MAESTRÍA EN INTELIGENCIA ARTIFICIAL APLICADA

MATERIA
Ciencia y Analítica de Datos

PROFESOR:
Dra. María de La Paz Rico

ALUMNO:

Carlos Enriquez Gorgonio
A01793102

Actividad Semanal -- 7 Regresiones y K means

Noviembre de 2022

El vínculo a la actividad en GitHub es:

https://github.com/PosgradoMNA/actividades-de-aprendizaje-KarltonBotics

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

Tecnológico de Monterrey

- María de la Paz Rico Fernández-
- Ciencia y analítica de datos
- Carlos Enriquez Gorgonio
- A01793102


```
1  ! pip install qeds fiona geopandas xgboost gensim folium pyLDAvis descartes
```

```
Requirement already satisfied: folium in /usr/local/lib/python3.7/dist-packages (0.12.1.post1)
Requirement already satisfied: pyLDAvis in /usr/local/lib/python3.7/dist-packages (3.3.1)
Requirement already satisfied: descartes in /usr/local/lib/python3.7/dist-packages (1.1.0)
```

```
Requirement already satisfied: pyarrow in /usr/local/lib/python3.7/dist-packages (from qeds) (6.0.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from qeds) (1.21.6)
Requirement already satisfied: pandas-datareader in /usr/local/lib/python3.7/dist-packages (from qeds) (0.9.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from qeds) (3.2.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from qeds) (5.5.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/dist-packages (from qeds) (0.11.2)
Requirement already satisfied: quantecon in /usr/local/lib/python3.7/dist-packages (from qeds) (0.5.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from qeds) (1.0.2)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (from qeds) (0.12.2)
Requirement already satisfied: quandl in /usr/local/lib/python3.7/dist-packages (from qeds) (3.7.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from qeds) (2.23.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from qeds) (1.3.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from qeds) (1.7.3)
Requirement already satisfied: openpyxl in /usr/local/lib/python3.7/dist-packages (from qeds) (3.0.10)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from fiona) (2022.9.24)
Requirement already satisfied: six>=1.7 in /usr/local/lib/python3.7/dist-packages (from fiona) (1.15.0)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.7/dist-packages (from fiona) (0.7.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from fiona) (57.4.0)
Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/dist-packages (from fiona) (22.1.0)
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.7/dist-packages (from fiona) (1.1.1)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/dist-packages (from fiona) (7.1.2)
Requirement already satisfied: munch in /usr/local/lib/python3.7/dist-packages (from fiona) (2.5.0)
Requirement already satisfied: pyproj>=2.2.0 in /usr/local/lib/python3.7/dist-packages (from geopandas) (3.2.1)
Requirement already satisfied: shapely>=1.6 in /usr/local/lib/python3.7/dist-packages (from geopandas) (1.8.5.post1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->qeds) (2022.6)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->qeds) (2.8.2)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from gensim) (5.2.1)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.7/dist-packages (from folium) (2.11.3)
Requirement already satisfied: branca>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from folium) (0.5.0)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2>=2.9->folium) (2.0.1)
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages (from pyLDAvis) (0.0.post1)
Requirement already satisfied: funcy in /usr/local/lib/python3.7/dist-packages (from pyLDAvis) (1.17)
Requirement already satisfied: future in /usr/local/lib/python3.7/dist-packages (from pyLDAvis) (0.16.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from pyLDAvis) (1.2.0)
Requirement already satisfied: numexpr in /usr/local/lib/python3.7/dist-packages (from pyLDAvis) (2.8.4)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->qeds) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplot
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->qeds) (1.4.4)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib-
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.7/dist-packages (from openpyxl->qeds) (1.1.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.7/dist-packages (from pandas-datareader->qeds) (4.9.1)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->qeds) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->qeds) (3.0.4)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-packages (from plotly->qeds) (8.1.0)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-packages (from quandl->qeds) (9.0.0)
Requirement already satisfied: inflection>=0.3.1 in /usr/local/lib/python3.7/dist-packages (from quandl->qeds) (0.5.1)
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-packages (from quantecon->qeds) (0.56.4)
Requirement already satisfied: sympy in /usr/local/lib/python3.7/dist-packages (from quantecon->qeds) (1.7.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.7/dist-packages (from numba->quantecon->qed
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from numba->quantecon->qeds) (4.13
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->numba->quantecon->
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->qeds) (3.1.0)
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.7/dist-packages (from statsmodels->qeds) (0.5.3)
```

```
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

```
#Car¿gamos el DataSet  y lo llamamos df
url="https://raw.githubusercontent.com/marypazrf/bdd/main/target-locations.csv"
df=pd.read_csv(url)
```

Exploremos los datos.

```
#Vemos que el DataSet tiene 6 columnas
df.head()

```

	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

```
#Observamos que el DataSet no tiene datos faltantes
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.

