Ejercicios 1 y 2 - Regresión lineal y polinomial

TecMty_Regresion_lineal_polinomial

November 7, 2022

1 Linear Models

1.0.1 Actividad Semanal #7

- Nombre: Rafael J. Mateo C
- Matrícula: A01793054
- Materia: Ciencia y Analítica de Datos
- Profesor: María de la Paz
- Fecha: 7 Nov 2022

2 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.

1. Importación y Análisis de los Datos Comenzaremos primero importando los datos y las librerías que usaremos.

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
import seaborn as sns
```

A continuación, definiremos algunas funciones que estaremos usando más adelante.

[ ]:
```
def plot_model(x, y_real, y_pred, title):

    #Grafica un diagrama de correlación
    plt.scatter(x, y_real, alpha = 0.5)
    plt.plot(x, y_pred, "r-", linewidth=2, label="Predictions")
```

```python
    plt.legend(loc = "upper right")
    plt.xlabel("X")
    plt.ylabel("y")
    plt.title(title)

    plt.show()

def display_metric_plots(scores):


    fig, (ax1, ax2, ax3) = plt.subplots(nrows = 3)

        #Grafica las diferentes métricas en un diagrama de barras
    ax1.set_title('Comparación MAE por modelo')
    ax1.barh(scores["names"], scores["MAE"])

    ax2.set_title('Comparación R^2 por modelo')
    ax2.barh(scores["names"], scores["R2"])

    #Grafica los residuos en un diagrama de caja
    ax3.set_title('Diagrama de caja de los residuos')
    ax3.boxplot(scores['res'], labels=scores['names'], showmeans = True)
    ax3.figure.set_figheight(15)

    plt.show()

#Obtiene los coeficientes e interceptos de los modelos
def get_linear_model_eq(model):
    return [model.coef_, model.intercept_]

#Esta función imprime los valores de las métricas calculadas
def print_metrics(y_real, y_pred):
    print('Error medio Absoluto (MAE):', mean_absolute_error(y_real, y_pred))
    print('Root Mean Squared Error:', np.sqrt(mean_squared_error(y_real,␣
 ↪y_pred)))
    print('r2_score',r2_score(y_real, y_pred))

 #Ordena la data de menor a mayor
def sort_data(X, y_real, y_pred):

    order = np.argsort(X)

    x_sorted = X[order]
    y_real_sorted = y_real[order]
    y_pred_sorted = y_pred[order]
```

```
    return x_sorted, y_real_sorted, y_pred_sorted
```

Ahora importamos los datos que estaremos utilizando

[ ]:
```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/marypazrf/bdd/main/
 ↪EconomiesOfScale.csv')
df.sample(10)
```

[ ]:
```
     Number of Units  Manufacturing Cost
142         3.063801           40.933678
950         6.727705           31.494308
122         2.943189           38.825026
642         4.977296           32.276264
803         5.554847           35.304973
35          2.004804           60.021543
17          1.739201           60.572597
835         5.664530           36.483292
69          2.544661           50.414450
96          2.762052           50.216171
```

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

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

[ ]:
```
1000
```

Ahora, revisamos las estadísticas descriptivas de los datos.

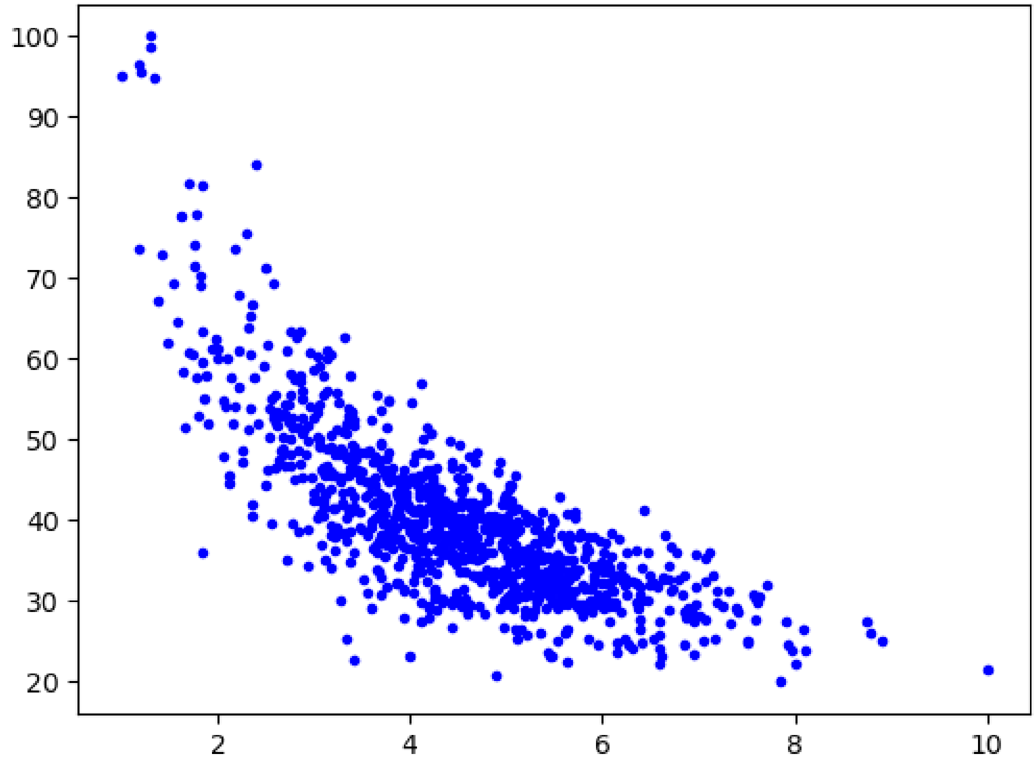
[ ]:
```
y.describe()
```

[ ]:
```
count    1000.000000
mean       40.052999
std        10.595322
min        20.000000
25%        32.912036
50%        38.345781
75%        44.531822
max       100.000000
Name: Manufacturing Cost, dtype: float64
```

De lo anterior podemos observar que los costos varían de 20 a 100 y tanto la media como la mediana tienen valores similares. Esto significa que los datos no tienen alta variación, o bien, presencia de atípicos. Ahora observemos el comportamiento de los datos con un diagrama de correlación.

[ ]:
```
plt.plot(X,y,'b.')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x17fd49030>]
```



De lo anterior podemos apreciar que los datos tienen un comportamiento ligeramente curveado, por lo que es muy probable que un polinmio de orden 2 tenga mejor desempeño que un modelo lineal. Ahora hagamos la partición de los datos.

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,␣
 ↪random_state=42)

#Aquí almacenaremos los scores
scores = {
    'MAE': [],
    'R2': [],
    'res': [],
    'names': ["Modelo Lineal", "Modelo Cuadrático", "Modelo Ridge", "Modelo␣
 ↪Lasso"]}
```

Ahora entrenaremos el modelo lineal

[ ]:
```
#Modelo lineal
lr_model = LinearRegression()

#Entrenamos el modelo y obtenemos las predicciones
lr_model.fit(X_train, y_train)
y_pred = lr_model.predict(X_test)
```

```
#Almacenamos las métricas y residuos
scores["MAE"].append(mean_absolute_error(y_test, y_pred))
scores["R2"].append(r2_score(y_test, y_pred))
scores["res"].append(y_test - y_pred)
```

Ahora obtendremos el coeficiente e intercepto para determinar la ecuación.

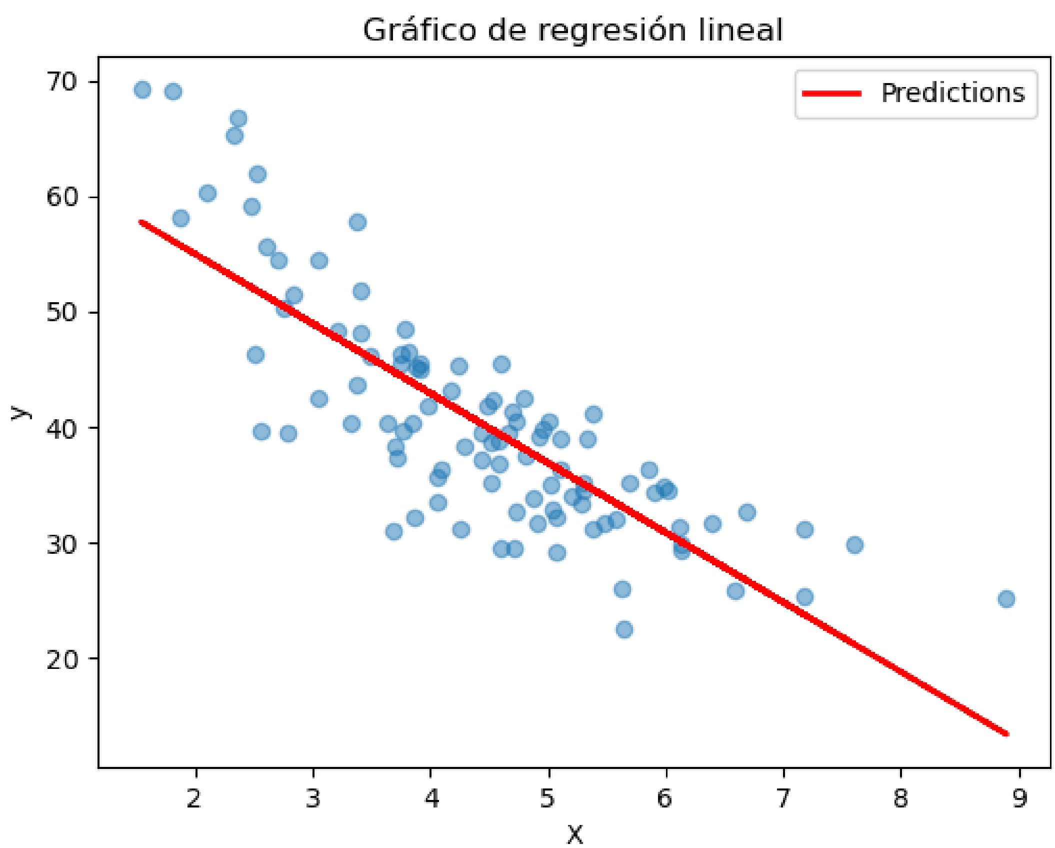
[ ]:
```
get_linear_model_eq(lr_model)
```

[ ]:
```
[array([-6.03357276]), 67.03883293539208]
```

La ecuación del modelo anterior es: $\hat{y} = 67.038 - 6.033X$

Ahora generemos el gráfico del modelo

[ ]:
```
plot_model(X_test, y_test, y_pred, "Gráfico de regresión lineal")
```



Del gráfico anterior se observa que los datos de prueba se ajustan al modelo lineal, aunque en la parte superior izquierda se pueden observar algunos puntos que se salen del patrón. Veamos ahora las métricas para este modelo.

[ ]:
```
print_metrics(y_test, y_pred)
```

```
Error medio Absoluto (MAE): 4.581575620531287
Root Mean Squared Error: 5.820691087508853
r2_score 0.6544705154382865
```

A continuación, entrenaremos un modelo de orden 2.

