli_errors = np.abs(y_test - y_pred)
poli_errors
```

Out[36]:

```
545     2.313779
298     1.749621
109     6.859096
837     2.961386
194     4.027661
          ...
68      0.547244
449    10.433620
715     0.903534
793     0.101250
688     1.363639
Name: Manufacturing Cost, Length: 200, dtype: float64
```

In [37]:

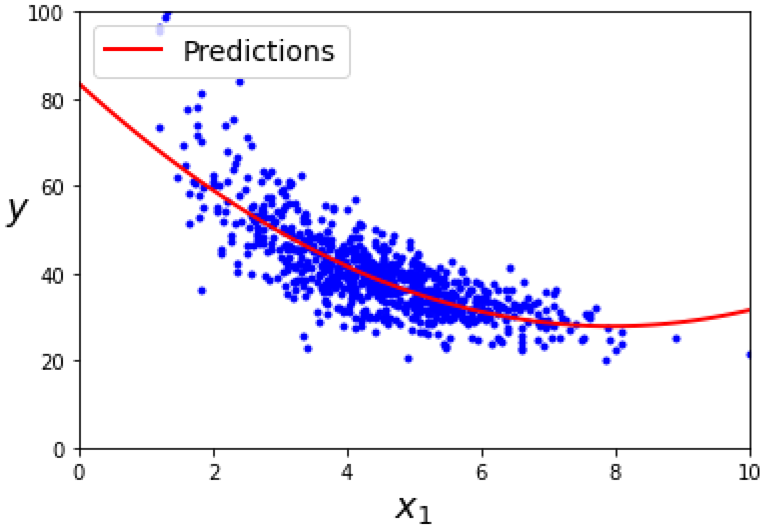
```
#Calculamos la R2, la cual podemos obtener del objeto del regresor
poly_r2 = poli_reg.score(X_test_poly,y_test)
print(f"La R2 polinomial es: {poly_r2:.4f}")
```

```
La R2 polinomial es: 0.7120
```

In [38]:

```python
#Regresión con Ridge
from sklearn.linear_model import Ridge

ridge_model = Ridge(alpha=10.0)
ridge_model.fit(X_poly, y_train)

print(f"La ecuación del modelo es: {ridge_model.intercept_:.4f} + {np.array2

#Visualización Ridge
y_new = ridge_model.predict(X_new_poly)
plt.plot(X_train, y_train, "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([0, 10, 0, 100])
plt.show;

y_pred = ridge_model.predict(X_test_poly)
ridge_errors = np.abs(y_test - y_pred)
print(f"Los errores son: \n{ridge_errors}")

ridge_r2 = ridge_model.score(X_test_poly,y_test)
print(f"La R2 Ridge es: {ridge_r2:.4f}")
```

```
La ecuación del modelo es: 83.5021 + -13.9949x + 0.88x^2
Los errores son:
545     1.872953
298     1.561755
109     7.326049
837     2.690493
194     4.134997
          ...
68      0.253093
449    10.816243
715     1.319525
793     0.436652
688     0.929950
Name: Manufacturing Cost, Length: 200, dtype: float64
La R2 Ridge es: 0.7005
```

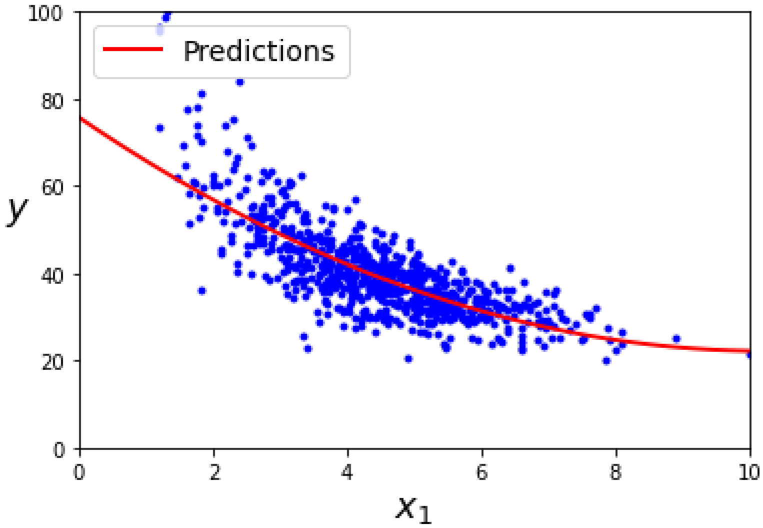


Predictions
100
80
60
40
20
0
y
0
2
4
6
8
10
x_1

In [39]:

```
#Regresion con Lasso
from sklearn.linear_model import Lasso

lasso_model = Lasso(alpha=0.4)
lasso_model.fit(X_poly, y_train)

print(f"La ecuación del modelo es: {lasso_model.intercept_:.4f} {np.array2st

#Visualización lasso
y_new = lasso_model.predict(X_new_poly)
plt.plot(X_train, y_train, "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([0, 10, 0, 100])
plt.show;

y_pred = lasso_model.predict(X_test_poly)
lasso_errors = np.abs(y_test - y_pred)
print(f"Los errores son: \n{lasso_errors}")

lasso_r2 = lasso_model.score(X_test_poly,y_test)
print(f"La R2 lasso es: {lasso_r2:.4f}")

```

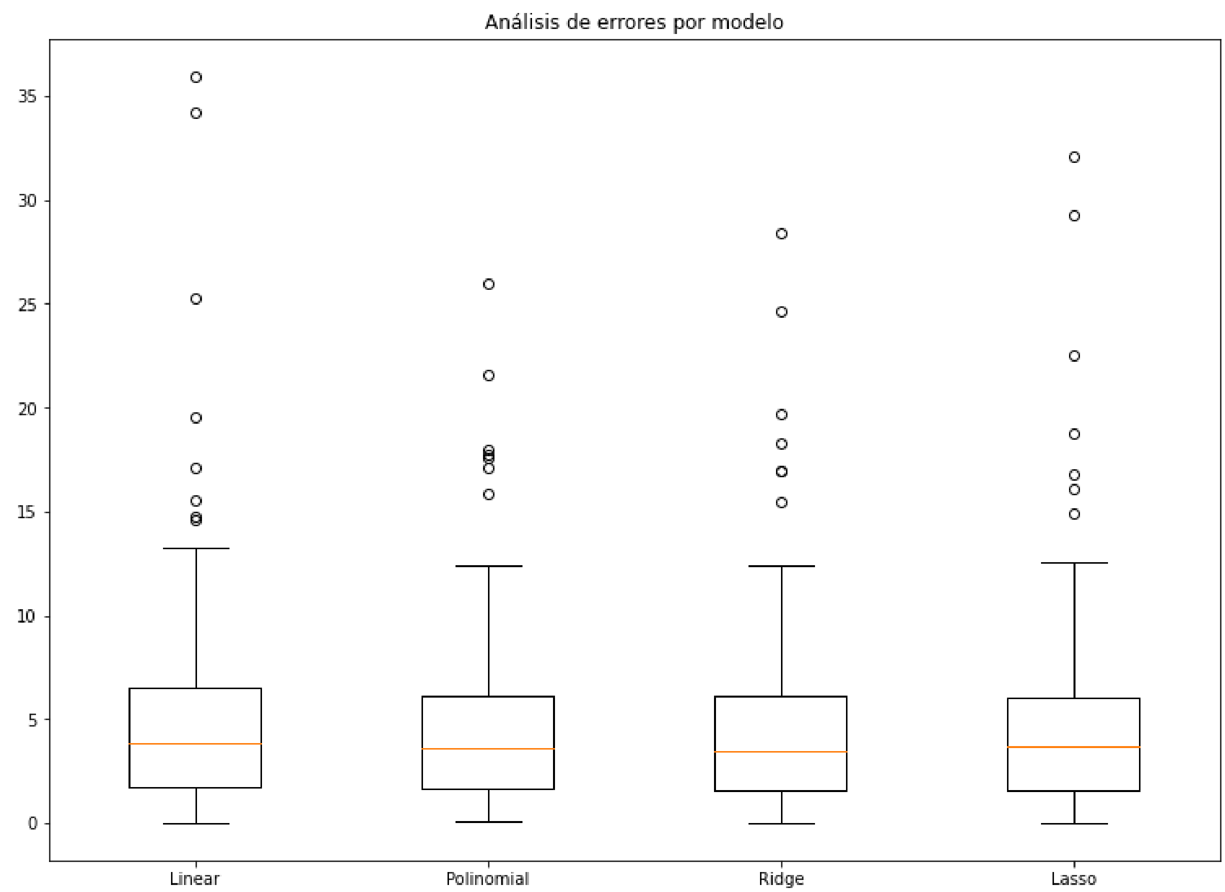
```
La ecuación del modelo es: 75.7262 -10.4592x + 0.5106x^2
Los errores son: 
545     1.202434
298     1.258660
109     8.002241
837     2.304635
194     4.271820
          ...   
68      1.430936
449    11.405107
715     1.937932
793     0.924949
688     0.282020
Name: Manufacturing Cost, Length: 200, dtype: float64
La R2 lasso es: 0.6672
```


Predictions
y
x1
0
20
40
60
80
100
0
2
4
6
8
10

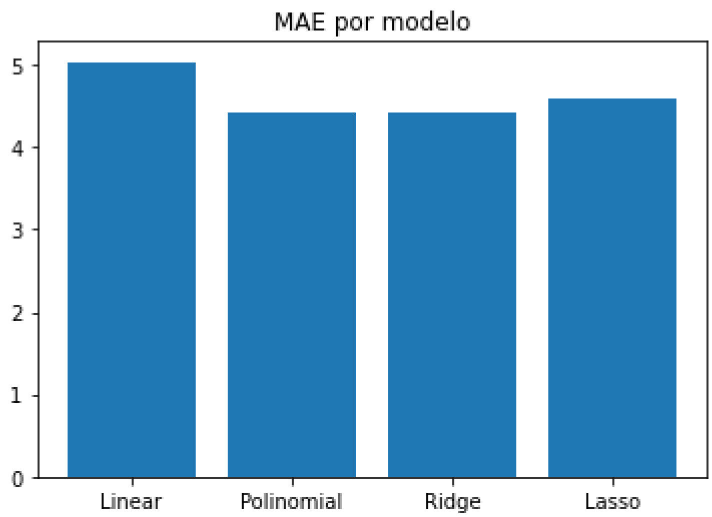
In [40]:

```python
#Grafica MAE de los 4 modelos

fig = plt.figure(figsize =(10, 7))

ax = fig.add_axes([0, 0, 1, 1])
ax.set_xticklabels(['Linear', 'Polinomial',
                    'Ridge', 'Lasso'])
plt.title("Análisis de errores por modelo")
bp = ax.boxplot([lin_errors, poli_errors, ridge_errors, lasso_errors])
plt.show()
```

```
C:\Users\sergi\AppData\Local\Temp/ipykernel_21344/813681842.py:6: UserWarning:
FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(['Linear', 'Polinomial',
```



In [41]:
```
#Graficas de MAE
plt.bar(['Linear', 'Polinomial','Ridge', 'Lasso'],
    [lin_errors.mean(), poli_errors.mean(), ridge_errors.mean(), lasso_error
    )
plt.title("MAE por modelo")
plt.show()
```

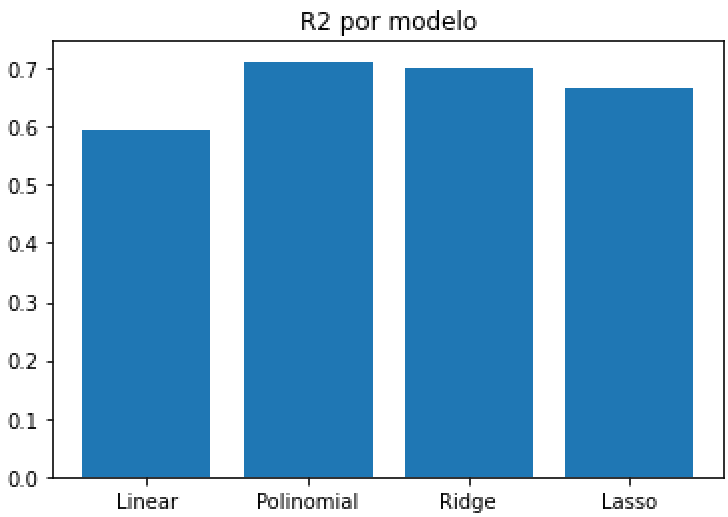


In [42]:
```
[lin_errors.mean(), poli_errors.mean(), ridge_errors.mean(), lasso_errors.me
```

Out[42]:
```
[5.03340366716028, 4.410633826931782, 4.415862873856254, 4.596362544583705]
```

In [43]:

```python
#Graficas de R2
plt.bar(['Linear', 'Polinomial','Ridge', 'Lasso'],
    [lin_r2, poly_r2, ridge_r2, lasso_r2],
    )
plt.title("R2 por modelo")
plt.show()
```



Conclusiones Ejercicio 1

Explica tus resultados, ¿que porcentajes de entrenamiento y evaluación?

Se utilizaron particiones del 80% entrenamiento y 20% pruebas.

Qué método conviene más a la empresa, ¿por que?, ¿que error tienes?, ¿es bueno?, ¿cómo lo sabes?

se concluye que el modelo que más conviene a la empresa es el modelo de generado por la Regresión polinómica, esto ya que de los 4 fue el que mantuvo un menor error (MAE = 4.41) y mayor R2 (71.20%) tras evaluar en el conjunto de pruebas.

Ejercicio 2

Realiza la regresión polinomial de los siguientes datos:

In [44]:
```
df = pd.read_csv('https://raw.githubusercontent.com/marypazrf/bdd/main/kc_ho
df.sample(10)
```

Out[44]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
5954	7852020250	20140602T000000	725995.0	4	2.50	3190	7869	2.0
8610	6392002020	20150324T000000	559000.0	3	1.75	1700	6500	1.0
7650	626049058	20150504T000000	275000.0	5	2.50	2570	17234	1.0
5683	2202500255	20150305T000000	335000.0	3	2.00	1210	9926	1.0
20773	7304301231	20140617T000000	345000.0	3	2.50	1680	2229	2.0
6959	723000114	20140505T000000	1395000.0	5	3.50	4010	8510	2.0
10784	4104900340	20150204T000000	710000.0	4	2.50	3220	18618	2.0
21529	2487200490	20140623T000000	670000.0	3	2.50	3310	5300	2.0
12319	2386000070	20141029T000000	795127.0	4	3.25	4360	91158	1.0
19948	293070090	20140711T000000	859990.0	4	2.75	3520	5500	2.0

10 rows × 21 columns

In [45]:

```python
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
```

In [46]:

```python
df.describe()
```

Out[46]:

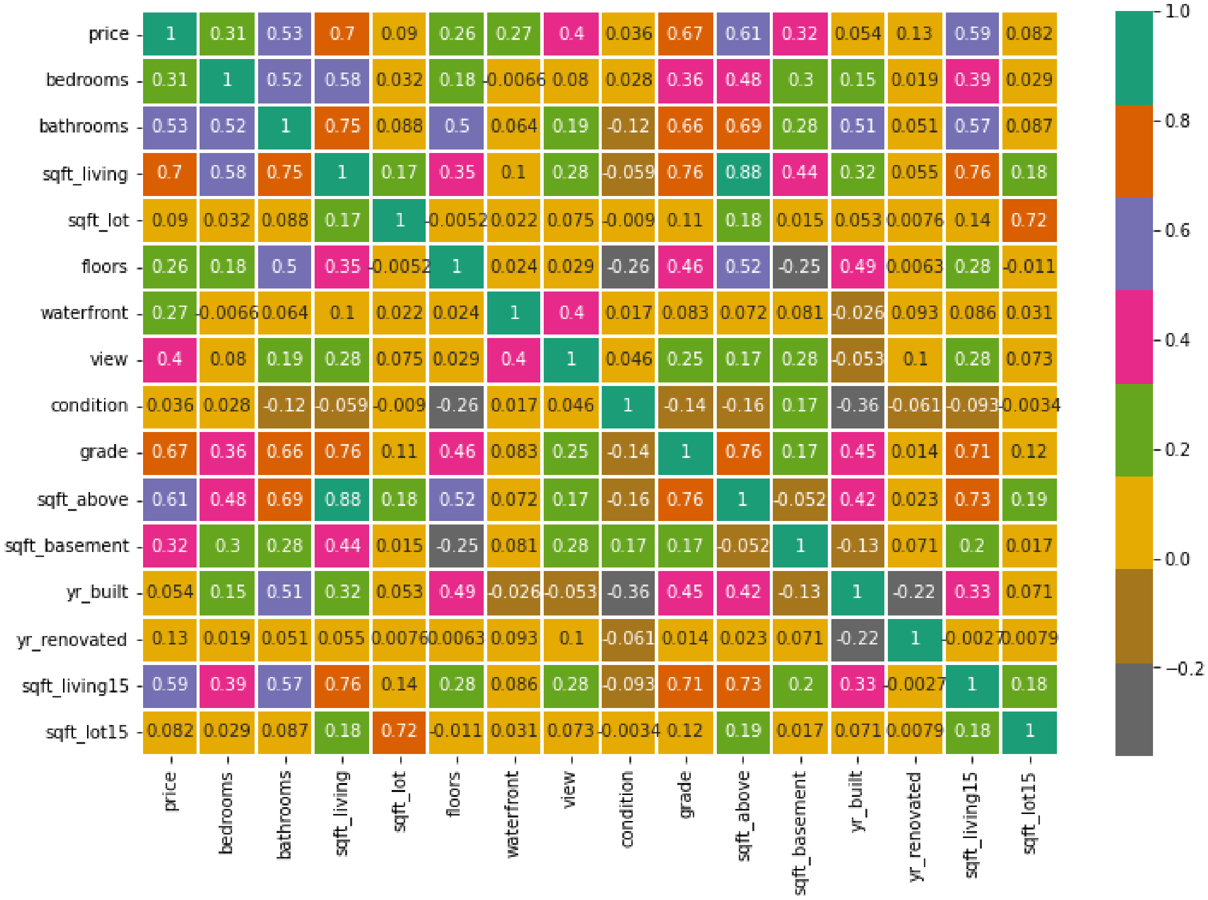
	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3

In [47]:

```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)

```

In [48]:

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



In [49]:

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

features = columns
label = ['price']

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

In [50]:

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r

print(f'Numero total de registros en la base de datos: {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 base de datos: 21613
**************************************************
Numero total de registros en el training set: 17290
Tamaño de X_train: (17290, 15)
**************************************************
Mumero total de registros en el test dataset: 4323
Tamaño del X_test: (4323, 15)
```