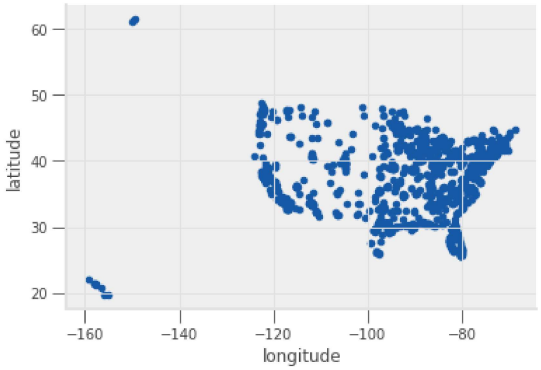
```
#Creamos un nuevo DataSet que solo contendra 3 columnas, seleccionada como las caracteristicas que usaremos
latlong=df[["latitude","longitude"]]
```

¡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")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f54249cb890>
```




```
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

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

```
#Importamos las librearias que vamos a utilizar
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();
```

```
#Creamos una nueva columna que contendra las coordenadas de cada columna, lo cual es basicamente invertir el oorden de las columna
df["Coordinates"] = list(zip(df.longitude, df.latitude))
df["Coordinates"] = df["Coordinates"].apply(Point)
df.head()
```

	name	latitude	longitude	address	phone	website	Coordinates
0	Alabaster	33.224225	-86.804174	250 S Colonial Dr, Alabaster, AL 35007-4657	205-564-2608	https://www.target.com/sl/alabaster/2276	POINT (-86.80417369999999 33.2242254)
1	Bessemer	33.334550	-86.989778	4889 Promenade Pkwy, Bessemer, AL 35022-7305	205-565-3760	https://www.target.com/sl/bessemer/2375	POINT (-86.98977789999999 33.3345501)
2	Daphne	30.602875	-87.895932	1698 US Highway 98, Daphne, AL	251-621-	https://www.target.com/sl/daphne/1274	POINT (-87.89593169999999

```
# Usamos la funcion GeoDataFrame
gdf = gpd.GeoDataFrame(df, geometry="Coordinates")
gdf.head()
```

	name	latitude	longitude	address	phone	website	Coordinates
0	Alabaster	33.224225	-86.804174	250 S Colonial Dr, Alabaster, AL 35007-4657	205-564-2608	https://www.target.com/sl/alabaster/2276	POINT (-86.80417 33.22423)
1	Bessemer	33.334550	-86.989778	4889 Promenade Pkwy, Bessemer, AL 35022-7305	205-565-3760	https://www.target.com/sl/bessemer/2375	POINT (-86.98978 33.33455)
2	Daphne	30.602875	-87.895932	1698 US Highway 98, Daphne, AL	251-621-	https://www.target.com/sl/daphne/1274	POINT (-87.89593

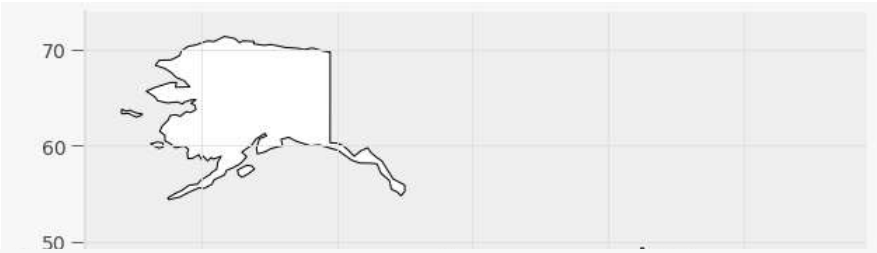
```
#mapa

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

world.head()
```

iso_a3	pop_est	continent	name	gdp_md_est	geometry
FJI	920938	Oceania	Fiji	8374.0	MULTIPOLYGON (((180.00000 -16.06713, 180.00000...
TZA	53950935	Africa	Tanzania	150600.0	POLYGON ((33.90371 -0.95000, 34.07262 -1.05982...
ESH	603253	Africa	W. Sahara	906.5	POLYGON ((-8.66559 27.65643, -8.66512 27.58948...

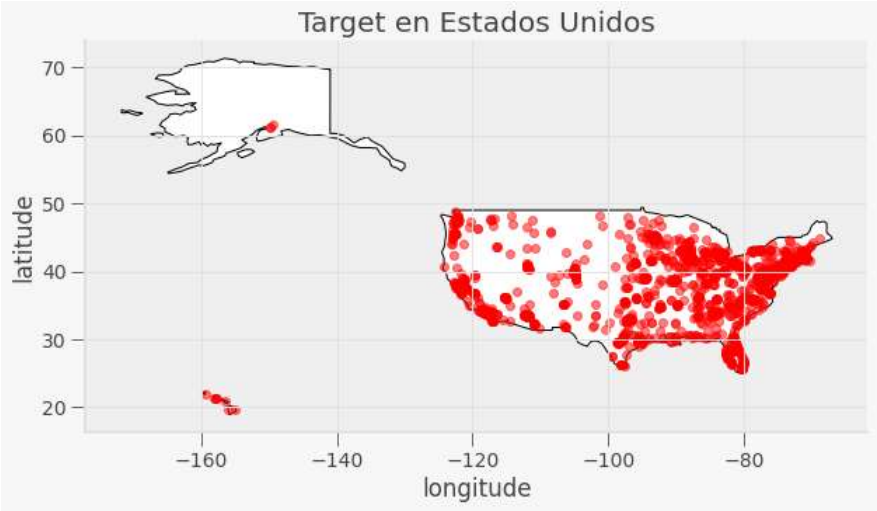
```
#graficar el mapa
world.name.unique()
```

```
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.', '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()
```




```
#Librerias que se usaran a partir de esta parte
from sklearn.cluster import KMeans
from geopy.geocoders.yandex import Location
from geopy.geocoders import Nominatim
from geopy.distance import geodesic
```