[ ]:
```
#polinomial
pipe = Pipeline(steps = [
    ("pol_transform", PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression())
])

#Entrenamos el modelo
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)

#Almacenamos las métricas y residuos
scores["MAE"].append(mean_absolute_error(y_test, y_pred))
scores["R2"].append(r2_score(y_test, y_pred))
scores["res"].append(y_test - y_pred)
```

Ahora obtengamos la ecuación del modelo

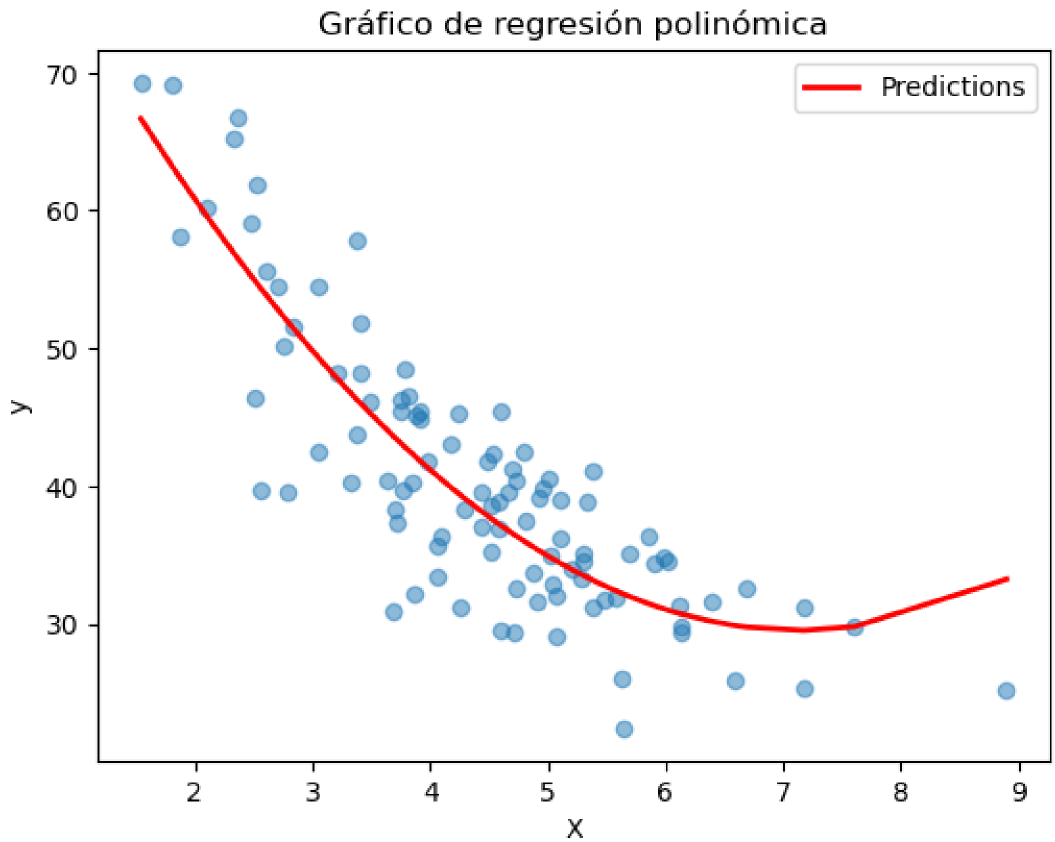
[ ]:
```
[pipe.named_steps['model'].coef_, pipe.named_steps['model'].intercept_]
```

[ ]:
```
[array([-16.95147175,   1.18852154]), 89.97388061400437]
```

La ecuación del modelo anterior es: $\hat{y} = -17.82X + 1.27X^2 + 92$

Ahora veamos su gráfico, pero primero ordenemos los datos para evitar que el gráfico se distorcione.

[ ]:
```
x_sorted, y_real_sorted, y_pred_sorted = sort_data(X_test.values.ravel(),␣
 ↪y_test.values.ravel(), y_pred)

plot_model(x_sorted, y_real_sorted, y_pred_sorted, "Gráfico de regresión␣
 ↪polinómica")
```



Del gráfico anterior podemos ver que el modelo se ajusta mejor, principalmente para los puntos de arriba que en el modelo lineal quedaban muy lejos de la curva. Procedamos a calcular las métricas para este modelo.

[ ]:
```
print_metrics(y_test, y_pred)
```

```
Error medio Absoluto (MAE): 4.070921827959767
Root Mean Squared Error: 5.101935770152972
r2_score 0.7345357864097419
```

A continuación estaremos entrenando los modelos Ridge y Lasso

[ ]:
```
#Modelo Ridge
lr_ridge = Ridge()

#Entrenamos el modelo y obtenemos las predicciones
lr_ridge.fit(X_train, y_train)
y_pred = lr_ridge.predict(X_test)

#Almacenamos las métricas y residuos
scores["MAE"].append(mean_absolute_error(y_test, y_pred))
```

```
scores["R2"].append(r2_score(y_test, y_pred))
scores["res"].append(y_test - y_pred)
```

[ ]:
```
#Obtenemos los coeficientes
get_linear_model_eq(lr_ridge)
```

[ ]: [array([-6.02982087]), 67.0220338533617]

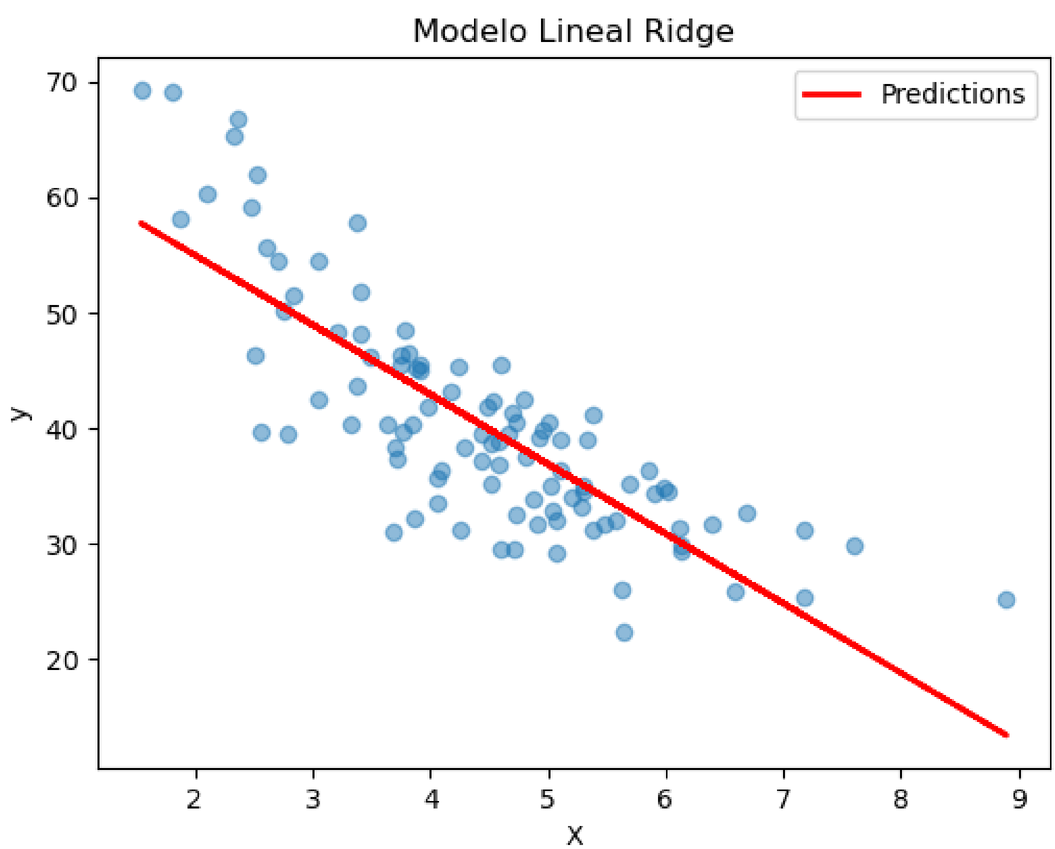
La ecuación del modelo ridge es: $\hat{y} = 67.02 - 6.03X$

Obtengamos las métricas y su gráfico.

[ ]:
```
print_metrics(y_test, y_pred)
```

```
Error medio Absoluto (MAE): 4.581736558347315
Root Mean Squared Error: 5.820690906561836
r2_score 0.6544705369211404
```

[ ]:
```
plot_model(X_test, y_test, y_pred, title="Modelo Lineal Ridge")
```



El modelo ridge tuvo un comportamiento muy similar al modelo lineal. Entrenemos ahora el modelo Lasso.

[ ]:
```
lr_lasso = Lasso()

lr_lasso.fit(X_train, y_train)
y_pred = lr_lasso.predict(X_test)

scores["MAE"].append(mean_absolute_error(y_test, y_pred))
scores["R2"].append(r2_score(y_test, y_pred))
scores["res"].append(y_test - y_pred)
```

[ ]:
```
get_linear_model_eq(lr_lasso)
```

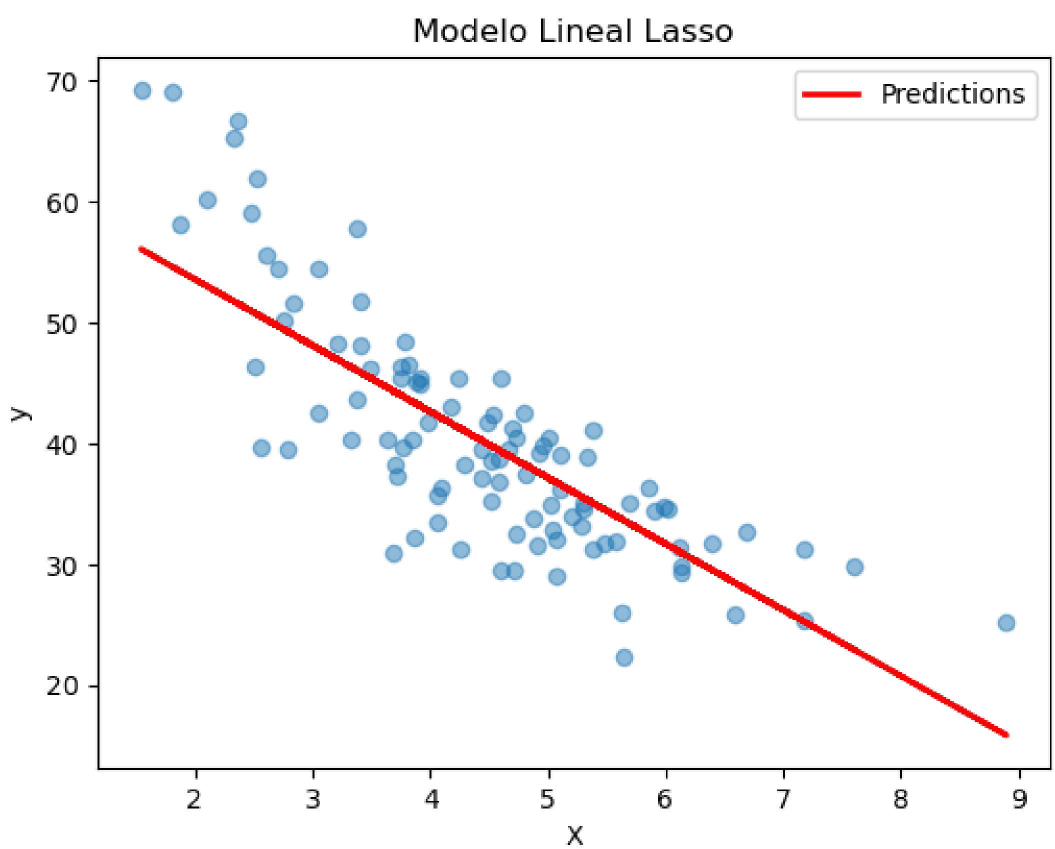
[ ]:
```
[array([-5.47357151]), 64.53143277158046]
```