In [51]:

```python
#Modelo Lineal
modelo_lin = LinearRegression()
modelo_lin.fit(X_train, y_train)
print(f"Los coeficientes de la ecuación son: {modelo_lin.coef_.tolist()} + {

y_pred_lin = modelo_lin.predict(X_test)
lin_errors = np.abs(y_test - y_pred_lin)
print(f"Los errores son: \n{lin_errors}")

lin_r2 = modelo_lin.score(X_test,y_test)
print(f"La R2 lin es: {lin_r2:.4f}")

```

```
Los coeficientes de la ecuación son: [[-39523.70751886475, 46199.71152316872, 1
14.65594510211683, -0.00723070940843862, 25627.672981355754, 585132.9575807431,
42313.815765014624, 19331.112633746623, 118699.28766157702, 54.8447218114229, 5
9.81122313511115, -3560.2261245000323, 12.174341027486264, 18.412879002095693,
-0.529946236186367]] + [6180810.615871318]
Los errores son: 
              price
15783  5.155319e+04
14209  1.789348e+05
13532  1.036993e+05
10846  2.145816e+05
15952  3.753463e+04
...             ...
11362  1.990291e+05
1159   3.387871e+04
12939  3.748589e+05
19484  1.205070e+06
2138   5.165555e+04

[4323 rows x 1 columns]
La R2 lin es: 0.6587
```

In [52]:

```python
#Modelo polinomial
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly_train = poly_features.fit_transform(X_train)

modelo_poli = LinearRegression()
modelo_poli.fit(X_poly_train, y_train)
print(f"Los coeficientes de la ecuación son: {modelo_poli.coef_.tolist()} +

X_poly_test = poly_features.fit_transform(X_test)
y_pred_poli = modelo_poli.predict(X_poly_test)
poli_errors = np.abs(y_test - y_pred_poli)
print(f"Los errores son: \n{poli_errors}")

poli_r2 = modelo_poli.score(X_poly_test,y_test)
print(f"La R2 poli es: {poli_r2:.4f}")
```

```
Los coeficientes de la ecuación son: [[938550.8073161189, -960686.8547207173, -
693.7846496749511, -47.63397510631104, -1973609.386869107, -3555945.054284187,
-261805.3761400745, 334632.7155465354, 1140975.1967470353, 96.33131274532772, -
231.45547734928482, -86499.48583650803, -2497.189427245845, 3657.1667213466526,
-12.883062427039363, 919.3924295345901, 5764.507955755042, -5.823495193564668,
0.272133015108011, 17285.589977331347, 20386.405989277348, -5856.310554966901,
-3495.263583538805, -1405.6162397108876, -14.37830856724986, -20.9787400877394
25, -489.6758807975284, -7.268747180116833, 18.927820278018544, 0.1695905244541
791, -8083.679230394810 5, 21.699707513580506, -0.6322588541679579, -30527.18434
3519857, -75.29594570762326, 14827.611391947634, 1472.549847323498, 22953.60129
330484, 22.331342221557406, -0.17337900816758633, 434.7240772195196 4, -20.52147
238034791, -33.179066235180585, 0.23569175543309484, -104.7252446319907 9, 3528.
9198612030596, -17.607432025029482, 164.83998476339897, -8.865197654757168, 8.7
5378929432304 2, 22.244645262323957, 911.5461234393297, -68.55612356355414, 305.
60545956366695, -44.07100272618118, -975.9397793503013, -1273.647353616077 5, 1.
058273483067751e-06, 0.146920950561479 9, -1.751961722420738, -0.018179776522856
628, 0.0516696024915290 76, 0.10013062898815406, -3528.920464913244 4, -3528.9202
09317701, 0.02377765951678157, -4.754328983835876e-05, 0.00037219002842903137,
3.1564850360155106e-06, 26686.896011884357, -57387.34301194945, 1049.6335479799
097, 26241.37157319107, 2475.464014972754, -16.882349951375943, 0.5948293174268
997, 943.6353149088435, 2.430649573119192, -6.434829361346829, -0.0768765287266
1488, -3557294.0602460834, -41258.93462396657, -1580.220163776031, -175166.5895
175321, 124.57566406902579, 40.23971277587221, 4022.649726092837, -55.790225388
427885, 246.24717005032466, 0.6921495334201282, 11580.297220169254, 8416.276921
291743, 15602.585337959867, -1.088524220246071 6, -7.836907605972513, 65.1757535
6804424, -4.199765098990251, -6.886773035466305, -0.21067223143393 22, -676.4031
083689728, -9726.241061401617, 0.33919450207679347, 8.586635147996958, -185.177
65408895463, -17.019920941866694, 49.82418506812252, -0.3297141212522092, 6647.
20556023012 5, 4.268429557765103, 14.747560748618639, -592.045542164994, -1.1422
99903876392, -22.502797825166454, -0.3801733170548687, -806.8039997821907, -63
3.5202265850385, -305.3746773144812 3, 44.10556815236487, 975.9317716455553, 127
3.6471558995545, 173.20239687997673, -305.20548661224893, 44.11605681916626, 97
5.9554743494082, 1273.6469277154538, 23.146800528368658, 0.426670655760972 3, -
1.8833872211107519, 0.007737979176454246, 0.8367935734713683, 0.041876504754327
47, 0.00010928031406365335, 0.03419372078496963, 7.308600470423698e-05, 1.40059
27368998528e-06]] + [81001931.61223687]
Los errores son: 
               price
15783  130597.803403
14209  126581.225352
```

```
13532    9327.132665
10846  189975.571900
15952   99953.558813
...              ...
11362  109882.239591
1159     9289.376986
12939  339435.089963
19484  889597.197113
2138    69292.594221

[4323 rows x 1 columns]
La R2 poli es: 0.6872
```

In [53]:

```python
#Modelo Ridge
modelo_ridge = Ridge(alpha=300)
modelo_ridge.fit(X_poly_train, y_train)
print(f"Los coeficientes de la ecuación son: {modelo_ridge.coef_.tolist()} +


y_pred_ridge = modelo_ridge.predict(X_poly_test)
ridge_errors = np.abs(y_test - y_pred_ridge)
print(f"Los errores son: \n{ridge_errors}")

ridge_r2 = modelo_ridge.score(X_poly_test,y_test)
print(f"La R2 ridge es: {ridge_r2:.4f}")
```

```
Los coeficientes de la ecuación son: [[3274.666449015365, -1347.199865038216, -
193.78613636685623, -47.40746067639414, -1158.1839650086229, -193.9203621752414
8, -1449.403668712095, 2022.3520177404278, 1972.6622568635394, -224.92792080510
839, 30.919636778205778, -57835.98013255099, -2404.525006947526, 3923.043117758
1454, -18.640663250744026, 846.5721627017399, -1950.206981823567, -12.344937350
478304, 0.26807072630197815, 10392.43159205991, 3622.3787872150683, -4309.92655
30817265, 1741.0840239681015, -1561.8613704114434, -3.289538392339913, -8.74786
3497667712, -11.53451413989098, 2.021556606701465, 15.703415412691578, 0.13408
266026224772, -1359.8505829597857, 21.699548249945042, -0.6202954680448688, -11
556.667544995229, -879.3063988283002, 13082.017770115941, -6872.781652361353, 1
1774.567600690612, 20.006278179070648, 1.3624761271469314, -20.567615180940866,
-29.542951858408212, -20.66299944938646, 0.2569964066850383, 0.25282395861973 2
3, 0.3680196300258921 6, -19.07542812557368, 137.72180778506547, -6.888482108100
172, 8.8611370529246 5, 26.237508274093365, -0.3784316320313897, 0.1338531239665
5272, 0.02667402947016694, 0.2955773198419278, 0.02449739778456953, -0.15508777
493940756, 1.057929471429375e-06, 0.1446546387070513 4, -1.0113975632957 9, -0.00
0902382948463525, 0.063276246549722 2, 0.1068971540577216, -0.3686199741325986 7,
-0.3683659207387868, 0.0236357234512445 2, 1.0033286865201788e-05, 0.00034250777
1332059, 3.193961256504483 4e-06, 22002.590168889263, -2282.193396147735 3, 264.1
8655890612985, 7302.05695807968, 1970.1068411081903, -15.750529909729755, -4.07
4960126375362, -6.025376861970772, -22.84669154336504, -13.090890087738154, -0.
18019310940771271, -193.7981642951833, -3871.723187461725, -3321.3247844063026,
-33106.28233493656, 73.88362861645587, 63.49904727159538, -97.65014110021495, -
81.44159055038216, 154.78023835114476, 0.7179541736244538, 10864.9803350152, 46
64.550551146465, 9598.942864092834, 2.182190461886857 5, -9.35270342483349 1, -4
5.7123904501414, -6.325196504694778, -1.8417981937324286, -0.24683320879457576,
487.9322105173725, -2231.8087366244295, 1.6991493655426868, 6.730268489675621,
-33.2752705178666, -11.791569174763778, 45.39613023755198 6, -0.322705908465775 7
4, 5904.878566844368, 3.809907198314118, 22.93064933075265, -9.05012055006518,
7.682819836956442, -26.465019387327633, -0.42671034253076406, 0.14219710411461 9
22, -0.24703772881114536, 0.0905740961550511, -0.2597973988828469, -0.031459226
395160196, 0.15489372444208693, -0.4735717916628699, -0.05152497850956082, -0.2
564136873283057, -0.013862643450068808, 0.1546511393318865 4, 14.9118210661067 5,
0.3581615680089434 4, -2.00009399994712, 0.010813341576341326, 0.82860185803122 7
3, 0.039023699125817976, 8.538058235611151e-05, 0.034503483277682 88, 0.00013643
077219863348, 1.6863594698189658e-06]] + [56380114.10617158]
Los errores son:
               price
15783  121474.030567
14209  136390.233417
13532    9075.517326
10846  188060.177791
15952   99227.980183
...              ...
```

```
11362  106947.111726
1159     1081.641701
12939  331200.685652
19484  893246.891967
2138    60406.641731

[4323 rows x 1 columns]
La R2 ridge es: 0.6868
```