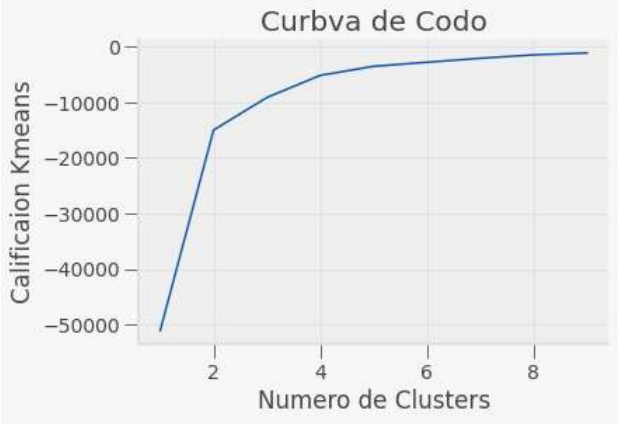
¿qué tal ahora?, tiene mayor sentido verdad, entonces los datos lejanos no eran atípicos, de aquí la importancia de ver los datos con el tipo de gráfica correcta.

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.

```
K_clusters = range(1,10)
kmeans = [KMeans(n_clusters=i) for i in K_clusters] #Aqui vamos evaluadno el kmeans en un numero de clusters, para ver cual es el
Y_axis = latlong[['latitude']]
X_axis = latlong[['longitude']]
score = [kmeans[i].fit(Y_axis).score(Y_axis) for i in range(len(kmeans))] #Evalua tus componentes minimos,

```

```
#Ploteamos los clusters en comparación con los scores
plt.plot(K_clusters, score)
plt.xlabel('Numero de Clusters')
plt.ylabel('Calificaion Kmeans')
plt.title('Curbva de Codo')
plt.show()
```



Se identifica la interseci[on de la curva entre el 2 y el 4, por lo tanto el punto optimo es el numero 3, correspondiente a los almacenes que seleccionaremos

```
kmeans = KMeans(n_clusters = 3, init ='k-means++') #Aqui no creamos nuevos clusters, solamente re creamos 3, que son los que nece
kmeans.fit(latlong[latlong.columns[0:2]])
labels = kmeans.labels_
labels
```

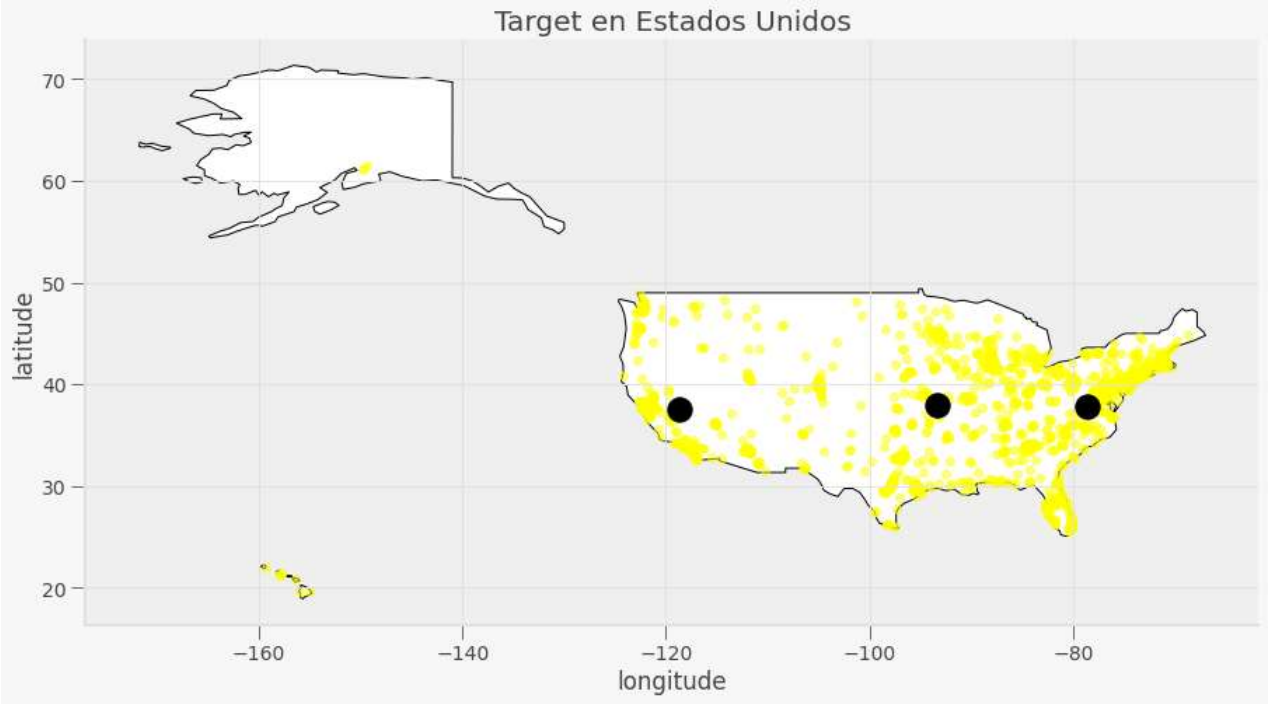
```
array([2, 2, 2, ..., 0, 2, 0], dtype=int32)
```

```
#USamos el mismo codigo del ejercicio de ejemplo para a;adir una columna al dataframe con las coordenadas

X = df[["longitude","latitude"]]

kmeans = KMeans(n_clusters=3).fit(X)
centroids = kmeans.cluster_centers_
labels = kmeans.predict(X)
C = kmeans.cluster_centers_

C_DF = pd.DataFrame(C)
C_DF["Coordinates"] = list(zip(C_DF[0], C_DF[1]))
C_DF["Coordinates"] = C_DF["Coordinates"].apply(Point)


gdf_C = gpd.GeoDataFrame(C_DF, geometry="Coordinates")
gdf_C
```