La ecuación del modelo lasso es: $\hat{y} = 64.53 - 5.47X$

[ ]:
```
print_metrics(y_test, y_pred)
```

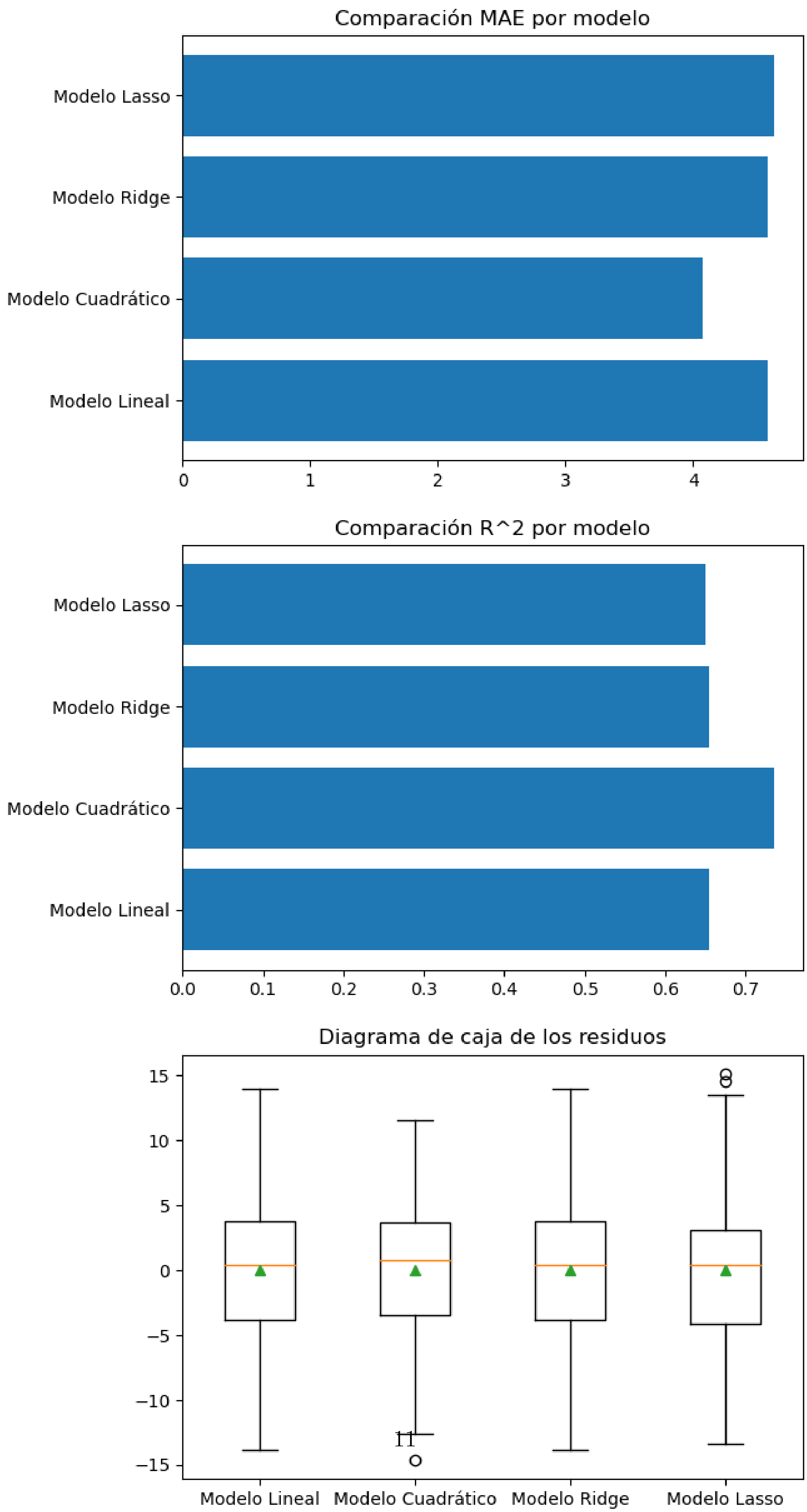
```
Error medio Absoluto (MAE): 4.630348756803918
Root Mean Squared Error: 5.867738730534317
r2_score 0.6488622307077648
```

[ ]:
```
plot_model(X_test, y_test, y_pred, title="Modelo Lineal Lasso")
```



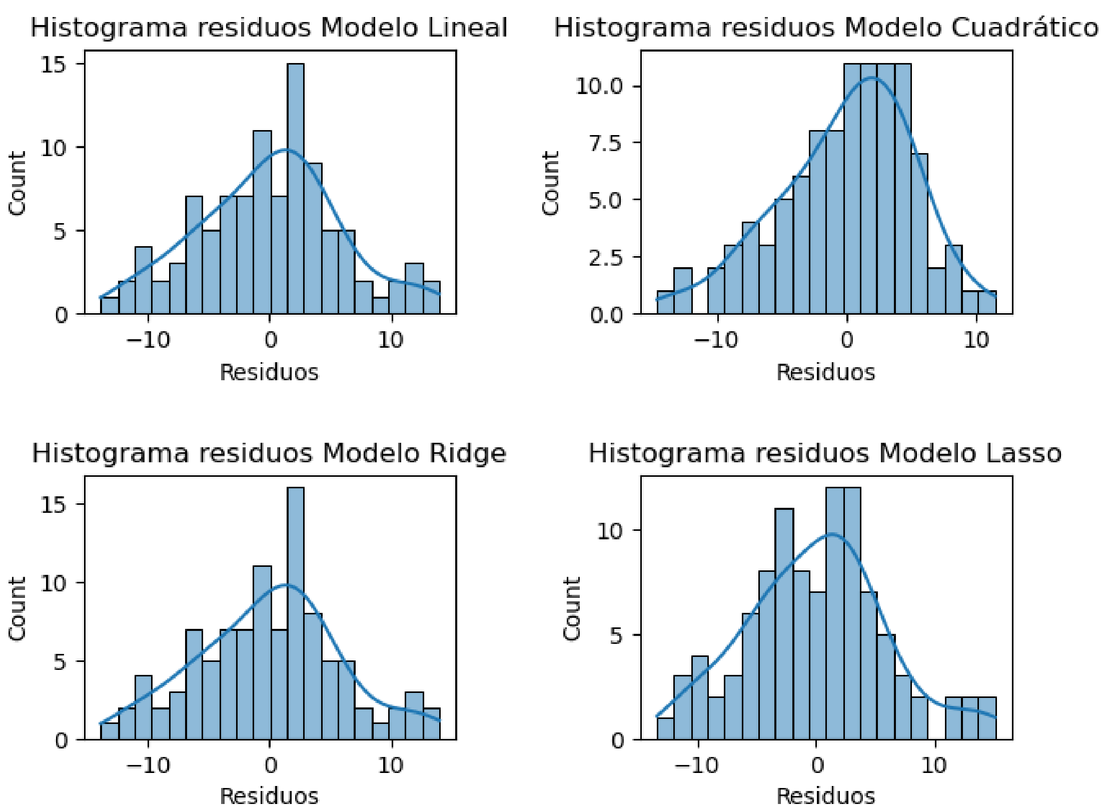
Al igual que ridge, tuvo un comportamiento similar al modelo lineal. Hagamos un comparativo de las métricas.

[ ]:
```
display_metric_plots(scores)
```


Comparación MAE por modelo
Modelo Lasso
Modelo Ridge
Modelo Cuadrático
Modelo Lineal
0
1
2
3
4
Comparación R^2 por modelo
Modelo Lasso
Modelo Ridge
Modelo Cuadrático
Modelo Lineal
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Diagrama de caja de los residuos
15
10
5
0
−5
−10
−15
11
Modelo Lineal
Modelo Cuadrático
Modelo Ridge
Modelo Lasso

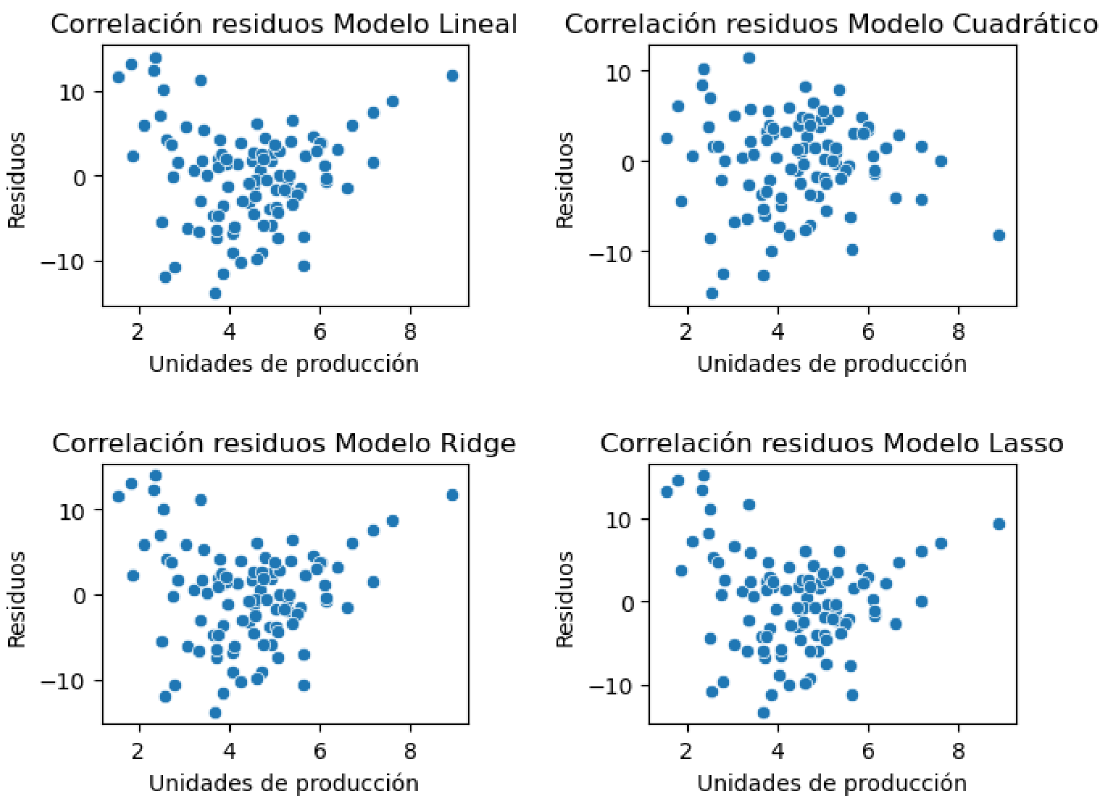
De lo anterior se observa que el modelo cuadrático fue el que tuvo mejor desempeño. Veamos el comportamiento de los residuos, viendo el histograma y su correlación.

[ ]:
```
#Hacemos un panel que tenga dos columnas y dos filas
fig, ax = plt.subplots(2,2)
#Ajustamos el layout
plt.tight_layout(h_pad=5, w_pad=5)
count = 0
#Ploteamos el gráfico en cada panel
for i in range(2):
    for j in range(2):
        ax[i][j].set_title(f'Histograma residuos {scores["names"][count]}')
        ax[i][j].set_xlabel('Residuos')
        sns.histplot(scores['res'][count], bins = 20, kde=True, ax= ax[i][j])
        count = count + 1
```



[ ]:
```
fig, ax = plt.subplots(2,2)
plt.tight_layout(h_pad=5, w_pad=5)
count = 0
```

```python
for i in range(2):
    for j in range(2):
        ax[i][j].set_title(f'Correlación residuos {scores["names"][count]}')
        ax[i][j].set_xlabel('Unidades de producción')
        ax[i][j].set_ylabel('Residuos')
        sns.scatterplot(x = X_test.values.ravel(), y = scores['res'][count], ax= ax[i][j])
        count = count + 1
```



Se observa que los residuos siguen una distribución tipo camapa y no se evidencia correlación entre ellos.

2.0.1 Conclusiones

- Se eligió una partición de los datos de entrenamiento/prueba de 80/20 respectivamente. Esto para mantener un equilibrio entre sesgo y varianza, ya que el contar con más datos de entrenamiento disminuye la varianza, pero si se tiene muy pocos datos para validación entonces el sesgo aumenta. La partición 80/20 es la más usada en la práctica.
- El modelo que mostró mejor desempeño fue el cuadrático, ya que este presenta el menor error entre todos los modelos, así como un coeficiente de determinación más alto. Por ejemplo, el modelo cuadrático explica alrededor del 73% de la variación de los datos, con un MAE de

4.03. Los demás modelos explican menos del 70% de la variación y su MAE es por encima de 4.

- Con relación a si el error aceptable o no, esto es algo que debería evaluarse con el negocio. El MAE de 4.03 significa un desfase de las predicciones realizadas por el modelo de un promedio de USD 4.03 (suponiendo que la moneda sea dólares). El negocio debe decidir si este error es tolareble o no para las predicciones que desea realizar.
- Revisando las suposiciones del modelo, se observa que todos los gráficos siguen una distribución mas o menos uniforme, tipo campana. También del diagrama de correlación no se puede apreciar patrones que indiquen una correlación de los residuos.