In [54]:

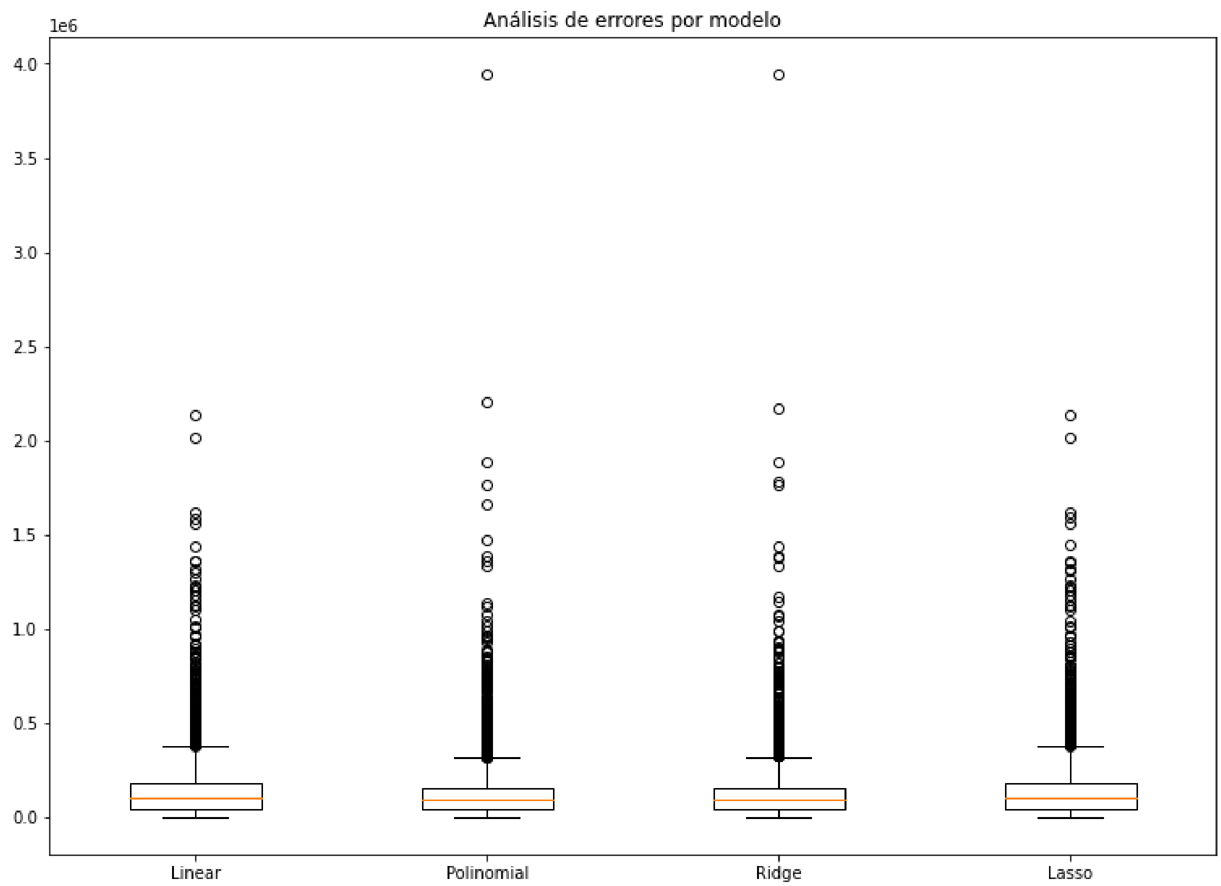
```python
#Modelo lasso
modelo_lasso = Lasso(alpha=50, max_iter=10000)
modelo_lasso.fit(X_train, y_train)
print(f"Los coeficientes de la ecuación son: {modelo_lasso.coef_.tolist()} +


y_pred_lasso = modelo_lasso.predict(X_test)
lasso_errors = np.abs(y_test - y_pred_lasso[:, np.newaxis])
print(f"Los errores son: \n{lasso_errors}")

lasso_r2 = modelo_lasso.score(X_test,y_test)
print(f"La R2 lasso es: {lasso_r2:.4f}")

#El modelo no convergio, requiere regularización
```

```
Los coeficientes de la ecuación son: [-39461.16770997991, 45997.54809271042, 30
1.30095576133806, -0.007810330863785161, 25405.698358738013, 577443.3834541187,
42602.02204860094, 19196.352901140308, 118625.08326616781, -131.56595312120956,
-126.74296943683832, -3557.6641042707765, 12.294101377579171, 18.3780134984115,
-0.5299221792815929] + [6176949.530841906]
Los errores son: 
              price
15783  5.127926e+04
14209  1.788603e+05
13532  1.038161e+05
10846  2.143938e+05
15952  3.769328e+04
...             ...
11362  1.989671e+05
1159   3.398334e+04
12939  3.745735e+05
19484  1.204901e+06
2138   5.171355e+04

[4323 rows x 1 columns]
La R2 lasso es: 0.6587

C:\Users\sergi\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_des
cent.py:648: ConvergenceWarning: Objective did not converge. You might want to
increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 3.780e+14, tolerance: 2.366e+11
  model = cd_fast.enet_coordinate_descent(
```

In [55]:

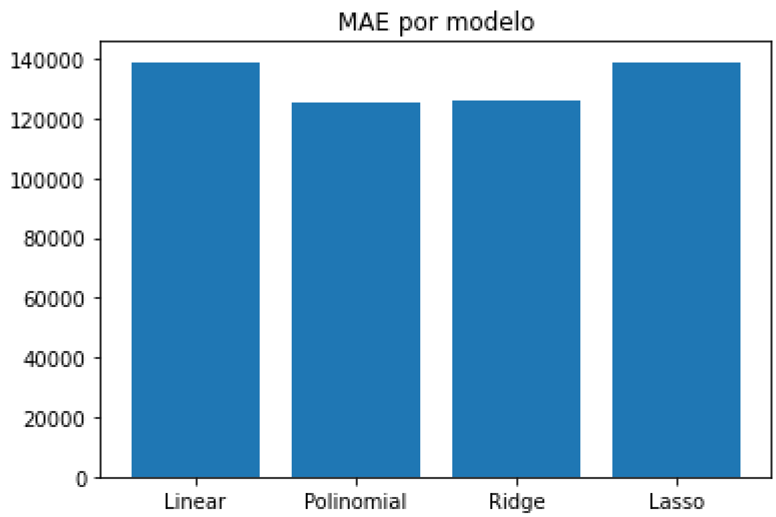
```python
#Grafica MAE de los 4 modelos

fig = plt.figure(figsize =(10, 7))

ax = fig.add_axes([0, 0, 1, 1])
ax.set_xticklabels(['Linear', 'Polinomial',
                    'Ridge', 'Lasso'])
plt.title("Análisis de errores por modelo")
bp = ax.boxplot([lin_errors.values.flatten(), poli_errors.values.flatten(),
plt.show()
```

```
C:\Users\sergi\AppData\Local\Temp/ipykernel_21344/3240669436.py:6: UserWarning:
FixedFormatter should only be used together with FixedLocator
  ax.set_xticklabels(['Linear', 'Polinomial',
```



In [56]:

```
#Graficas de MAE
plt.bar(['Linear', 'Polinomial','Ridge', 'Lasso'],
    [lin_errors.values.mean(), poli_errors.values.mean(), ridge_errors.value
    )
plt.title("MAE por modelo")
plt.show()
```



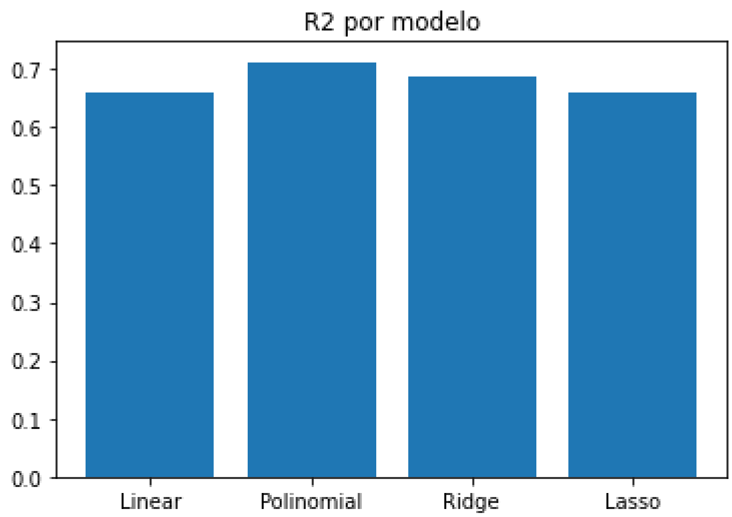
In [57]:

```
[lin_errors.values.mean(), poli_errors.values.mean(), ridge_errors.values.me
```

Out[57]: [138988.2678292845, 125521.55273071026, 125871.51142084808, 138990.79981667033]

In [58]:

```
#Graficas de R2
plt.bar(['Linear', 'Polinomial','Ridge', 'Lasso'],
    [lin_r2, poly_r2, ridge_r2, lasso_r2],
    )
plt.title("R2 por modelo")
plt.show()
```



In [59]:

```
ridge_r2
```

Out[59]: 0.686756780640699

Conclusiones Ejercicio 2

Explica tus resultados, ¿que porcentajes de entrenamiento y evaluación?