	0	1	Coordinates
0	-78.569908	37.789554	POINT (-78.56991 37.78955)
1	-118.624473	37.487342	POINT (-118.62447 37.48734)
2	-93.327172	37.980063	POINT (-93.32717 37.98006)

```
fig, gax = plt.subplots(figsize=(15,10))

world.query("name == 'United States of America'").plot(ax = gax, edgecolor='black', color='white')

gdf.plot(ax=gax, color='yellow', alpha = 0.5)
gdf_C.plot(ax=gax, color='black', alpha = 1, markersize = 300)

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()
```




```
latlong['kmeans'] = kmeans.labels_
latlong.loc[:, 'kmeans'].value_counts()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers
  """Entry point for launching an IPython kernel.
0    826
2    628
1    385
Name: kmeans, dtype: int64
```

```
Location1 = str(gdf_C[1][0]) + ", " + str(gdf_C[0][0])
print('Ubicacion 1---', Location1)
Location2 = str(gdf_C[1][1]) + ", " + str(gdf_C[0][1])
print('Ubicacion 2---',Location2)
Location3 = str(gdf_C[1][2]) + ", " + str(gdf_C[0][2])
print('Ubicacion 3---',Location3)

```

```
Ubicacion 1 37.789554004474006, -78.56990807484885
Ubicacion 2 37.48734203064935, -118.62447331844157
Ubicacion 3 37.98006260590112, -93.32717230430622
```

```
geolocator = Nominatim(user_agent="my-application") #Inicializamos la libreria
Locations = [Location1, Location2, Location3] #Metemos las coordenadas obtenidas

for ciudad in Locations:
  location = geolocator.reverse(ciudad)
  #invertimos la busqueda, introduciendo coordenadas obtenemos el nobre
  print('almacen en ---', location.address)
```

```
almacen en --- Langhorne Road, Totier Hills, Albemarle County, Virginia, 22946, United States
almacen en --- Paradise Estates, Mono County, California, United States
almacen en --- Hickory County, Missouri, United States
```

```
distancia1 = str(geodesic(Location1, Location2).miles)
print("\nDistancia entre el primer y segundo almacén : ", distancia1, " ft2 \n")
distancia2 = str(geodesic(Location2, Location3).miles)
print("Distancia entre el segundo y tercer almacén : ", distancia2, " ml \n")


#type(distancia1)
#int(distancia1)
#Convertimos a Kilometros dividieno millas entre 1.16

print (2179.65/1.16)
print(1381.42/1.16)

```

```
Distancia entre el primer y segundo almacén :  2179.654449831999  ft2

Distancia entre el segundo y tercer almacén :  1381.7597109962394  ml

1879.0086206896553
1190.8793103448277
```

Encuentra el numero ideal de almacenes, justifica tu respuesta:

Encuentra las latitudes y longitudes de los almacenes, ¿qué ciudad es?, ¿a cuantas tiendas va surtir?, ¿sabes a que distancia estará? ¿Cómo elegiste el número de almacenes?, justifica tu respuesta técnicamente. Adicionalmente, en el notebook notaras que al inicio exploramos los datos y los graficamos de manera simple, después nos auxiliamos de una librería de datos geográficos.