3 Ejercicio 2

Realiza la regresión polinomial de los siguientes datos:

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

[ ]:
```
               id             date     price  bedrooms  bathrooms  \
13620  7011201475  20140527T000000  780000.0         3       3.00
683    3438500486  20141016T000000  413000.0         4       3.50
19941  7237450600  20141030T000000  450000.0         5       2.75
3880   7774200236  20141211T000000  357000.0         3       1.50
17876  5700000245  20140602T000000  540000.0         4       1.75
642    4140090320  20150320T000000  595000.0         5       2.75
7575   2391600330  20150410T000000  505000.0         2       1.00
13605  2724079061  20141010T000000  610000.0         3       1.75
17121  5096300130  20140714T000000  413000.0         3       2.00
2925   5103300090  20140801T000000  699000.0         5       2.50

       sqft_living  sqft_lot  floors  waterfront  view  …  grade  \
13620         2520      2152     1.5           0     0  …      8
683           2380      5809     2.0           0     0  …      7
19941         2710      6220     2.0           0     0  …      8
3880          1340     11744     1.0           0     0  …      7
17876         1720      4240     1.5           0     0  …      7
642           3740      6750     1.0           0     0  …      8
7575           810      5060     1.0           0     0  …      6
13605         1650    221720     1.0           0     0  …      7
17121         1520      3451     1.0           0     0  …      8
2925          3340     24755     2.0           0     0  …     10

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
13620        1560            960      1925          2006    98119  47.6363
683          1750            630      1995             0    98106  47.5536
19941        2710              0      2014             0    98038  47.3555
```

```
3880          1340             0      1950             0    98146  47.4947
17876         1460           260      1925             0    98144  47.5790
642           1980          1760      1978             0    98028  47.7679
7575           810             0      1941             0    98116  47.5635
13605         1650             0      1992             0    98024  47.5297
17121         1520             0      1996             0    98177  47.7753
2925          3340             0      2002             0    98038  47.4565

          long  sqft_living15  sqft_lot15
13620 -122.371           1140        2152
683   -122.359           1620        5775
19941 -122.061           2530        4759
3880  -122.360           2020       13673
17876 -122.294           1930        4280
642   -122.261           2620        7920
7575  -122.394            900        5060
13605 -121.901           2520      221284
17121 -122.375           1800        3451
2925  -122.066           3420       23274

[10 rows x 21 columns]
```

Veamos un resumen de la estructura del conjunto de datos

```
[ ]: 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
```

Revisemos las estadísticas descriptiva de los datos

[ ]:
```
df.describe()
```

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

           sqft_lot        floors    waterfront          view     condition  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000  21613.000000
mean   1.510697e+04      1.494309      0.007542      0.234303      3.409430
std    4.142051e+04      0.539989      0.086517      0.766318      0.650743
min    5.200000e+02      1.000000      0.000000      0.000000      1.000000
25%    5.040000e+03      1.000000      0.000000      0.000000      3.000000
50%    7.618000e+03      1.500000      0.000000      0.000000      3.000000
75%    1.068800e+04      2.000000      0.000000      0.000000      4.000000
max    1.651359e+06      3.500000      1.000000      4.000000      5.000000

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

            zipcode           lat          long  sqft_living15    sqft_lot15
count  21613.000000  21613.000000  21613.000000   21613.000000  21613.000000
mean   98077.939805     47.560053   -122.213896    1986.552492  12768.455652
std       53.505026      0.138564      0.140828     685.391304  27304.179631
min    98001.000000     47.155900   -122.519000     399.000000    651.000000
25%    98033.000000     47.471000   -122.328000    1490.000000   5100.000000
50%    98065.000000     47.571800   -122.230000    1840.000000   7620.000000
```

75%	98118.000000	47.678000	-122.125000	2360.000000	10083.000000
max	98199.000000	47.777600	-121.315000	6210.000000	871200.000000

Eliminamos algunas columnas que no estaremos usando

[ ]:
```
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)
```

Revisamos la correlación entre cada una de las variables

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

Definimos las Xs y Ys

[ ]:
```
columns = df.columns.drop('price')

features = columns
```

```
label = ['price']

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

[ ]:
```
y.shape
```

[ ]: (21613, 1)

[ ]:
```
X.shape
```

[ ]: (21613, 15)

Generamos la partición de los datos

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1,␣
 ↪random_state = 101)
scores = {
    'MAE': [],
    'R2': [],
    'res': [],
    'names': ["Modelo Lineal", "Modelo Cuadrático", "Modelo Ridge", "Modelo␣
 ↪Lasso"]}

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)
```

[ ]:
```
X_test.head()
```

[ ]:
```
       bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  \
3834          2       1.00         1050      6317     1.5           0     0
1348          4       2.25         2040      9565     1.0           0     0
20366         4       2.50         2500      4000     2.0           0     0
16617         5       2.00         2360     19899     1.0           0     0
20925         3       3.00         1670      4440     1.0           0     0
```

```
       condition  grade  sqft_above  sqft_basement  yr_built  yr_renovated  \
3834           4      7        1050              0      1913             0
1348           3      8        1400            640      1959             0
20366          3      8        2500              0      2014             0
16617          4      7        2360              0      1968             0
20925          3      7        1670              0      2014             0

       sqft_living15  sqft_lot15
3834            1600        9616
1348            1890        8580
20366           1480        4300
16617           1860       19998
20925           1670        4622
```

Ahora entrenamos los modelos

[ ]:
```
mlr_model = LinearRegression()

mlr_model.fit(X_train, y_train)
y_pred = mlr_model.predict(X_test)

scores["MAE"].append(mean_absolute_error(y_test, y_pred))
scores["R2"].append(r2_score(y_test, y_pred))
scores["res"].append(y_test.values.flatten() - y_pred.flatten())
```

Obtenemos su ecuación

[ ]:
```
get_linear_model_eq(mlr_model)
```

[ ]:
```
[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,
          4.77750270e+01,  6.02175564e+01, -3.55090216e+03,
          1.32602215e+01,  2.90059284e+01, -5.48132603e-01]]),
 array([6151359.26274254])]
```

La ecuación del modelo lineal es: $\hat{y} = -38,200X_1 + 41,466X_2 + 107.99X_3 + ... - 0.548X_{15} + 6,151,359.26$

[ ]:
```
print_metrics(y_test, y_pred)
```

```
Error medio Absoluto (MAE): 137480.13882730895
Root Mean Squared Error: 232133.36762408607
r2_score 0.6579723205007484
```

Hacemos lo mismo para el modelo polinomial

[ ]:
```
#polinomial
pipe = Pipeline(steps = [
```

```
    ("pol_transform", PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression())
])

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)

scores["MAE"].append(mean_absolute_error(y_test, y_pred))
scores["R2"].append(r2_score(y_test, y_pred))
scores["res"].append(y_test.values.flatten() - y_pred.flatten())
```

Para sacar la ecuación, obtenemos cada uno de sus términos.

[ ]:
```
pipe.named_steps['pol_transform'].get_feature_names_out(X_train.columns)
```

[ ]:
```
array(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15',
       'sqft_lot15', 'bedrooms^2', 'bedrooms bathrooms',
       'bedrooms sqft_living', 'bedrooms sqft_lot', 'bedrooms floors',
       'bedrooms waterfront', 'bedrooms view', 'bedrooms condition',
       'bedrooms grade', 'bedrooms sqft_above', 'bedrooms sqft_basement',
       'bedrooms yr_built', 'bedrooms yr_renovated',
       'bedrooms sqft_living15', 'bedrooms sqft_lot15', 'bathrooms^2',
       'bathrooms sqft_living', 'bathrooms sqft_lot', 'bathrooms floors',
       'bathrooms waterfront', 'bathrooms view', 'bathrooms condition',
       'bathrooms grade', 'bathrooms sqft_above',
       'bathrooms sqft_basement', 'bathrooms yr_built',
       'bathrooms yr_renovated', 'bathrooms sqft_living15',
       'bathrooms sqft_lot15', 'sqft_living^2', 'sqft_living sqft_lot',
       'sqft_living floors', 'sqft_living waterfront', 'sqft_living view',
       'sqft_living condition', 'sqft_living grade',
       'sqft_living sqft_above', 'sqft_living sqft_basement',
       'sqft_living yr_built', 'sqft_living yr_renovated',
       'sqft_living sqft_living15', 'sqft_living sqft_lot15',
       'sqft_lot^2', 'sqft_lot floors', 'sqft_lot waterfront',
       'sqft_lot view', 'sqft_lot condition', 'sqft_lot grade',
       'sqft_lot sqft_above', 'sqft_lot sqft_basement',
       'sqft_lot yr_built', 'sqft_lot yr_renovated',
       'sqft_lot sqft_living15', 'sqft_lot sqft_lot15', 'floors^2',
       'floors waterfront', 'floors view', 'floors condition',
       'floors grade', 'floors sqft_above', 'floors sqft_basement',
       'floors yr_built', 'floors yr_renovated', 'floors sqft_living15',
       'floors sqft_lot15', 'waterfront^2', 'waterfront view',
       'waterfront condition', 'waterfront grade',
       'waterfront sqft_above', 'waterfront sqft_basement',
       'waterfront yr_built', 'waterfront yr_renovated',
       'waterfront sqft_living15', 'waterfront sqft_lot15', 'view^2',
```

```
'view condition', 'view grade', 'view sqft_above',
'view sqft_basement', 'view yr_built', 'view yr_renovated',
'view sqft_living15', 'view sqft_lot15', 'condition^2',
'condition grade', 'condition sqft_above',
'condition sqft_basement', 'condition yr_built',
'condition yr_renovated', 'condition sqft_living15',
'condition sqft_lot15', 'grade^2', 'grade sqft_above',
'grade sqft_basement', 'grade yr_built', 'grade yr_renovated',
'grade sqft_living15', 'grade sqft_lot15', 'sqft_above^2',
'sqft_above sqft_basement', 'sqft_above yr_built',
'sqft_above yr_renovated', 'sqft_above sqft_living15',
'sqft_above sqft_lot15', 'sqft_basement^2',
'sqft_basement yr_built', 'sqft_basement yr_renovated',
'sqft_basement sqft_living15', 'sqft_basement sqft_lot15',
'yr_built^2', 'yr_built yr_renovated', 'yr_built sqft_living15',
'yr_built sqft_lot15', 'yr_renovated^2',
'yr_renovated sqft_living15', 'yr_renovated sqft_lot15',
'sqft_living15^2', 'sqft_living15 sqft_lot15', 'sqft_lot15^2'],
dtype=object)
```

[ ]:
```
get_linear_model_eq(pipe.named_steps['model'])
```

[ ]:
```
[array([[ 9.33759792e+05, -1.09256281e+06, -4.87073481e+02,
        -2.78640773e+01, -2.07411510e+06, -3.95442052e+06,
        -2.88609605e+05,  5.58561365e+05,  1.09858962e+06,
        -1.89460489e+02, -3.51919037e+02, -8.59710910e+04,
        -2.95368812e+03,  3.99969066e+03, -3.40093880e+01,
         9.57190309e+02,  7.49943875e+03, -1.35831412e+01,
        -1.85900279e-02,  8.57794789e+03, -1.14902590e+04,
        -3.10213305e+02, -5.24211970e+03, -5.36750775e+03,
        -1.42102923e+00, -1.48183565e+01, -4.67026618e+02,
        -9.12924954e+00,  1.72777392e+01,  2.15712937e-01,
        -9.63321067e+03,  1.54490102e+01, -1.35669703e-01,
        -2.59057884e+04,  4.33659178e+04,  3.82998703e+03,
        -1.08104637e+03,  2.22965624e+04,  1.43684650e+01,
         1.27598335e+00,  5.03027728e+02, -1.70867076e+01,
        -1.72873779e+01, -5.60495383e-02,  6.38849872e+00,
         1.16959192e+00,  4.83302023e+00,  1.60302970e+02,
        -1.42698253e+01,  1.31168769e+01,  2.07220532e+01,
        -3.92516795e+00, -5.50665757e+00,  3.08098665e-01,
        -8.19396162e-02, -8.76377049e-01, -1.83617515e+01,
         3.05939466e-07,  4.40855929e-01, -9.84700231e-02,
        -9.75763222e-02,  9.17214589e-02,  1.51968044e-01,
        -1.17027786e+00, -1.17005593e+00,  1.37630147e-02,
        -1.48869367e-04,  1.76420901e-04,  1.05102663e-06,
         2.16108435e+04, -1.31323760e+05,  1.44414219e+04,
         2.02996974e+04, -4.27946442e+03, -2.82028072e+00,
```