Se emplearon las particiones con 80% de entrenamiento y 20% de pruebas.

¿que método se aproxima mejor, ¿por que?, ¿que error tienes?, ¿es bueno?, ¿cómo lo sabes?

El modelo que más conviene a la empresa es el modelo de generado por Regresión Polinómica, esto ya que de los 4 fue el que mantuvo un menor error (MAE = 125,521) y mayor R2 (68.72%).

Se podría mejorar el cálculo de los modelos aplicandoles pipelines que estandaricen los datos. Ya que el dataset incluye variables en escalas diferentes

Ejercicio 3

Análisis de tiendas Target

In [60]:

```
pip install geopy
```

```
Requirement already satisfied: geopy in c:\users\sergi\anaconda3\lib\site-packa
ges (2.2.0)
Requirement already satisfied: geographiclib<2,>=1.49 in c:\users\sergi\anacond
a3\lib\site-packages (from geopy) (1.52)
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [61]:

```
#Instalamos librerias necesaria
! pip install qeds fiona geopandas xgboost gensim folium pyLDAvis descartes
```

```
Requirement already satisfied: qeds in c:\users\sergi\anaconda3\lib\site-packag
es (0.7.0)
Requirement already satisfied: fiona in c:\users\sergi\anaconda3\lib\site-packa
ges (1.8.22)
Requirement already satisfied: geopandas in c:\users\sergi\anaconda3\lib\site-p
ackages (0.12.1)
Requirement already satisfied: xgboost in c:\users\sergi\anaconda3\lib\site-pac
kages (1.7.1)
Requirement already satisfied: gensim in c:\users\sergi\anaconda3\lib\site-pack
ages (4.2.0)
Requirement already satisfied: folium in c:\users\sergi\anaconda3\lib\site-pack
ages (0.13.0)
Requirement already satisfied: pyLDAvis in c:\users\sergi\anaconda3\lib\site-pa
ckages (3.3.1)
Requirement already satisfied: descartes in c:\users\sergi\anaconda3\lib\site-p
ackages (1.1.0)
Requirement already satisfied: seaborn in c:\users\sergi\anaconda3\lib\site-pac
kages (from qeds) (0.11.2)
Requirement already satisfied: quantecon in c:\users\sergi\anaconda3\lib\site-p
ackages (from qeds) (0.5.3)
Requirement already satisfied: numpy in c:\users\sergi\anaconda3\lib\site-packa
ges (from qeds) (1.20.3)
Requirement already satisfied: scikit-learn in c:\users\sergi\anaconda3\lib\sit
e-packages (from qeds) (1.1.2)
Requirement already satisfied: quandl in c:\users\sergi\anaconda3\lib\site-pack
ages (from qeds) (3.7.0)
Requirement already satisfied: pandas-datareader in c:\users\sergi\anaconda3\li
b\site-packages (from qeds) (0.10.0)
Requirement already satisfied: pyarrow in c:\users\sergi\anaconda3\lib\site-pac
kages (from qeds) (10.0.0)
Requirement already satisfied: scipy in c:\users\sergi\anaconda3\lib\site-packa
ges (from qeds) (1.7.1)
Requirement already satisfied: requests in c:\users\sergi\anaconda3\lib\site-pa
ckages (from qeds) (2.26.0)


[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip


Requirement already satisfied: plotly in c:\users\sergi\anaconda3\lib\site-pack
ages (from qeds) (5.11.0)
Requirement already satisfied: statsmodels in c:\users\sergi\anaconda3\lib\site
-packages (from qeds) (0.12.2)
Requirement already satisfied: matplotlib in c:\users\sergi\anaconda3\lib\site-
packages (from qeds) (3.4.3)
Requirement already satisfied: pandas in c:\users\sergi\anaconda3\lib\site-pack
ages (from qeds) (1.3.4)
Requirement already satisfied: openpyxl in c:\users\sergi\anaconda3\lib\site-pa
ckages (from qeds) (3.0.9)
Requirement already satisfied: click>=4.0 in c:\users\sergi\anaconda3\lib\site-
packages (from fiona) (8.0.3)
Requirement already satisfied: six>=1.7 in c:\users\sergi\anaconda3\lib\site-pa
ckages (from fiona) (1.16.0)
```

```
Requirement already satisfied: setuptools in c:\users\sergi\anaconda3\lib\site-
packages (from fiona) (65.4.0)
Requirement already satisfied: certifi in c:\users\sergi\anaconda3\lib\site-pac
kages (from fiona) (2021.10.8)
Requirement already satisfied: click-plugins>=1.0 in c:\users\sergi\anaconda3\l
ib\site-packages (from fiona) (1.1.1)
Requirement already satisfied: munch in c:\users\sergi\anaconda3\lib\site-packa
ges (from fiona) (2.5.0)
Requirement already satisfied: attrs>=17 in c:\users\sergi\anaconda3\lib\site-p
ackages (from fiona) (21.2.0)
Requirement already satisfied: cligj>=0.5 in c:\users\sergi\anaconda3\lib\site-
packages (from fiona) (0.7.2)
Requirement already satisfied: packaging in c:\users\sergi\anaconda3\lib\site-p
ackages (from geopandas) (21.0)
Requirement already satisfied: shapely>=1.7 in c:\users\sergi\anaconda3\lib\sit
e-packages (from geopandas) (1.8.5.post1)
Requirement already satisfied: pyproj>=2.6.1.post1 in c:\users\sergi\anaconda3
\lib\site-packages (from geopandas) (3.4.0)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\sergi\anaconda3\li
b\site-packages (from gensim) (5.2.1)
Requirement already satisfied: Cython==0.29.28 in c:\users\sergi\anaconda3\lib
\site-packages (from gensim) (0.29.28)
Requirement already satisfied: jinja2>=2.9 in c:\users\sergi\anaconda3\lib\site
-packages (from folium) (2.11.3)
Requirement already satisfied: branca>=0.3.0 in c:\users\sergi\anaconda3\lib\si
te-packages (from folium) (0.6.0)
Requirement already satisfied: sklearn in c:\users\sergi\anaconda3\lib\site-pac
kages (from pyLDAvis) (0.0.post1)
Requirement already satisfied: funcy in c:\users\sergi\anaconda3\lib\site-packa
ges (from pyLDAvis) (1.17)
Requirement already satisfied: joblib in c:\users\sergi\anaconda3\lib\site-pack
ages (from pyLDAvis) (1.1.0)
Requirement already satisfied: numexpr in c:\users\sergi\anaconda3\lib\site-pac
kages (from pyLDAvis) (2.7.3)
Requirement already satisfied: future in c:\users\sergi\anaconda3\lib\site-pack
ages (from pyLDAvis) (0.18.2)
Requirement already satisfied: colorama in c:\users\sergi\anaconda3\lib\site-pa
ckages (from click>=4.0->fiona) (0.4.4)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\sergi\anaconda3\lib
\site-packages (from jinja2>=2.9->folium) (1.1.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\sergi\anaconda3\lib\sit
e-packages (from pandas->qeds) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\sergi\anacond
a3\lib\site-packages (from pandas->qeds) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\sergi\anaconda3\lib
\site-packages (from matplotlib->qeds) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\sergi\anaconda3\li
b\site-packages (from matplotlib->qeds) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\sergi\anaconda3\lib\sit
e-packages (from matplotlib->qeds) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\sergi\anaconda3\lib\si
te-packages (from matplotlib->qeds) (8.4.0)
Requirement already satisfied: et-xmlfile in c:\users\sergi\anaconda3\lib\site-
packages (from openpyxl->qeds) (1.1.0)
Requirement already satisfied: lxml in c:\users\sergi\anaconda3\lib\site-packag
es (from pandas-datareader->qeds) (4.6.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\sergi\anac
```

```
onda3\lib\site-packages (from requests->qeds) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sergi\anaconda3\lib\sit
e-packages (from requests->qeds) (3.2)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\sergi\anaconda
3\lib\site-packages (from requests->qeds) (1.26.7)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\sergi\anaconda3\lib
\site-packages (from plotly->qeds) (8.1.0)
Requirement already satisfied: inflection>=0.3.1 in c:\users\sergi\anaconda3\li
b\site-packages (from quandl->qeds) (0.5.1)
Requirement already satisfied: more-itertools in c:\users\sergi\anaconda3\lib\s
ite-packages (from quandl->qeds) (8.10.0)
Requirement already satisfied: numba in c:\users\sergi\anaconda3\lib\site-packa
ges (from quantecon->qeds) (0.54.1)
Requirement already satisfied: sympy in c:\users\sergi\anaconda3\lib\site-packa
ges (from quantecon->qeds) (1.9)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\sergi\anaconda3
\lib\site-packages (from scikit-learn->qeds) (2.2.0)
Requirement already satisfied: patsy>=0.5 in c:\users\sergi\anaconda3\lib\site-
packages (from statsmodels->qeds) (0.5.2)
Requirement already satisfied: llvmlite<0.38,>=0.37.0rc1 in c:\users\sergi\anac
onda3\lib\site-packages (from numba->quantecon->qeds) (0.37.0)
Requirement already satisfied: mpmath>=0.19 in c:\users\sergi\anaconda3\lib\sit
e-packages (from sympy->quantecon->qeds) (1.2.1)
```

In [62]:

```python
#Librerias para números y dataframes
import numpy as np
import pandas as pd

#Librerias para manejo de coordenadas
import geopandas as gpd
from shapely.geometry import Point
from geopy.geocoders import Nominatim

#Librerias para gráficos
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns; sns.set()

#Librerias para Machine Learning
from sklearn.cluster import KMeans

#Otras librerias
from tqdm import tqdm
import qeds

#Importamos los datos y exploramos
url="https://raw.githubusercontent.com/marypazrf/bdd/main/target-locations.c
df=pd.read_csv(url)
df.head()
```

Out[62]:

	name	latitude	longitude	address	phone	website
0	Alabaster	33.224225	-86.804174	250 S Colonial Dr, Alabaster, AL 35007-4657	205-564-2608	https://www.target.com/sl/alabaster/2276
1	Bessemer	33.334550	-86.989778	4889 Promenade Pkwy, Bessemer, AL 35022-7305	205-565-3760	https://www.target.com/sl/bessemer/2375
2	Daphne	30.602875	-87.895932	1698 US Highway 98, Daphne, AL 36526-4252	251-621-3540	https://www.target.com/sl/daphne/1274
3	Decatur	34.560148	-86.971559	1235 Point Mallard Pkwy SE, Decatur, AL 35601-...	256-898-3036	https://www.target.com/sl/decatur/2084
4	Dothan	31.266061	-85.446422	4601 Montgomery Hwy, Dothan, AL 36303-1522	334-340-1112	https://www.target.com/sl/dothan/1468

In [63]:

```python
#Revisamos si tienen nulos
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1839 entries, 0 to 1838
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   name       1839 non-null   object 
 1   latitude   1839 non-null   float64
 2   longitude  1839 non-null   float64
 3   address    1839 non-null   object 
 4   phone      1839 non-null   object 
 5   website    1839 non-null   object 
dtypes: float64(2), object(4)
memory usage: 86.3+ KB
```

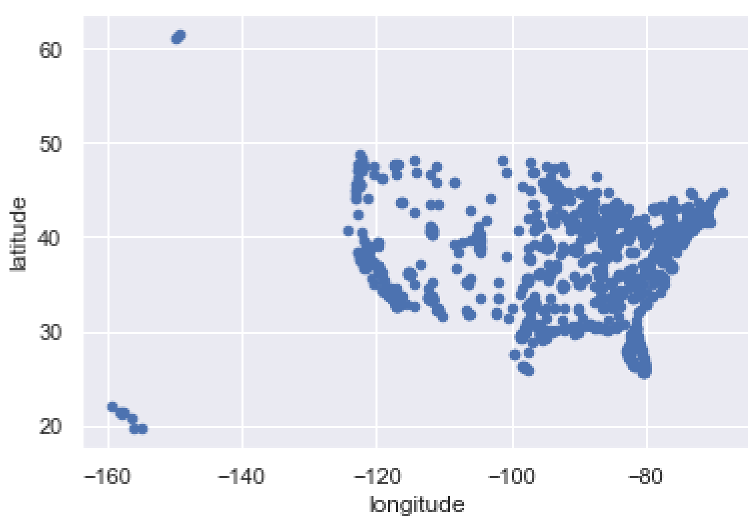
In [64]:

```python
#Graficamos rápidamente
latlong=df[["latitude","longitude"]]

#extrae los datos interesantes
latlong.plot.scatter( "longitude","latitude")
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with *
x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
```

Out[64]:

```
<AxesSubplot:xlabel='longitude', ylabel='latitude'>
```



In [65]:

```python
#Revisamos la distribución estadística de los datos
latlong.describe()
```

Out[65]:

	latitude	longitude
count	1839.000000	1839.000000
mean	37.791238	-91.986881
std	5.272299	16.108046
min	19.647855	-159.376962
25%	33.882605	-98.268828
50%	38.955432	-87.746346
75%	41.658341	-80.084833
max	61.577919	-68.742331

In [66]:

```python
#Agregamos un estilo
qeds.themes.mpl_style();

#Definimos nuevo dataframe solo con las coordenadas de las tiendas y lo most
df["Coordinates"] = list(zip(df.longitude, df.latitude))
df["Coordinates"] = df["Coordinates"].apply(Point)
df.head()
```

Out[66]:

	name	latitude	longitude	address	phone	website	
0	Alabaster	33.224225	-86.804174	250 S Colonial Dr, Alabaster, AL 35007-4657	205-564-2608	https://www.target.com/sl/alabaster/2276	(-86.
1	Bessemer	33.334550	-86.989778	4889 Promenade Pkwy, Bessemer, AL 35022-7305	205-565-3760	https://www.target.com/sl/bessemer/2375	(-86.
2	Daphne	30.602875	-87.895932	1698 US Highway 98, Daphne, AL 36526-4252	251-621-3540	https://www.target.com/sl/daphne/1274	(-87.
3	Decatur	34.560148	-86.971559	1235 Point Mallard Pkwy SE, Decatur, AL 35601-...	256-898-3036	https://www.target.com/sl/decatur/2084	POI
4	Dothan	31.266061	-85.446422	4601 Montgomery Hwy, Dothan, AL 36303-1522	334-340-1112	https://www.target.com/sl/dothan/1468	POI

In [67]:

```
#Convertimos el DataFrame a Geoespacial
gdf = gpd.GeoDataFrame(df, geometry="Coordinates")
gdf.head()
```

Out[67]:

	name	latitude	longitude	address	phone	website	Coor
0	Alabaster	33.224225	-86.804174	250 S Colonial Dr, Alabaster, AL 35007-4657	205-564-2608	https://www.target.com/sl/alabaster/2276	(-8 33
1	Bessemer	33.334550	-86.989778	4889 Promenade Pkwy, Bessemer, AL 35022-7305	205-565-3760	https://www.target.com/sl/bessemer/2375	(-8 33
2	Daphne	30.602875	-87.895932	1698 US Highway 98, Daphne, AL 36526-4252	251-621-3540	https://www.target.com/sl/daphne/1274	(-8 30
3	Decatur	34.560148	-86.971559	1235 Point Mallard Pkwy SE, Decatur, AL 35601-...	256-898-3036	https://www.target.com/sl/decatur/2084	(-8 34
4	Dothan	31.266061	-85.446422	4601 Montgomery Hwy, Dothan, AL 36303-1522	334-340-1112	https://www.target.com/sl/dothan/1468	(-8 31

In [68]:

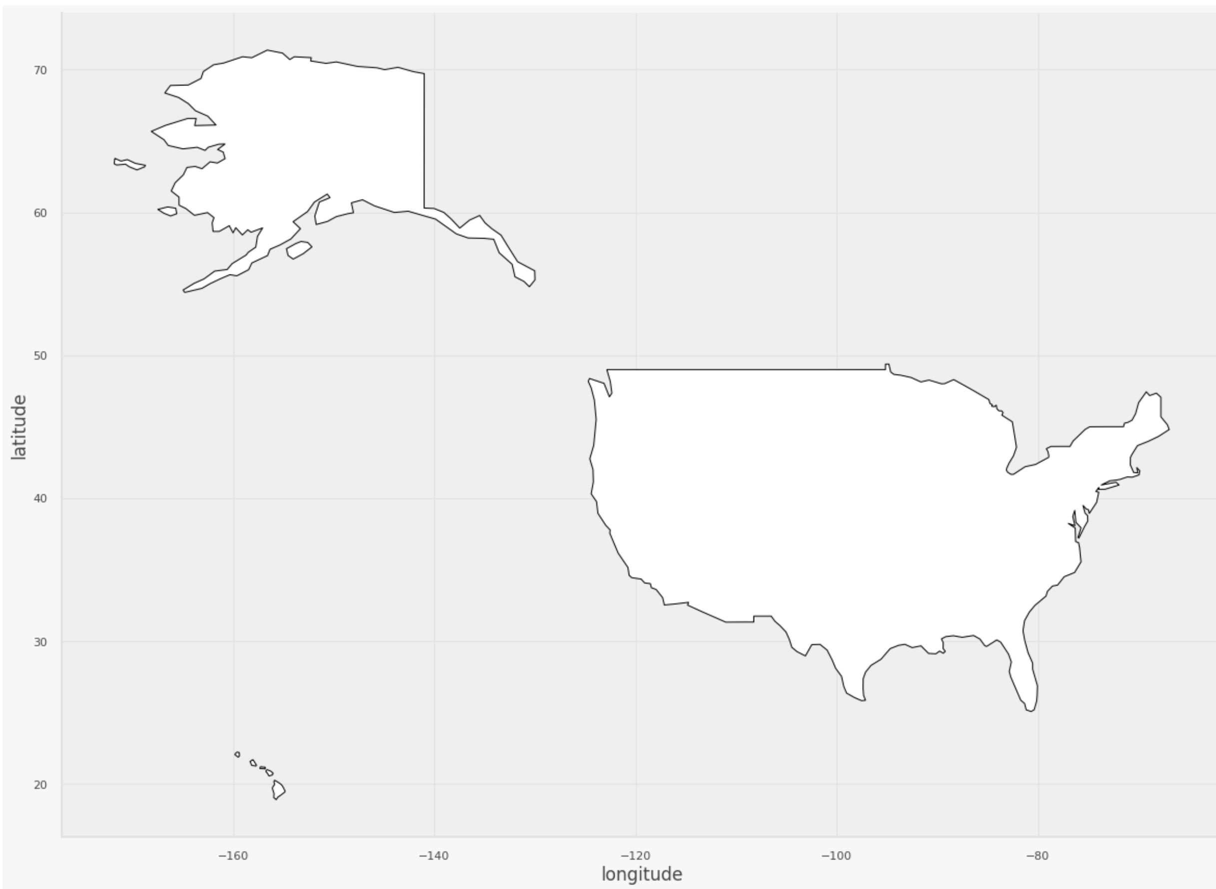
```
#Buscamos el nombre del mapa que queremos usar para graficar, este caso Esta
world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
world = world.set_index("iso_a3")
world.head()
```

Out[68]:

	pop_est	continent	name	gdp_md_est	geometry
iso_a3					
FJI	889953.0	Oceania	Fiji	5496	MULTIPOLYGON (((180.00000 -16.06713, 180.00000...
TZA	58005463.0	Africa	Tanzania	63177	POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
ESH	603253.0	Africa	W. Sahara	907	POLYGON ((-8.66559 27.65643, -8.66512 27.58948...
CAN	37589262.0	North America	Canada	1736425	MULTIPOLYGON (((-122.84000 49.00000, -122.9742...
USA	328239523.0	North America	United States of America	21433226	MULTIPOLYGON (((-122.84000 49.00000, -120.0000...

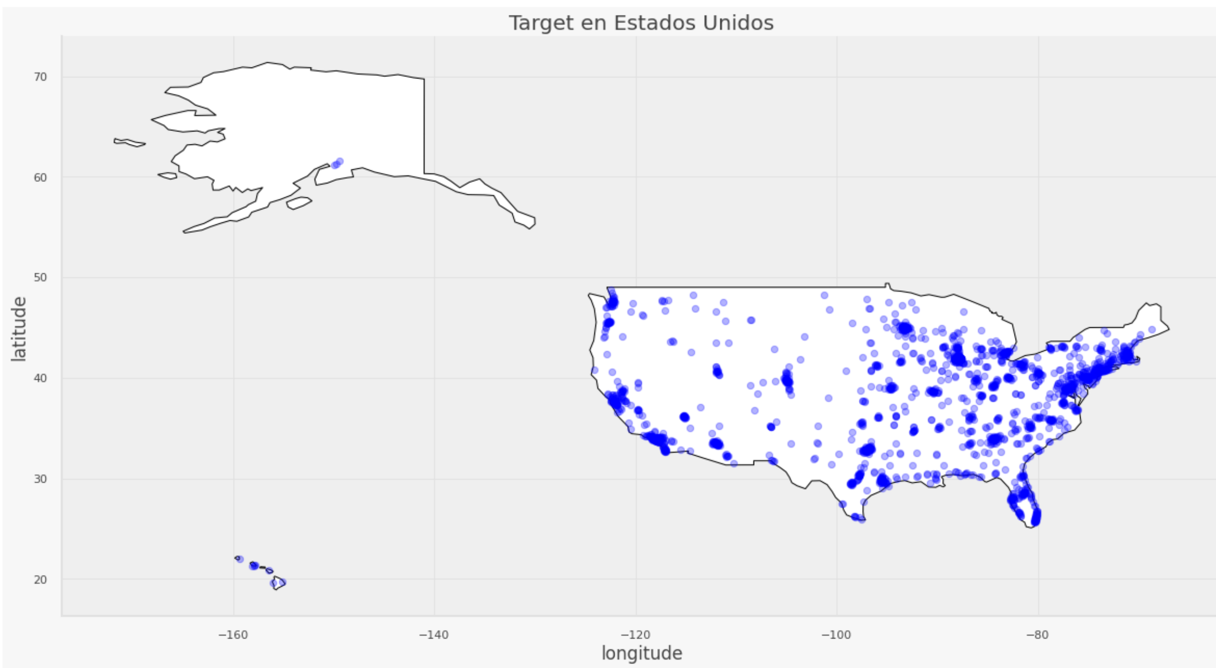
In [69]:

```python
#Generamos primer gráfica solo con el mapa

#Definimos el tamaño del gráfico
fig, gax = plt.subplots(figsize=(20,20))

#Agregamos la capa del mapa
world.query("name == 'United States of America'").plot(ax=gax, edgecolor='bl

#Nombramos los ejes
gax.set_xlabel('longitude')
gax.set_ylabel('latitude')

#Quitamos los límites de las cajas
gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)
```



In [70]:

```
#Generamos segundo gráfico con el mapa y las tiendas

#Definimos el tamaño del gráfico
fig, gax = plt.subplots(figsize=(20,20))

#Agregamos la capa del mapa
world.query("name == 'United States of America'").plot(ax = gax, edgecolor='

#Agregamos la capa de las tiendas
gdf.plot(ax=gax, color='blue', alpha = 0.3, markersize=40)

#Nombramos los ejes y el gráfico
gax.set_xlabel('longitude')
gax.set_ylabel('latitude')
gax.set_title('Target en Estados Unidos')

#Quitamos los límites de las cajas
gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)

plt.show()
```



In [71]:

```
#Definimos número de almacenes deseados
num_almacenes=10

#Ajustamos el modelo
kmeans = KMeans(n_clusters=num_almacenes, random_state=42).fit(latlong)

#Recuperamos la distancia de cada tienda a los almacenes
kmeans_distances = KMeans(n_clusters=num_almacenes, random_state=42).fit_tra
```

In [72]:

```
#Para analizar la distancia máxima entre tienda y almacen asignado
df_center_distances = pd.DataFrame(kmeans_distances)            #El numpy arr
df_center_distances["Class"] = kmeans.labels_.reshape(-1,1)     #Al DF le agr

#Calculamos la distancia máxima
df_max_distance = pd.DataFrame(np.diag(df_center_distances.groupby(["Class"]
df_max_distance.columns=["Max Distancia"]         #Le pongo nombre a la colum

#Calculamos la distancia promedio
df_ave_distance = pd.DataFrame(np.diag(df_center_distances.groupby(["Class"]
df_ave_distance.columns=["Prom Distancia"]         #Le pongo nombre a la colu

#Imprimimos ambas tablas
print(f"********** DISTANCIAS MAXIMAS ENTRE TIENDAS Y {num_almacenes} ALMACE
print(f"El promedio de distancias máximas es: {df_max_distance.values.mean()
print(f"********** DISTANCIAS PROMEDIO ENTRE TIENDAS Y {num_almacenes} ALMAC
print(f"El promedio de distancias promedios es: {df_ave_distance.values.mean
```

```
********** DISTANCIAS MAXIMAS ENTRE TIENDAS Y 10 ALMACENES
El promedio de distancias máximas es: 9.0634
y se distribuyen de la siguiente forma:
    Max Distancia
0        6.678997
1       30.677046
2        8.880668
3        4.856152
4        7.692105
5        6.058998
6       10.407559
7        6.632734
8        2.573167
9        6.176218

********** DISTANCIAS PROMEDIO ENTRE TIENDAS Y 10 ALMACENES
El promedio de distancias promedios es: 2.9102
y se distribuyen de la siguiente forma:
    Prom Distancia
0         3.542579
1         3.349093
2         2.843543
3         2.658125
4         3.044052
5         1.775038
6         4.914387
7         2.951312
8         1.305184
9         2.719061
```

In [73]:

```python
#Empaquetamos las coordenadas en lista de tuplas
centers = kmeans.cluster_centers_                                        #Array c
Lat = list()
Long = list()
Lat = centers[:,0]
Long = centers[:,1]
tmp1 = list(zip(Lat, Long))

#Usamos Geolocator para recuperar el estado y ciudad de las tuplas
tmp2 = list()
geolocator = Nominatim(user_agent="Test")

for i in range(num_almacenes):
  location = geolocator.reverse(tmp1[i])
  address = location.raw['address']
  state = address.get('state', '')
  if state == "":
    state = "ND"
  city = address.get('city', '')
  if city == "":
    city = "ND"
  tmp2.append((state, city))

#Generamos DF de almacenes y anexamos coordenadas, ciudad, estado y número d
df_centers = pd.DataFrame()                                              #Iniciam
df_centers["Coordinates"] = list(zip(centers[:,1], centers[:,0]))   #Empaque
df_centers["Coordinates"] = df_centers["Coordinates"].apply(Point)  #Les apl
df_centers["State"] =[i[0] for i in tmp2]
df_centers["City"] =[i[1] for i in tmp2]
df_centers["Tiendas"] = df_center_distances.groupby(["Class"]).count()[0]