¿qué librerías nos pueden ayudar a graficar este tipo de datos? ¿Consideras importante que se grafique en un mapa?, ¿por qué? Agrega las conclusiones

Conclusiones El numero ideal de almacenes para surtir todas las sucursales, se ubico en 3, ya que la grafica que compara los clústers creados con las metricas kmeans, permite observar un quiebre en la curvbaturoa notable justo en el numero 3

Las ciudades son: Albemarle Country, Mono Country, y Hickory. y surtiran 826, 628 y 385 tiendas respectivamente, las distancias

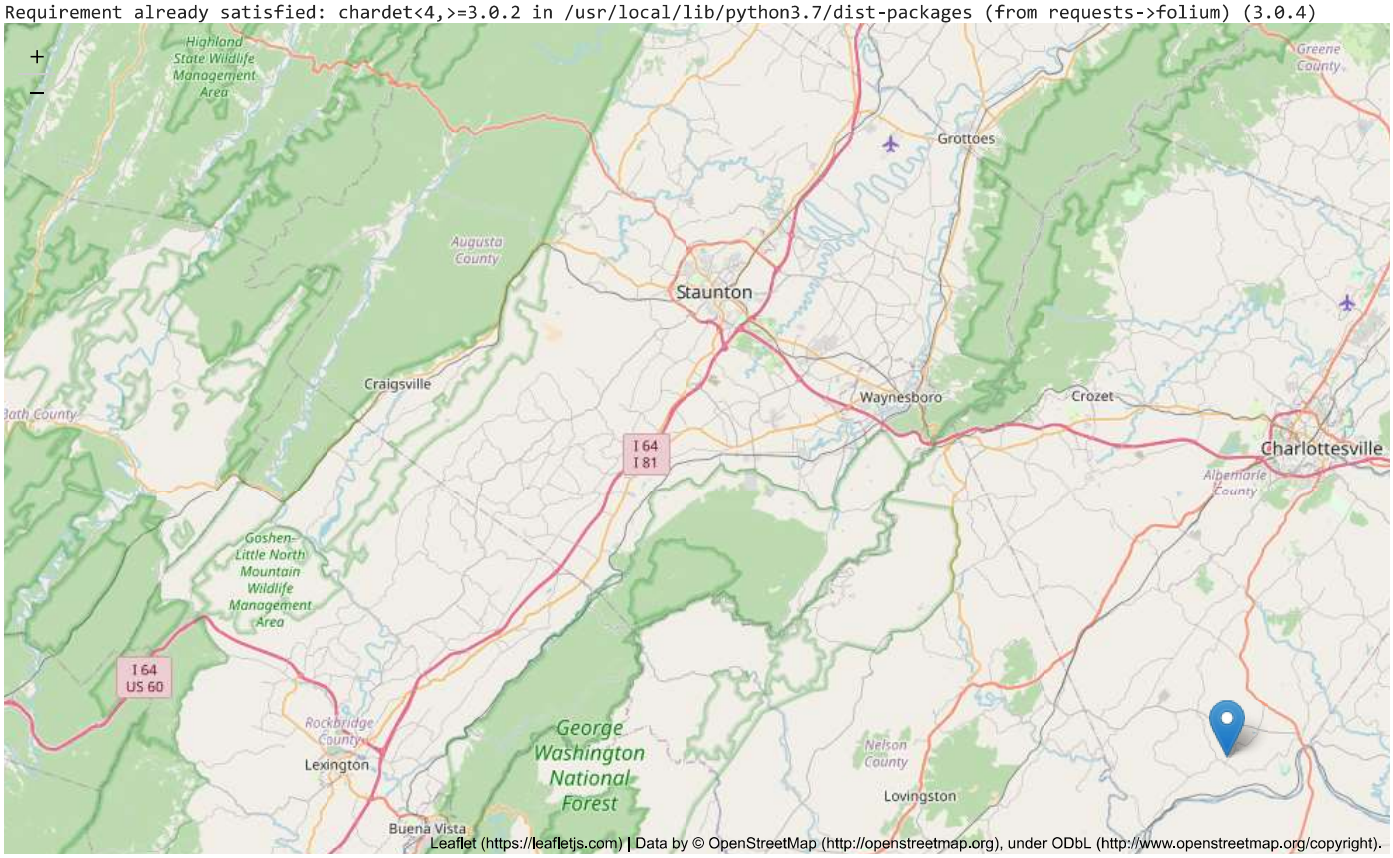
- CIUDAD------TIENAS--SEPARACIÓN
- Albemarle---826-----0 Km
- Mono--------628-----1879 km
- Hickory-----385-----1190 km

A continuacion tambeinse añade la libreria folium que nos permite graficar este tipo de datos, los cuales son geograficos.