```
          7.66348437e+00,  1.03667956e+03,  2.59300036e+00,
         -3.20408627e+01, -5.28610515e-01, -3.95442319e+06,
         -1.60508024e+04,  9.19751316e+03, -1.61637357e+05,
          1.83106392e+02, -2.28033101e+01,  4.42848011e+03,
         -2.99151807e+01,  1.70605192e+02, -8.13464849e-01,
          7.98405124e+03,  7.33990994e+03,  1.79554497e+04,
         -1.24974992e+01, -1.77355714e+00,  7.20606086e+01,
         -9.96837991e+00,  4.63373705e+00, -5.24319923e-02,
         -5.41650364e+02, -6.03804838e+03,  1.16308638e+00,
          1.19556751e+01, -3.07012074e+02, -2.01983660e+01,
          4.63197994e+01, -2.76907544e-01,  7.36649567e+03,
          9.61288017e+00,  1.11519860e+01, -5.71961781e+02,
         -1.01339271e+01, -2.27559208e+01, -5.08575987e-01,
         -2.45799620e+00, -3.30982355e+00, -6.84216574e-02,
          1.20118954e-01,  8.74302886e-01,  1.83619421e+01,
         -9.43588782e-01,  3.27052895e-02,  1.26737839e-01,
          8.79546590e-01,  1.83612352e+01,  2.30618614e+01,
          4.07056903e-01, -2.04696794e+00,  1.88926517e-02,
          1.11629263e+00,  4.71177737e-02,  2.55666375e-04,
          3.31497627e-02,  1.11705856e-04,  2.35927291e-06]]),
 array([80235192.04249534])]
```

La ecuación del modelo polinómico es: $\hat{y} = 9.34 \times 10^5 X_1 - 1.09 \times 10^6 X_2 + 4.26 \times 10^3 X_3 + ... + 1.11 \times 10^{-4} X_{14} X_{15} + 2.35 \times 10^{-6} X_{15}^2 + 80,231,985.72$

[ ]:
```
print_metrics(y_test, y_pred)
```

```
Error medio Absoluto (MAE): 121314.02630524807
Root Mean Squared Error: 186263.26544801242
r2_score 0.7797882262461182
```

Ahora entrenemos los modelos Ridge y Lasso

[ ]:
```
lr_ridge = Ridge()

lr_ridge.fit(X_train, y_train)

y_pred = lr_ridge.predict(X_test)

scores["MAE"].append(mean_absolute_error(y_test, y_pred))
scores["R2"].append(r2_score(y_test, y_pred))
scores["res"].append(y_test.values.flatten() - y_pred.flatten())
```

[ ]:
```
get_linear_model_eq(lr_ridge)
```

[ ]:
```
[array([[-3.82192265e+04,  4.14509656e+04,  1.08015168e+02,
          1.69318239e-02,  3.16891574e+04,  5.48258333e+05,
          4.14559235e+04,  2.12264347e+04,  1.19470568e+05,
          4.78054716e+01,  6.02053443e+01, -3.55050129e+03,
```

```
           1.33171890e+01,  2.89851987e+01, -5.47998512e-01]]),
  array([6150751.38368672])]
```

La ecuación del modelo ridge es: $\hat{y} = -38,219X_1 + 41,450X_2 + 1,080X_3 + ... - 0.548X_{15} + 6,150,751.38$

```
[ ]: print_metrics(y_test, y_pred)
```

```
Error medio Absoluto (MAE): 137491.04339403284
Root Mean Squared Error: 232165.24266477523
r2_score 0.657878384029501
```

```
[ ]: lr_lasso = Lasso(max_iter=1000, tol=0.1 )
     lr_lasso.fit(X_train, y_train)

     y_pred = lr_lasso.predict(X_test)

     y_pred

     scores["MAE"].append(mean_absolute_error(y_test, y_pred))
     scores["R2"].append(r2_score(y_test, y_pred))
     scores["res"].append(y_test.values.flatten() - y_pred.flatten())
```

```
[ ]: get_linear_model_eq(lr_lasso)
```

```
[ ]: [array([-3.81995388e+04,  4.14618764e+04,  2.93483794e+02,  1.71281550e-02,
              3.16871401e+04,  5.52541107e+05,  4.12549653e+04,  2.12193878e+04,
              1.19491830e+05, -1.37711515e+02, -1.25271769e+02, -3.55085021e+03,
              1.32628425e+01,  2.90053345e+01, -5.48136040e-01]),
      array([6151280.47002667])]
```

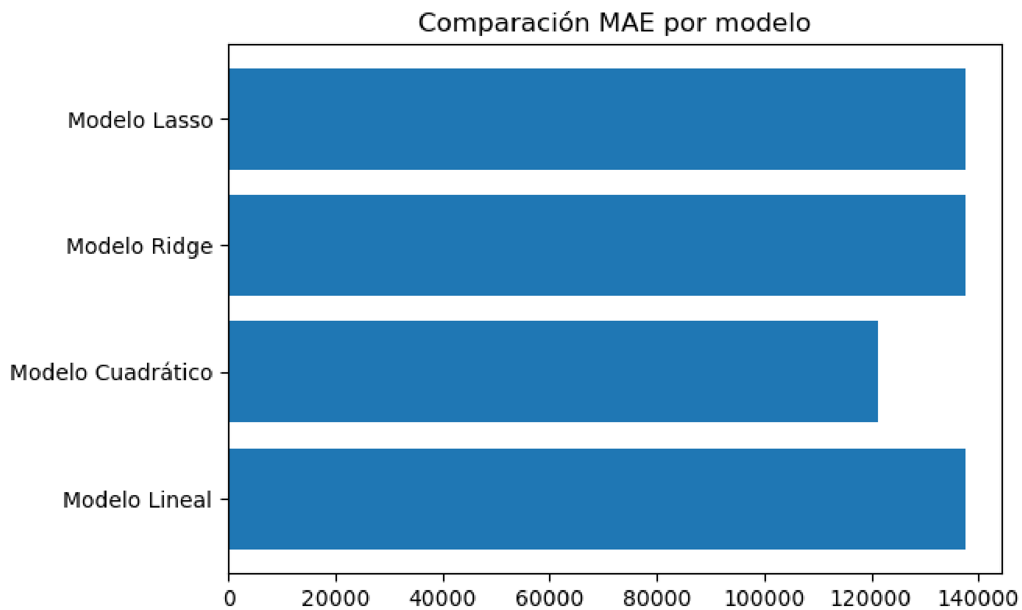
La ecuación del modelo lasso es: $\hat{y} = -38,199X_1 + 41,4618X_2 + 2,934X_3 + ... - 0.548X_{15} + 6,151,280.47$

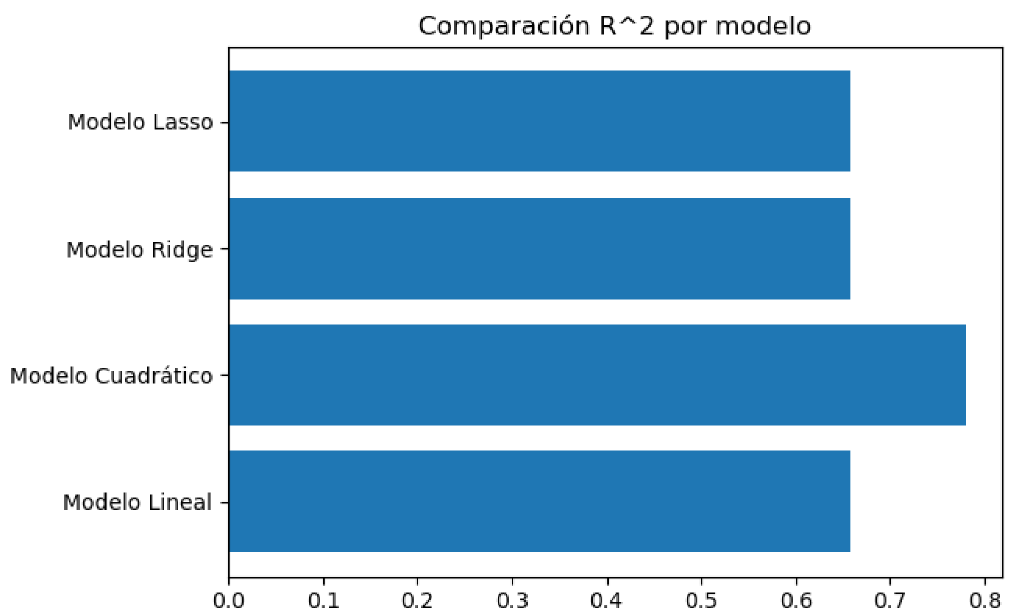
```
[ ]: print_metrics(y_test, y_pred)
```

```
Error medio Absoluto (MAE): 137480.57126997688
Root Mean Squared Error: 232134.52702878078
r2_score 0.6579689039347314
```

Comparemos las métricas de los modelos

```
[ ]: #scores['res']
     display_metric_plots(scores)
```


Comparación MAE por modelo
Modelo Lasso
Modelo Ridge
Modelo Cuadrático
Modelo Lineal
0
20000
40000
60000
80000
100000
120000
140000


Comparación R^2 por modelo
Modelo Lasso
Modelo Ridge
Modelo Cuadrático
Modelo Lineal
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

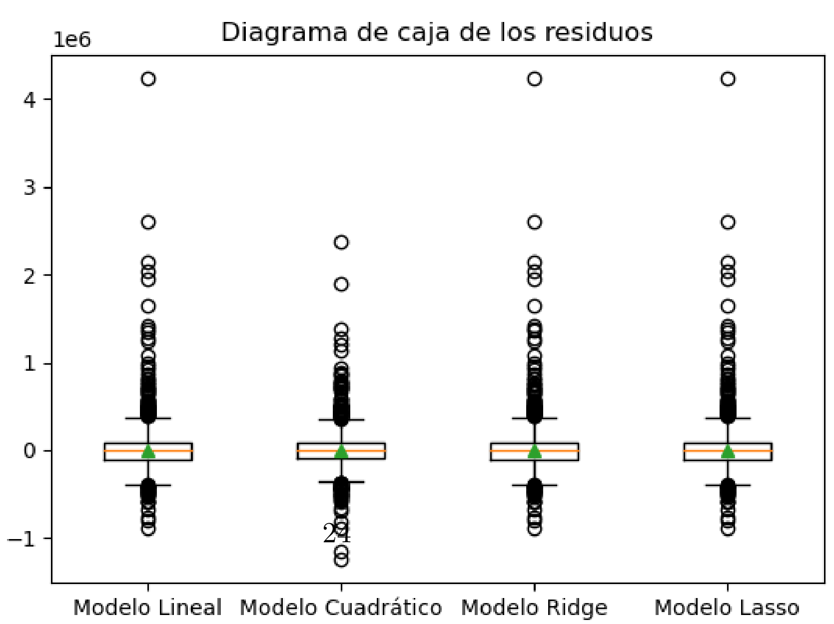
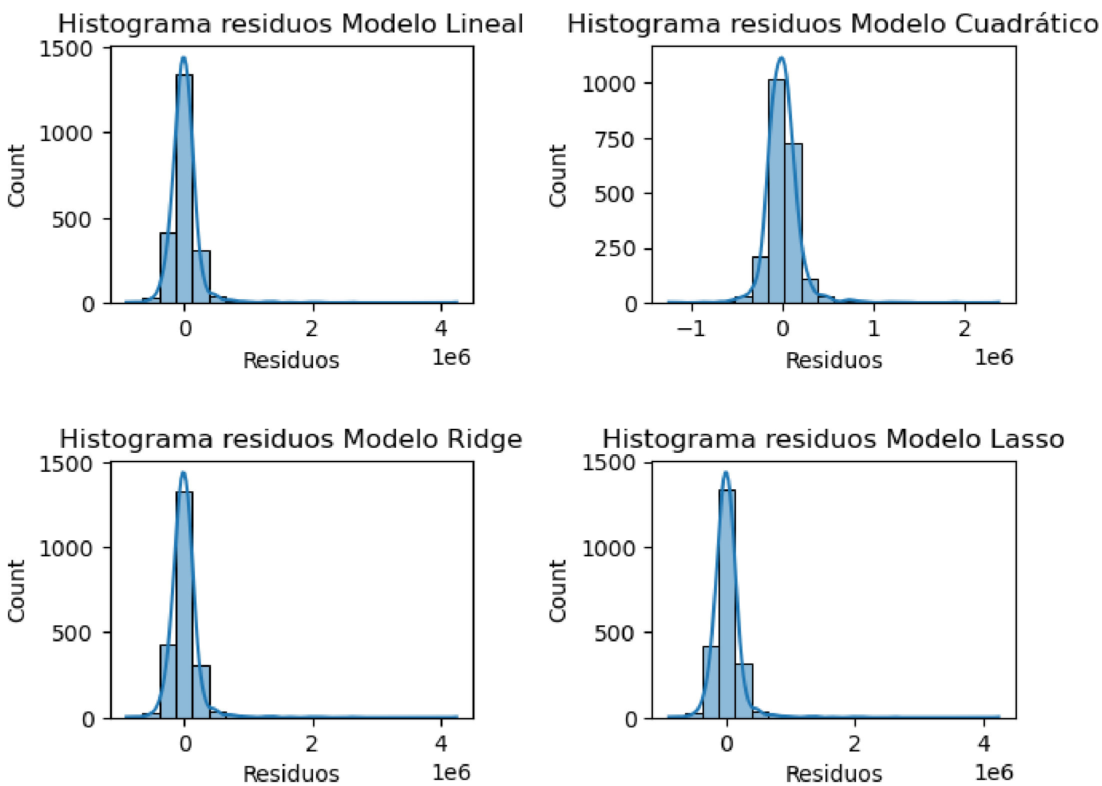


Diagrama de caja de los residuos
1e6
4
3
2
1
0
−1
Modelo Lineal
Modelo Cuadrático
Modelo Ridge
Modelo Lasso

Al igual que en el ejercicio 1, el modelo cuadrático tuvo mejor desempeño. También se observa la presencia de atípicos en los residuos. Exploremos ahora la distribución de estos residuos.