#Convertimos el DF en uno geoespacial para las graficas
gdf_centers = gpd.GeoDataFrame(df_centers, geometry="Coordinates")
gdf_centers
```

Out[73]:

	Coordinates	State	City	Tiendas
0	POINT (-82.79528 30.23350)	Florida	ND	218
1	POINT (-122.66306 46.97944)	Washington	ND	73
2	POINT (-93.95728 43.22839)	Iowa	ND	148
3	POINT (-78.75614 38.49744)	Virginia	ND	240
4	POINT (-96.17848 31.86383)	Texas	ND	206
5	POINT (-73.66385 41.30782)	New York	ND	280
6	POINT (-108.67721 37.41393)	Colorado	ND	130
7	POINT (-86.69027 40.71134)	Indiana	ND	317
8	POINT (-157.31225 20.94543)	ND	ND	8
9	POINT (-118.94133 35.43472)	California	ND	219

In [74]:
```
#Agregamos al GDF el almacen asignado a cada tienda
gdf["Class"]= kmeans.labels_.reshape(-1,1)
gdf
```

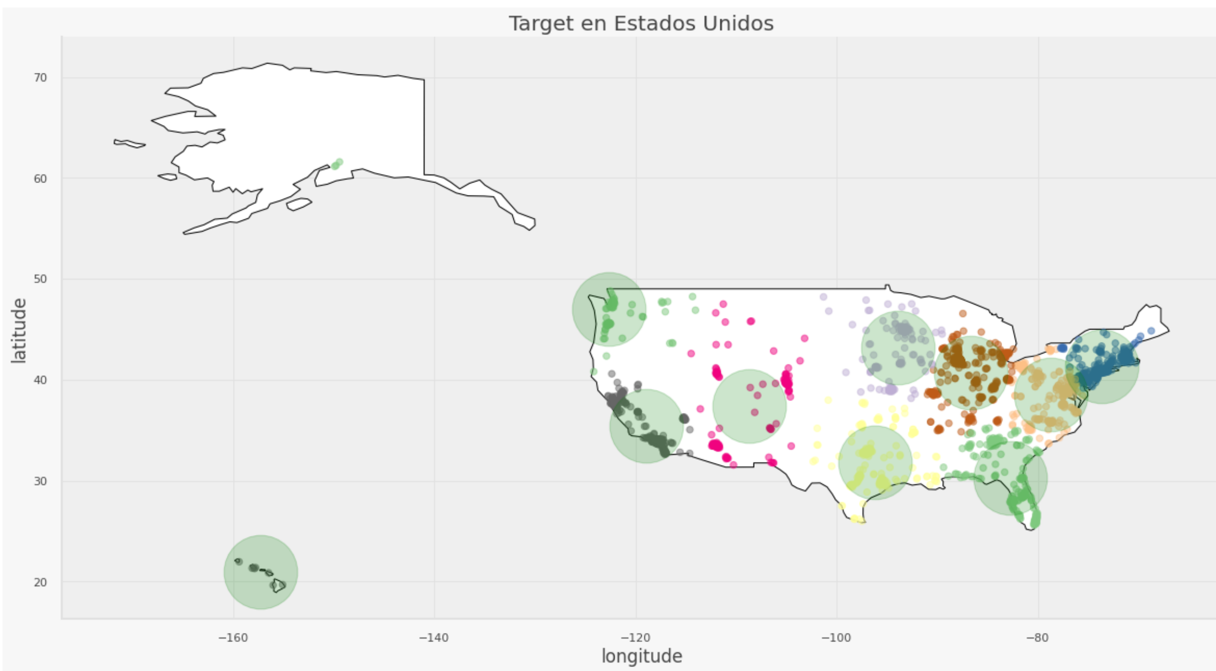
Out[74]:

	name	latitude	longitude	address	phone	website
0	Alabaster	33.224225	-86.804174	250 S Colonial Dr, Alabaster, AL 35007-4657	205-564-2608	https://www.target.com/sl/alabaster/2276
1	Bessemer	33.334550	-86.989778	4889 Promenade Pkwy, Bessemer, AL 35022-7305	205-565-3760	https://www.target.com/sl/bessemer/2375
2	Daphne	30.602875	-87.895932	1698 US Highway 98, Daphne, AL 36526-4252	251-621-3540	https://www.target.com/sl/daphne/1274
3	Decatur	34.560148	-86.971559	1235 Point Mallard Pkwy SE, Decatur, AL 35601-...	256-898-3036	https://www.target.com/sl/decatur/2084
4	Dothan	31.266061	-85.446422	4601 Montgomery Hwy, Dothan, AL 36303-1522	334-340-1112	https://www.target.com/sl/dothan/1468
...	...	...	...	...	...	...
1834	Waukesha	43.034293	-88.176840	2401 Kossow Rd, Waukesha, WI 53186-2904	262-784-8646	https://www.target.com/sl/waukesha/82
1835	Waukesha South	42.989604	-88.259806	1250 W Sunset Dr, Waukesha, WI 53189-8423	262-832-1272	https://www.target.com/sl/waukesha/2546
1836	Casper	42.846799	-106.264166	401 SE Wyoming Blvd, Casper, WY 82609-4219	307-265-8214	https://www.target.com/sl/casper/164
1837	Cheyenne	41.162019	-104.800048	1708 Dell Range Blvd, Cheyenne, WY 82009-4945	307-637-8888	https://www.target.com/sl/cheyenne/224
1838	Jackson Hole	43.469617	-110.789456	510 S Hwy 89, Jackson, WY 83001	307-200-3139	https://www.target.com/sl/jackson-hole/3409

1839 rows × 8 columns

In [75]:

```python
#Definimos el tamaño del gráfico
fig, gax = plt.subplots(figsize=(20,20))

#Agregamos la capa del mapa
world.query("name == 'United States of America'").plot(ax = gax, edgecolor='

#Agregamos la capa de las tiendas
gdf.plot(column= "Class", ax=gax, cmap="Accent",  alpha = 0.5, markersize=40

#Agregamos la capa de los almacenes
gdf_centers.plot(ax=gax, color='green', alpha = .2, markersize=5000)

#Nombramos los ejes y el gráfico
gax.set_xlabel('longitude')
gax.set_ylabel('latitude')
gax.set_title('Target en Estados Unidos')

#Quitamos los límites de las cajas
gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)

plt.show()
```



Conclusiones Ejercicio 3

Encuentra el numero ideal de almacenes, justifica tu respuesta:

Se escogió como el número ideal de almacenes de 10.

Encuentra las latitudes y longitudes de los almacenes, ¿que ciudad es?, ¿a cuantas tiendas va surtir?

En la tabla a continuación se muestran las latitudes y longitudes de cada uno de los almacenes, también en base a estas se calculo a qué Estado y Ciudad pertenecen. En particular los valores que indican ND se refieren a que la ubicación que se obtuvo no se encuentra dentro de una ciudad, en el caso del Estado con ND es porque sugiere una ubicación en el mar cerca de Hawaii.

Tambien se incluye el número de tiendas a las que atiende.

	Coordinates	State	City	Tiendas
0	POINT (-82.79528 30.23350)	Florida	ND	218
1	POINT (-122.66306 46.97944)	Washington	ND	73
2	POINT (-93.95728 43.22839)	Iowa	ND	148
3	POINT (-78.75614 38.49744)	Virginia	ND	240
4	POINT (-96.17848 31.86383)	Texas	ND	206
5	POINT (-73.66385 41.30782)	New York	ND	280
6	POINT (-108.67721 37.41393)	Colorado	ND	130
7	POINT (-86.69027 40.71134)	Indiana	ND	317
8	POINT (-157.31225 20.94543)	ND	ND	8
9	POINT (-118.94133 35.43472)	California	ND	219

¿Sabes a que distancia estara?

En la tabla a continuación se brinda un análisis basado en las distancias máximas y promedio de las tiendas con su respectivo almacen. Por ejemplo el almacen 1 tiene la mayor distancia con una tienda porque es el que debe dar servicio a Alaska. Se puede apreciar que el promedio de distancias máximas es de 9.0634 grados, mientras que el promedio de las distancias promedio es de 2.91 grados.

```
********** DISTANCIAS MAXIMAS ENTRE TIENDAS Y 10 ALMACENES
El promedio de distancias máximas es: 9.0634
y se distribuyen de la siguiente forma:
    Max Distancia
0       6.678997
1      30.677046
2       8.880668
3       4.856152
4       7.692105
5       6.058998
6      10.407559
7       6.632734
8       2.573167
9       6.176218

********** DISTANCIAS PROMEDIO ENTRE TIENDAS Y 10 ALMACENES
El promedio de distancias promedios es: 2.9102
y se distribuyen de la siguiente forma:
    Prom Distancia
0        3.542579
1        3.349093
2        2.843543
3        2.658125
4        3.044052
5        1.775038
6        4.914387
7        2.951312
8        1.305184
9        2.719061
```

¿Cómo elegiste el numero de almacenes? Justifica tu respuesta tecnicamente.

Se eligió el número de almacenes en base a las métricas previas, por ejemplo al modelar un total de 9 almacenes se obtenia que uno de ellos tendría que dar servicio a 450 tiendas, lo cual sonaba poco viable ya que en general se movian en un rango de 200 tiendas. Aunque bien se podría tomar un rango entre 9 y 11 almacenes, en unos casos, se puede observar en el mapa que hay tiendas alejadas de los almacenes

Adicionalmente, en el notebook notaras que al inicio exploramos los datos y los graficamos de manera simple, despues nos auxiliamos de una librería de datos geograficos.

¿Qué librerías nos pueden ayudar a graficar este tipo de datos?

PyCountry, GeoPy, Reverse Geocoder y GeoPandas

¿Consideras importante que se grafique en un mapa?, ¿por qué?

sí, ya que permite ejemplificar y explicar los motivos a la alta dirección sobre las ubicaciones de los almacenes y las tiendas.

In [ ]:

```
1
```

In [ ]:

```
```