Este tipo de datos son necesarios de graficar, ya que por el tipo de actividad es un entorno directivo, el cual requiere elemtnos graficos para la toma de desiciones, es decir, el equipo de planeacion no necesita coordenadas, necesita mapas y nombres de ciudades, para poder anlizar el contexto tanto social, como economis

```
!pip install folium
import folium
mapa = folium.Map(location=[37.789554004474006, -78.56990807484885])
mapa

coordenada = 'Almacen 1'

folium.Marker([37.789554004474006, -78.56990807484885],tooltip= coordenada).add_to(mapa)
mapa
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: folium in /usr/local/lib/python3.7/dist-packages (0.12.1.post1)
Requirement already satisfied: jinja2>=2.9 in /usr/local/lib/python3.7/dist-packages (from folium) (2.11.3)
Requirement already satisfied: branca>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from folium) (0.5.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from folium) (2.23.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from folium) (1.21.6)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2>=2.9->folium) (2.0.1)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (2022.9.24)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->folium) (3.0.4)
```



Linear Models

```
1
```

- 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.

```
1 import numpy as np
2 %matplotlib inline
3 import matplotlib
4 import matplotlib.pyplot as plt
5 import pandas as pd
6 import seaborn as sns
7 # to make this notebook's output stable across runs
8 np.random.seed(42)
```

```
1 5-2
```

```
3
```

Simple Linear Regression

Simple linear regression equation:

$y = ax + b$

a: slope

b: intercept

Generate linear-looking data with the equation:

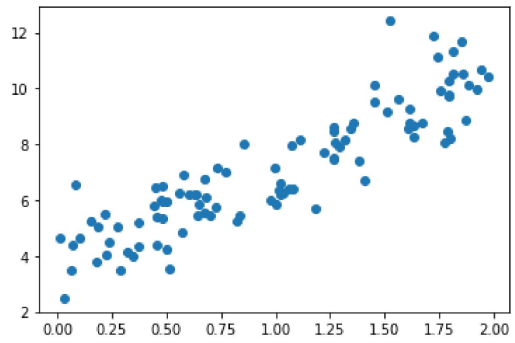
$y = 3X + 4 + noise$

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

```
array([[0.37454012],
       [0.95071431],
       [0.73199394],
       [0.59865848],
       [0.15601864],
       [0.15599452],
       [0.05808361],
       [0.86617615],
       [0.60111501],
       [0.70807258],
       [0.02058449],
       [0.96990985],
       [0.83244264],
       [0.21233911],
       [0.18182497],
       [0.18340451],
       [0.30424224],
       [0.52475643],
       [0.43194502],
       [0.29122914],
       [0.61185289],
       [0.13949386],
       [0.29214465],
       [0.36636184],
       [0.45606998],
       [0.78517596],
       [0.19967378],
       [0.51423444],
       [0.59241457],
       [0.04645041],
       [0.60754485],
       [0.17052412],
```

```
[0.06505159],
[0.94888554],
[0.96563203],
[0.80839735],
[0.30461377],
[0.09767211],
[0.68423303],
[0.44015249],
[0.12203823],
[0.49517691],
[0.03438852],
[0.9093204 ],
[0.25877998],
[0.66252228],
[0.31171108],
[0.52006802],
[0.54671028],
[0.18485446],
[0.96958463],
[0.77513282],
[0.93949894],
[0.89482735],
[0.59789998],
[0.92187424],
[0.0884925 ],
[0.19598286],
```

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




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

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

100 rows × 1 columns

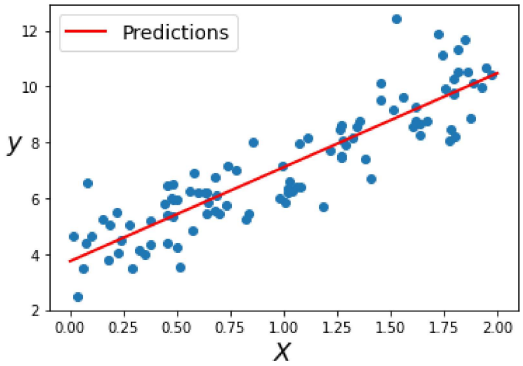
```
from sklearn.linear_model import LinearRegression

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

```
LinearRegression()
```

Plot the model's predictions:

```
#X_fit[]
```

```
# construct best fit line
X_fit = np.linspace(0, 2, 100)
y_fit = linear_reg.predict(X_fit[:, np.newaxis])

plt.scatter(X, y)
plt.plot(X_fit, y_fit, "r-", linewidth=2, label="Predictions")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);
```



Predictions are a good fit.

Generate new data to make predictions with the model:

```
X_new = np.array([[0], [2]])
X_new
```

```
array([[0],
       [2]])
```

```
X_new.shape
```

```
(2, 1)
```

```
y_new = linear_reg.predict(X_new)
y_new
```

```
array([[ 3.74406122],
       [10.47517611]])
```

```
 linear_reg.coef_, linear_reg.intercept_
```

```
(array([[3.36555744]]), array([3.74406122]))
```

The model estimates:

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