```
[ ]: fig, ax = plt.subplots(2,2)
     plt.tight_layout(h_pad=5, w_pad=5)
     count = 0
     for i in range(2):
         for j in range(2):
             ax[i][j].set_title(f'Histograma residuos {scores["names"][count]}')
             ax[i][j].set_xlabel('Residuos')
             sns.histplot(scores['res'][count], bins = 20, kde=True, ax= ax[i][j])
             count = count + 1
```



Se observa una distribución leptocúrtica sesgada a la derecha, esto debido a los atípicos que se visualizaron en el diagrama de caja.

3.0.1 Conclusiones

- Se eligió una partición de los datos de entrenamiento/prueba de 90/10 respectivamente. Esto para mantener un equilibrio entre sesgo y varianza, ya que el contar con más datos de entrenamiento disminuye la varianza, pero si se tiene muy pocos datos para validación entonces el sesgo aumenta.

- El modelo que mostró mejor desempeño fue el cuadrático, ya que este presenta el menor error entre todos los modelos, así como un coeficiente de determinación más alto. Por ejemplo, el modelo cuadrático explica alrededor del 80% de la variación de los datos, con un MAE de 1,200,000.00. Los demás modelos explican menos del 70% de la variación y su MAE de la cifra anterior.
- Aunque la tolerancia al error dependerá del negocio, se puede observar a primera vista que una diferencia de cerca de 1,200,000.00 entre el precio real y las predicciones es bastante alta. También, esta conclusión puede complementarse al analizar los residuos del modelo. Por ejemplo, en el diagrama de caja se evidencia la presencia de valores atípicos en todos los modelos. También en el histograma se observa que estos valores atípicos están sesgando el modelo hacia la derecha.
- Lo anterior puede deberse a que no se aplicó ningún tipo de escalamiento a las variables de entrada. Por ejemplo, se tienen variables como número de habitaciones con rangos que van de 1 a 30 y variables como pies cuadrado que tienen un rango mucho mayor.
- Otro punto que se observa es la presencia de variables dependientes correlacionadas, como el número de baños y los pies cuadrados del inmueble. Es importante dar tratamiento a estas variables correlacionadas antes de elegir un modelo. Por ejemplo, podrían reducirse las dimensiones por medio de un PCA.
- Por tanto, se sugiere aplicar primero un escalamiento de los datos, así como un análisis exploratorio para identificar atípicos y otros patrones que deban ser tomados en cuenta antes de entrenar el modelo.

Ejercicio 3 - KNN Means

TecMty_kmeans_target

November 7, 2022

0.0.1 Actividad Semanal #6

- Nombre: Rafael J. Mateo C
- Matrícula: A01793054
- Materia: Ciencia y Analítica de Datos
- Profesor: María de la Paz
- Fecha: 7 Nov 2022

Este notebook se basa en información de target

Ahora imagina que somos parte del equipo de data science de la empresa Target, una de las tiendas con mayor presencia en Estados Unidos. El departamento de logistica acude a nosotros para saber donde le conviene poner sus almacenes, para que se optimice el gasto de gasolina, los tiempos de entrega de los productos y se disminuyan costos. Para ello, nos pasan los datos de latitud y longitud de cada una de las tiendas.

https://www.kaggle.com/datasets/saejinmahlauheinert/target-store-locations?select=target-locations.csv

Si quieres saber un poco más de graficas geográficas consulta el siguiente notebook https://colab.research.google.com/github/QuantEcon/quantecon-notebooks-datascience/blob/master/applications/maps.ipynb#scrollTo=uo2oPtSCeAOz

```
[ ]: #! pip install qeds fiona geopandas xgboost gensim folium pyLDAvis descartes
```

```
[ ]: import pandas as pd
     import numpy as np
     from tqdm import tqdm
     %matplotlib inline
     import numpy as np
     import matplotlib.pyplot as plt
     import geopandas
```

Importa la base de datos

```
[ ]: url="https://raw.githubusercontent.com/marypazrf/bdd/main/target-locations.csv"
     df=pd.read_csv(url)
```

Exploremos los datos.

```
[ ]: df.head()
```

[ ]:
```
        name   latitude  longitude  \
0  Alabaster  33.224225 -86.804174
1   Bessemer  33.334550 -86.989778
2     Daphne  30.602875 -87.895932
3    Decatur  34.560148 -86.971559
4     Dothan  31.266061 -85.446422

                                          address         phone  \
0       250 S Colonial Dr, Alabaster, AL 35007-4657  205-564-2608
1      4889 Promenade Pkwy, Bessemer, AL 35022-7305  205-565-3760
2         1698 US Highway 98, Daphne, AL 36526-4252  251-621-3540
3  1235 Point Mallard Pkwy SE, Decatur, AL 35601-…  256-898-3036
4        4601 Montgomery Hwy, Dothan, AL 36303-1522  334-340-1112

                                  website
0  https://www.target.com/sl/alabaster/2276
1   https://www.target.com/sl/bessemer/2375
2     https://www.target.com/sl/daphne/1274
3    https://www.target.com/sl/decatur/2084
4     https://www.target.com/sl/dothan/1468
```

[ ]:
```
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
```

Definición de Latitud y Longitud

Latitud Es la distancia en grados, minutos y segundos que hay con respecto al paralelo principal, que es el ecuador (0º). La latitud puede ser norte y sur.

Longitud: Es la distancia en grados, minutos y segundos que hay con respecto al meridiano principal, que es el meridiano de Greenwich (0º).La longitud puede ser este y oeste.

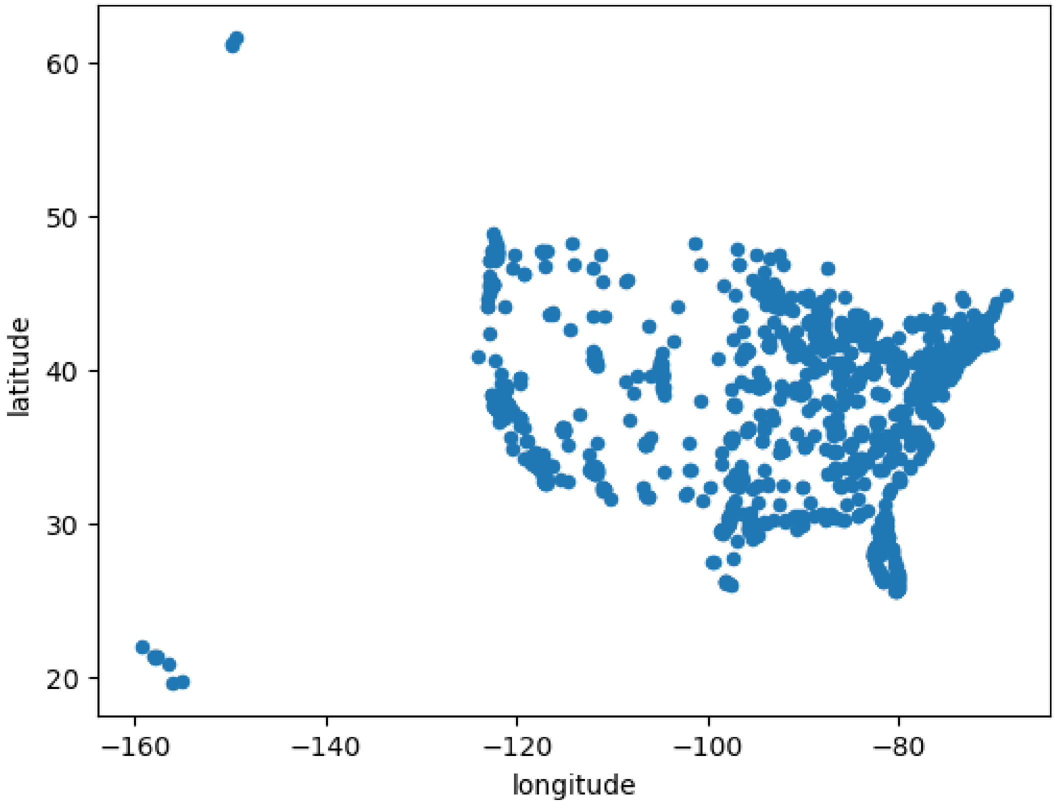
[ ]:
```
latlong=df[["latitude","longitude"]].copy()
```

¡Visualizemos los datos!, para empezar a notar algún patron.

A simple vista pudieramos pensar que tenemos algunos datos atípicos u outliers, pero …. no es así, simplemente esta grafica no nos está dando toda la información.

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

[ ]: <AxesSubplot: xlabel='longitude', ylabel='latitude'>



[ ]:
```
latlong.describe()
```

[ ]:

	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

Para entender un poco más, nos auxiliaremos de una librería para graficar datos geográficos. Esto nos ayudara a tener un mejor entendimiento de ellos.

[ ]:
```python
import geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd

from shapely.geometry import Point

%matplotlib inline
# activate plot theme
# import qeds
# qeds.themes.mpl_style();
```

[ ]:
```python
df["Coordinates"] = list(zip(df.longitude, df.latitude))
df["Coordinates"] = df["Coordinates"].apply(Point)
df.head()
```

[ ]:
```
        name   latitude  longitude  \
0  Alabaster  33.224225 -86.804174
1   Bessemer  33.334550 -86.989778
2     Daphne  30.602875 -87.895932
3    Decatur  34.560148 -86.971559
4     Dothan  31.266061 -85.446422

                                            address         phone  \
0       250 S Colonial Dr, Alabaster, AL 35007-4657  205-564-2608
1      4889 Promenade Pkwy, Bessemer, AL 35022-7305  205-565-3760
2         1698 US Highway 98, Daphne, AL 36526-4252  251-621-3540
3  1235 Point Mallard Pkwy SE, Decatur, AL 35601-…  256-898-3036
4        4601 Montgomery Hwy, Dothan, AL 36303-1522  334-340-1112

                                   website  \
0  https://www.target.com/sl/alabaster/2276
1   https://www.target.com/sl/bessemer/2375
2     https://www.target.com/sl/daphne/1274
3    https://www.target.com/sl/decatur/2084
4     https://www.target.com/sl/dothan/1468

                               Coordinates
0  POINT (-86.80417369999999 33.2242254)
1  POINT (-86.98977789999999 33.3345501)
2  POINT (-87.89593169999999 30.6028747)
3         POINT (-86.9715595 34.5601477)
4         POINT (-85.4464222 31.2660613)
```

[ ]:
```python
gdf = gpd.GeoDataFrame(df, geometry="Coordinates")
gdf.head()
```

```
[ ]:         name   latitude  longitude  \
     0  Alabaster  33.224225 -86.804174
     1   Bessemer  33.334550 -86.989778
     2     Daphne  30.602875 -87.895932
     3    Decatur  34.560148 -86.971559
     4     Dothan  31.266061 -85.446422

                                               address         phone  \
     0        250 S Colonial Dr, Alabaster, AL 35007-4657  205-564-2608
     1       4889 Promenade Pkwy, Bessemer, AL 35022-7305  205-565-3760
     2          1698 US Highway 98, Daphne, AL 36526-4252  251-621-3540
     3  1235 Point Mallard Pkwy SE, Decatur, AL 35601-…  256-898-3036
     4         4601 Montgomery Hwy, Dothan, AL 36303-1522  334-340-1112

                                         website                Coordinates
     0  https://www.target.com/sl/alabaster/2276  POINT (-86.80417 33.22423)
     1   https://www.target.com/sl/bessemer/2375  POINT (-86.98978 33.33455)
     2     https://www.target.com/sl/daphne/1274  POINT (-87.89593 30.60287)
     3    https://www.target.com/sl/decatur/2084  POINT (-86.97156 34.56015)
     4     https://www.target.com/sl/dothan/1468  POINT (-85.44642 31.26606)
```

```
[ ]: #mapa

     world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
     world = world.set_index("iso_a3")

     world.head()
```

```
[ ]:             pop_est      continent                      name  gdp_md_est  \
     iso_a3
     FJI        889953.0        Oceania                      Fiji        5496
     TZA      58005463.0         Africa                  Tanzania       63177
     ESH        603253.0         Africa                 W. Sahara         907
     CAN      37589262.0  North America                    Canada     1736425
     USA     328239523.0  North America  United States of America    21433226

                                                      geometry
     iso_a3
     FJI     MULTIPOLYGON (((180.00000 -16.06713, 180.00000…
     TZA     POLYGON ((33.90371 -0.95000, 34.07262 -1.05982…
     ESH     POLYGON ((-8.66559 27.65643, -8.66512 27.58948…
     CAN     MULTIPOLYGON (((-122.84000 49.00000, -122.9742…
     USA     MULTIPOLYGON (((-122.84000 49.00000, -120.0000…
```

```
[ ]: #graficar el mapa
     world.name.unique()
```

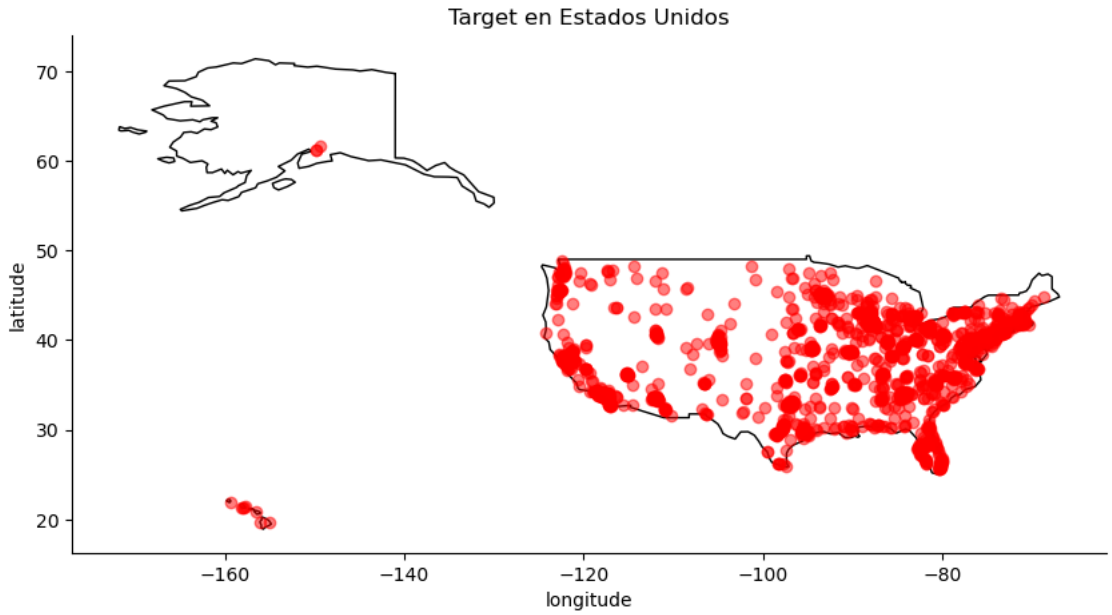
```
[ ]: array(['Fiji', 'Tanzania', 'W. Sahara', 'Canada',
            'United States of America', 'Kazakhstan', 'Uzbekistan',
            'Papua New Guinea', 'Indonesia', 'Argentina', 'Chile',
            'Dem. Rep. Congo', 'Somalia', 'Kenya', 'Sudan', 'Chad', 'Haiti',
            'Dominican Rep.', 'Russia', 'Bahamas', 'Falkland Is.', 'Norway',
            'Greenland', 'Fr. S. Antarctic Lands', 'Timor-Leste',
            'South Africa', 'Lesotho', 'Mexico', 'Uruguay', 'Brazil',
            'Bolivia', 'Peru', 'Colombia', 'Panama', 'Costa Rica', 'Nicaragua',
            'Honduras', 'El Salvador', 'Guatemala', 'Belize', 'Venezuela',
            'Guyana', 'Suriname', 'France', 'Ecuador', 'Puerto Rico',
            'Jamaica', 'Cuba', 'Zimbabwe', 'Botswana', 'Namibia', 'Senegal',
            'Mali', 'Mauritania', 'Benin', 'Niger', 'Nigeria', 'Cameroon',
            'Togo', 'Ghana', "Côte d'Ivoire", 'Guinea', 'Guinea-Bissau',
            'Liberia', 'Sierra Leone', 'Burkina Faso', 'Central African Rep.',
            'Congo', 'Gabon', 'Eq. Guinea', 'Zambia', 'Malawi', 'Mozambique',
            'eSwatini', 'Angola', 'Burundi', 'Israel', 'Lebanon', 'Madagascar',
            'Palestine', 'Gambia', 'Tunisia', 'Algeria', 'Jordan',
            'United Arab Emirates', 'Qatar', 'Kuwait', 'Iraq', 'Oman',
            'Vanuatu', 'Cambodia', 'Thailand', 'Laos', 'Myanmar', 'Vietnam',
            'North Korea', 'South Korea', 'Mongolia', 'India', 'Bangladesh',
            'Bhutan', 'Nepal', 'Pakistan', 'Afghanistan', 'Tajikistan',
            'Kyrgyzstan', 'Turkmenistan', 'Iran', 'Syria', 'Armenia', 'Sweden',
            'Belarus', 'Ukraine', 'Poland', 'Austria', 'Hungary', 'Moldova',
            'Romania', 'Lithuania', 'Latvia', 'Estonia', 'Germany', 'Bulgaria',
            'Greece', 'Turkey', 'Albania', 'Croatia', 'Switzerland',
            'Luxembourg', 'Belgium', 'Netherlands', 'Portugal', 'Spain',
            'Ireland', 'New Caledonia', 'Solomon Is.', 'New Zealand',
            'Australia', 'Sri Lanka', 'China', 'Taiwan', 'Italy', 'Denmark',
            'United Kingdom', 'Iceland', 'Azerbaijan', 'Georgia',
            'Philippines', 'Malaysia', 'Brunei', 'Slovenia', 'Finland',
            'Slovakia', 'Czechia', 'Eritrea', 'Japan', 'Paraguay', 'Yemen',
            'Saudi Arabia', 'Antarctica', 'N. Cyprus', 'Cyprus', 'Morocco',
            'Egypt', 'Libya', 'Ethiopia', 'Djibouti', 'Somaliland', 'Uganda',
            'Rwanda', 'Bosnia and Herz.', 'North Macedonia', 'Serbia',
            'Montenegro', 'Kosovo', 'Trinidad and Tobago', 'S. Sudan'],
           dtype=object)
```

```
[ ]: fig, gax = plt.subplots(figsize=(10,10))

     # By only plotting rows in which the continent is 'South America' we only plot␣
      ↪SA.
     world.query("name == 'United States of America'").plot(ax=gax,␣
      ↪edgecolor='black',color='white')

     # By the way, if you haven't read the book 'longitude' by Dava Sobel, you␣
      ↪should...
     gax.set_xlabel('longitude')
```

```
gax.set_ylabel('latitude')

gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)
```


```
[ ]: # Step 3: Plot the cities onto the map
# We mostly use the code from before --- we still want the country borders␣
 ↪plotted --- and we
# add a command to plot the cities
fig, gax = plt.subplots(figsize=(10,10))

# By only plotting rows in which the continent is 'South America' we only plot,␣
 ↪well,
# South America.
world.query("name == 'United States of America'").plot(ax = gax,␣
 ↪edgecolor='black', color='white')

# This plot the cities. It's the same syntax, but we are plotting from a␣
 ↪different GeoDataFrame.
# I want the cities as pale red dots.
gdf.plot(ax=gax, color='red', alpha = 0.5)
```

```
gax.set_xlabel('longitude')
gax.set_ylabel('latitude')
gax.set_title('Target en Estados Unidos')

gax.spines['top'].set_visible(False)
gax.spines['right'].set_visible(False)

plt.show()
```



¿qué tal ahora?, tiene mayor sentido verdad, entonces los datos lejanos no eran atípicos, de aquí la importancia de ver los datos con el tipo de gráfica correcta.

Ahora sí, implementa K means a los datos de latitud y longitud :) y encuentra donde colocar los almacenes.

Nota: si te llama la atención implementar alguna otra visualización con otra librería, lo puedes hacer, no hay restricciones.

0.0.2 Búsqueda del número óptimo de almacenes

Ahora implementemos el kmeans para buscar el número óptimo de almacenes. Empecemos primero entrenando varios modelos de kmeans con diferentes números de clústeres.

[ ]:
```
from sklearn.cluster import KMeans

#Probemos con hasta 15 clústeres
K = range(1,15)
```

```
#Almacenaremos los errores cuadráticos de cada clúster
wss = []

for k in K:
    kmeans = KMeans(n_clusters=k, init='k-means++')

    #Entrenamos el modelo
    kmeans = kmeans.fit(latlong)
    #Almacenamos la inercia, o bien, la suma de los cuadrados del cluster
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)
```

Ahora obtengamos los centros de cada clúster y pongámoslo dentro de un dataframe

[ ]:
```
centers = pd.DataFrame({'Clusters': K, 'WSS': wss})
centers
```

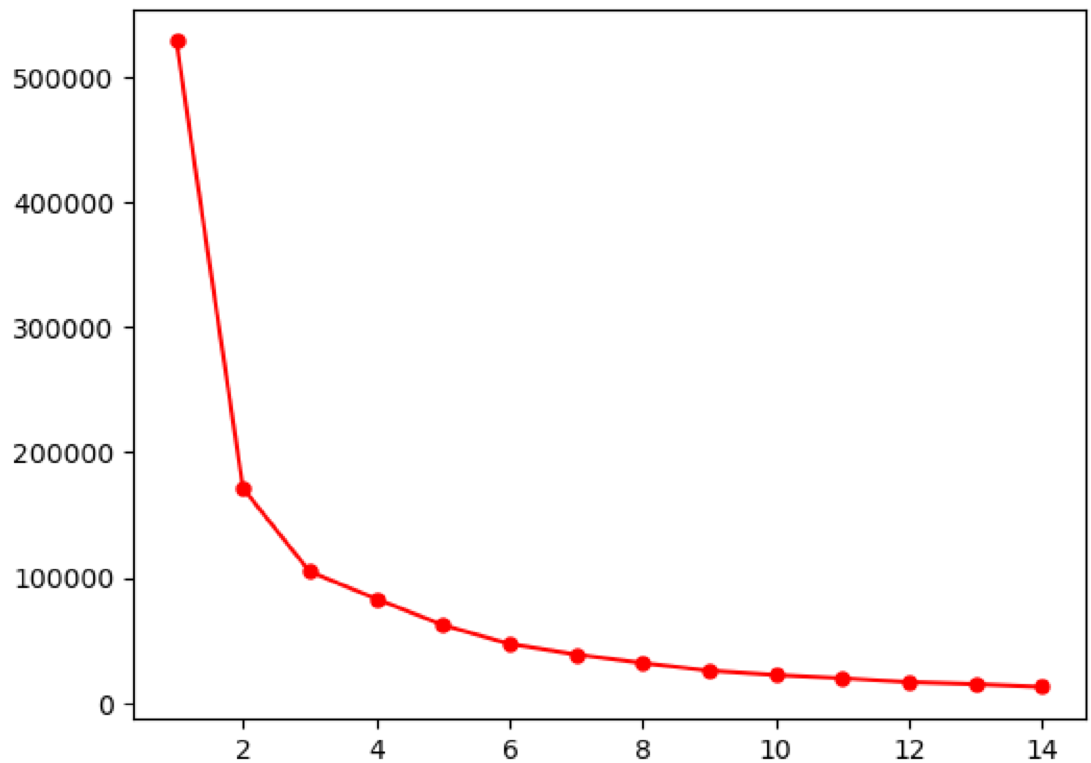
[ ]:

	Clusters	WSS
0	1	527995.443069
1	2	171146.625996
2	3	104758.597585
3	4	83021.705437
4	5	62083.344662
5	6	47006.582670
6	7	38624.131631
7	8	31765.131456
8	9	25653.227441
9	10	22225.319374
10	11	19582.906100
11	12	16472.976761
12	13	14844.783292
13	14	12687.520396

El próximo paso consiste en buscar el número óptimo de clústeres. Para esto usaremos dos métodos: El método del codo y el método de las siluetas. Comencemos primero con el método del codo.

[ ]:
```
import seaborn as sns

#Graficamos cada clústeres con sus sumas cuadráticas
plt.plot(centers.Clusters, centers.WSS, 'ro-', markersize = 5)
```

[ ]: [<matplotlib.lines.Line2D at 0x190875660>]



El método del codo consiste en ubicar el número donde se forma un codo en el gráfico. Este número sería nuestro valor óptimo de clústeres. Para este caso se observa que el valor óptimo es de 3 clústeres.

Ahora apliquemos el método de las siluetas.