```
#|VENTAS|GANANCIAS|
#COEF*VENTAS+B
#|VENTAS|COMPRAS|GANANCIAS|
#COEF1*X1+COEF2*X2+B=Y
```

Polynomial Regression

If data is more complex than a straight line, you can use a linear model ti fit non-linear data adding powers of each feature as new features and then train a linear model on the extended set of features.

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

to

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

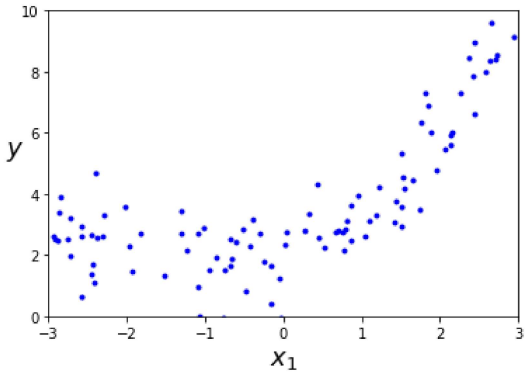
This is still a linear model, the linearity refers to the fact that the coefficients never multiply or divide each other.

To generate polynomial data we use the function:

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

```
# generate non-linear data e.g. quadratic equation
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```

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




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

	0
0	8.529240
1	3.768929
2	3.354423
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.

```
from sklearn.preprocessing import PolynomialFeatures

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

```
X_poly
```

```
array([[ 2.72919168e+00,  7.44848725e+00],
       [ 1.42738150e+00,  2.03741795e+00],
       [ 3.26124315e-01,  1.06357069e-01],
       [ 6.70324477e-01,  4.49334905e-01],
       [-4.82399625e-01,  2.32709399e-01],
       [-1.51361406e+00,  2.29102753e+00],
       [-8.64163928e-01,  7.46779295e-01],
       [ 1.54707666e+00,  2.39344620e+00],
       [-2.91363907e+00,  8.48929262e+00],
       [-2.30356416e+00,  5.30640783e+00],
       [-2.72398415e+00,  7.42008964e+00],
       [-2.75562719e+00,  7.59348119e+00],
       [ 2.13276350e+00,  4.54868016e+00],
       [ 1.22194716e+00,  1.49315485e+00],
       [-1.54957025e-01,  2.40116797e-02],
       [-2.41299504e+00,  5.82254504e+00],
       [-5.03047493e-02,  2.53056780e-03],
       [-1.59169375e-01,  2.53348900e-02],
       [-1.96078878e+00,  3.84469264e+00],
       [-3.96890105e-01,  1.57521755e-01],
       [-6.08971594e-01,  3.70846402e-01],
       [ 6.95100588e-01,  4.83164828e-01],
       [ 8.10561905e-01,  6.57010602e-01],
       [-2.72817594e+00,  7.44294397e+00],
       [-7.52324312e-01,  5.65991871e-01],
       [ 7.55159494e-01,  5.70265862e-01],
       [ 1.88175515e-02,  3.54100244e-04],
       [ 2.13893905e+00,  4.57506025e+00],
       [ 9.52161790e-01,  9.06612074e-01],
       [-2.02239344e+00,  4.09007522e+00],
       [-2.57658752e+00,  6.63880323e+00],
       [ 8.54515669e-01,  7.30197029e-01],
       [-2.84093214e+00,  8.07089541e+00],
       [ 5.14653488e-01,  2.64868212e-01],
       [ 2.64138145e+00,  6.97689596e+00],
       [ 4.52845067e-01,  2.05068655e-01],
       [-6.70980443e-01,  4.50214755e-01],
       [ 8.59729311e-01,  7.39134488e-01],
       [-2.50482657e-01,  6.27415615e-02],
       [ 2.73700736e-01,  7.49120928e-02],
       [ 2.64878885e+00,  7.01608239e+00],
       [-6.83384173e-01,  4.67013928e-01],
       [ 2.76714338e+00,  7.65708250e+00],
       [ 2.43210385e+00,  5.91512915e+00],
       [-1.82525319e+00,  3.33154921e+00],
       [-2.58383219e+00,  6.67618881e+00],
       [-2.39533199e+00,  5.73761535e+00],
       [-2.89066905e+00,  8.35596753e+00],
       [-2.43334224e+00,  5.92115443e+00],
       [ 1.09804064e+00,  1.20569325e+00],
       [-2.57286811e+00,  6.61965031e+00],
       [-1.08614622e+00,  1.17971361e+00],
       [ 2.06925187e+00,  4.28180328e+00],
       [-2.86036839e+00,  8.18170730e+00],
       [ 1.88681090e+00,  3.56005536e+00],
       [-1.30887135e+00,  1.71314421e+00],
       [-2.29101103e+00,  5.24873156e+00],
       [ 1.18042299e+00,  1.39339844e+00],
```

`X_poly` now contains the original feature of X plus the square of the feature:

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

```

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

```
1 X_poly[0]
```

```
array([2.72919168, 7.44848725])
```

Fit the model to this extended training data:

```
1 lin_reg = LinearRegression(fit_intercept=True)
2 lin_reg.fit(X_poly, y)
3 lin_reg.coef_, lin_reg.intercept_
```