```
[ ]: import sklearn.metrics as metrics

     #Empezamos a partir del segundo clúster
     for i in range(2,15):
         #Entrenamos nuevamente el modelo para obtener el score de cada uno de ellos
         labels=KMeans(n_clusters=i,init="k-means++",random_state=200).fit(latlong).
      ↪labels_
         #Imprimimos las métricas
         print ("Silhouette score for k(clusters) = "+str(i)+" is "
                +str(
                    metrics.silhouette_score(
                        latlong,labels,
                        metric="euclidean",
                        sample_size=1000,random_state=200)))
```

```
Silhouette score for k(clusters) = 2 is 0.6299987649622539
```

```
Silhouette score for k(clusters) = 3 is 0.47114786819856613
Silhouette score for k(clusters) = 4 is 0.4378026408108228
Silhouette score for k(clusters) = 5 is 0.5192796982298434
Silhouette score for k(clusters) = 6 is 0.5323590651619174
Silhouette score for k(clusters) = 7 is 0.5075239608223188
Silhouette score for k(clusters) = 8 is 0.5362463678889792
Silhouette score for k(clusters) = 9 is 0.5401582573037924
Silhouette score for k(clusters) = 10 is 0.5327073478889603
Silhouette score for k(clusters) = 11 is 0.5119847146492725
Silhouette score for k(clusters) = 12 is 0.5114520369541952
Silhouette score for k(clusters) = 13 is 0.49627113508779314
Silhouette score for k(clusters) = 14 is 0.5198387801039602
```

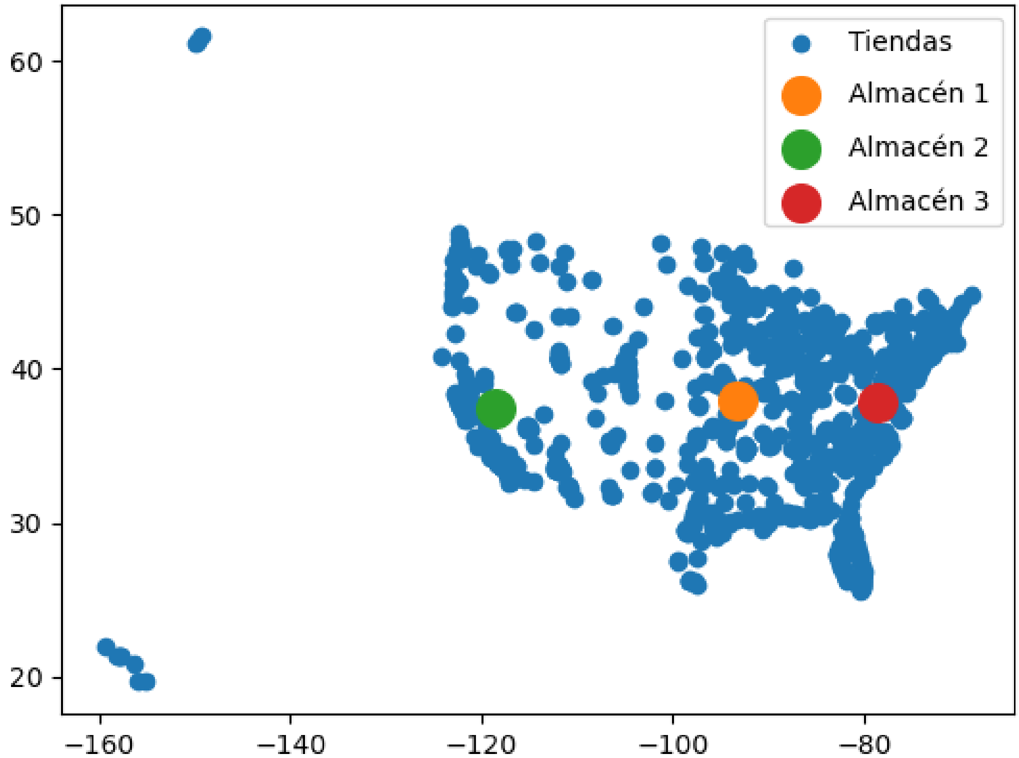
El resultado anterior nos muestra que el valor óptimo es de dos clústeres, ya que este es el que tiene el score más alto. Esto significa que podemos elegir entre 2 y 3 almacenes. Para este caso vamos a elegir 3 almacenes para minimizar la cantidad de tiendas que estos almacenes les darán respuestas.

[ ]:
```
#Entrenamos el modelo con 3 clústeres y obtenemos los centros
kmeans = KMeans(n_clusters=3, init='k-means++')
kmeans = kmeans.fit(latlong)
kmeans.cluster_centers_
```

[ ]:
```
array([[  37.98006261,  -93.3271723 ],
       [  37.48734203, -118.62447332],
       [  37.789554  ,  -78.56990807]])
```

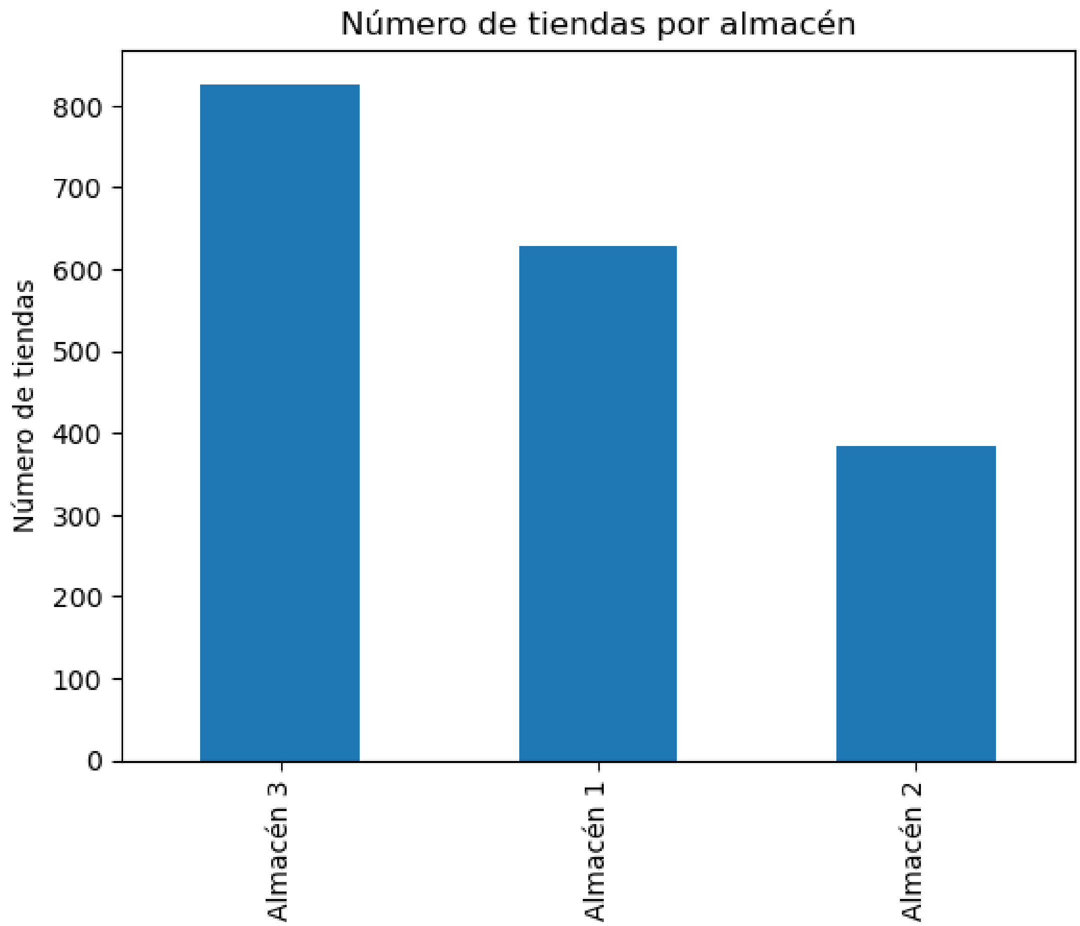
Ahora generemos un gráfico para visualizar donde ubicaríamos los almacenes

[ ]:
```
#Obtenemos los centros
centers = kmeans.cluster_centers_
#Obtenemos los clústeres y las etiquetas
clusters = np.unique(kmeans.labels_)
labels = ["Tiendas","Almacén 1", "Almacén 2", "Almacén 3"]

#Mapa de las tiendas
plt.scatter(latlong.longitude, latlong.latitude)

for i,v in enumerate(clusters):
    #Gráfico de los almacenes
    plt.scatter(centers[i][1], centers[i][0], label = i, s=200)


plt.legend(loc = 'upper right', labelspacing = 1,labels = labels)
plt.show()
```



Como se muestra en el gráfico, los almacenes estarían ubicados en la costa este, midwest y la costa oeste. Ahora veamos la cantidad de tiendas por almacén.

[ ]:
```
#Nombre de cada clúster
cluster_map = {
    0: 'Almacén 1',
    1: 'Almacén 2',
    2: 'Almacén 3'
}

#Asignamos los clústeres a cada tienda
latlong['kmeans'] = kmeans.labels_
latlong.kmeans = latlong['kmeans'].apply(lambda cluster: cluster_map[cluster])

#Generamos el gráfico
plt.title('Número de tiendas por almacén')
plt.ylabel('Número de tiendas')
latlong.kmeans.value_counts().plot(kind='bar')

plt.show()
```



Como se observa del gráfico anterior, cada almacén estará dando servicio entre 400 y 800 tiendas aproximadamente. Ahora veamos la distancia promedio de cada tienda a sus almacenes.

[ ]:
```
#Distancia promedio entre los almacenes y las tiendas
np.sqrt(kmeans.inertia_)
```

[ ]: 323.6624924486999

Conclusiones

- Se observa que el valor óptimo de almacenes es 3, los cuales estarían ubicados en la costa este, oeste y el midwest. También se puede observar que cada almacén estaría dando servicio a un total de 400 a 800 tiendas. El valor óptimo de los almacenes fue obtenido por medio del método del codo y análisis de siluetas.
- De acuerdo a las coordinadas, estas serían las localides de los almacenes (esto fue realizado en una búsqueda de google maps)

Almacén	Ciudad	Coordenadas
1	Hermitage, Missouri	37.9827023,-93.34747643
2	Round Valley, California	37.4817419,-118.657146
3	Scottsville, Virginia	37.789554,-78.56990807

- La distancia promedio de cada tienda a sus almacenes es de unos 323.66 Kms.
- Es importante entender las limitaciones de este estudio. Si bien el número óptimo de clústeres es 3, en la realidad se deben considerar otros factores además de la distancia. Por ejemplo, capacidad del almacén, demanda de las zonas, salarios, costo de construcción, valor del solar, etc. Todo estos son factores a tomar en cuenta para decidir la ubicación de los almacenes.
- El modelo Kmeans funciona bajo el supuesto de que los clústeres son isotrópicos (uniformes en todas las direcciones, es decir, esféricas) y convexas. Como se observa en el gráfico de correlación, el primer supuesto no se cumple, ya que los clústeres tienen formas muy diferentes.
- Con relación a las librerías que existen para graficar mapas se encuentran: Arcpy, Geopandas, GDAL/OGR, PyProj. Es importante generar utilizar mapas para datos geoespaciales para evitar sacar conclusiones erróneas, como por ejemplo la existencia de datos atípicos.