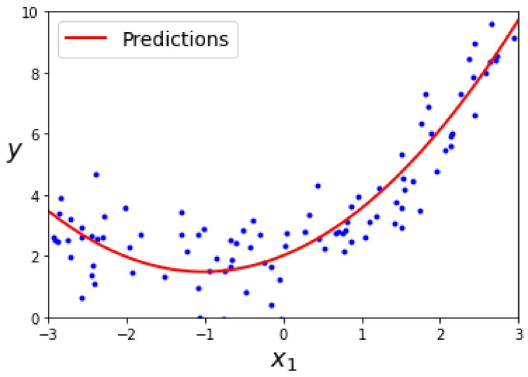
```
(array([[1.04271531, 0.50866711]]), array([2.01873554]))
```

The model estimates:

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

Plot the data and the predictions:

```
1 X_new=np.linspace(-3, 3, 100).reshape(100, 1)
2 X_new_poly = poly_features.transform(X_new)
3 y_new = lin_reg.predict(X_new_poly)
4 plt.plot(X, y, "b.")
5 plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
6 plt.xlabel("$x_1$", fontsize=18)
7 plt.ylabel("$y$", rotation=0, fontsize=18)
8 plt.legend(loc="upper left", fontsize=14)
9 plt.axis([-3, 3, 0, 10]);
```



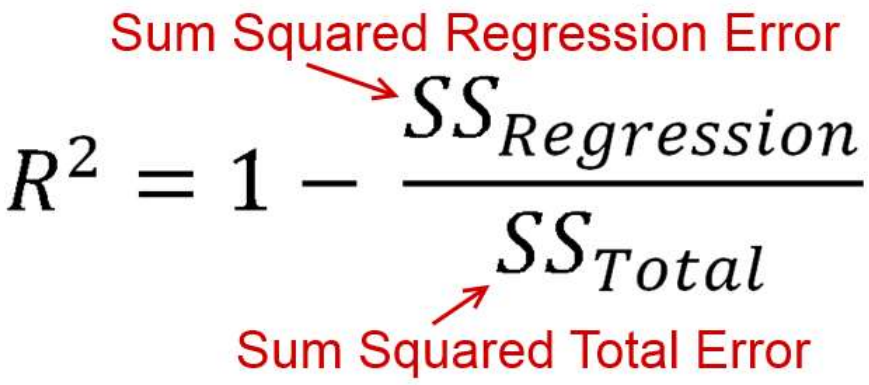

```
1
```

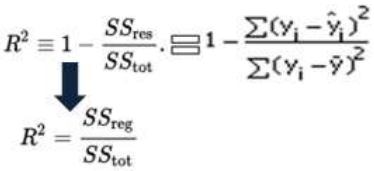
R square

R² es una medida estadística de qué tan cerca están los datos de la línea de regresión ajustada. También se conoce como el coeficiente de determinación o el coeficiente de determinación múltiple para la regresión múltiple. Para decirlo en un lenguaje más simple, R² es una medida de ajuste para los modelos de regresión lineal.

R² no indica si un modelo de regresión se ajusta adecuadamente a sus datos. Un buen modelo puede tener un valor R² bajo. Por otro lado, un modelo sesgado puede tener un valor alto de R².

SSres + SSreg = SStot, R² = Explained variation / Total Variation





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

	Number of Units	Manufacturing Cost
968	7.065653	27.804027
212	3.372115	41.127212
416	4.194513	43.832711
677	5.068888	41.225741
550	4.604122	37.569764
764	5.389522	31.191501
386	4.104190	42.988730
339	3.942214	46.291435
82	2.665856	48.578425
487	4.399514	37.567914

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

```
1 len(X)
```

```
1000
```

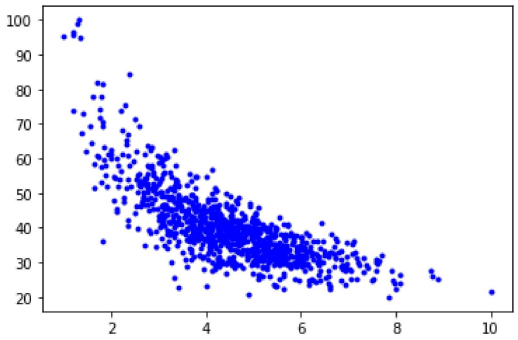
```
1 y.describe
```

```
<bound method NDFrame.describe of 0      95.066056
1      96.531750
2      73.661311
```

```
3      95.566843
4      98.777013
         ...
995    23.855067
996    27.536542
997    25.973787
998    25.138311
999    21.547777
Name: Manufacturing Cost, Length: 1000, dtype: float64>
```

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

```
[<matplotlib.lines.Line2D at 0x7f61f0157dd0>]
```



Las nuevas librerias para esta parte del ejercicio son

```
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.model_selection import train_test_split, GridSearchCV, train_test_split, RepeatedKFold, cross_validate
from sklearn.preprocessing import PolynomialFeatures

```

```
#HAcemos las division de nuestros datos de prueba y entrenamiento
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 101)

lista_para_mae =[]
lista_para_r2 =[]
```

CODIGO PARA REGRESION LINEAL

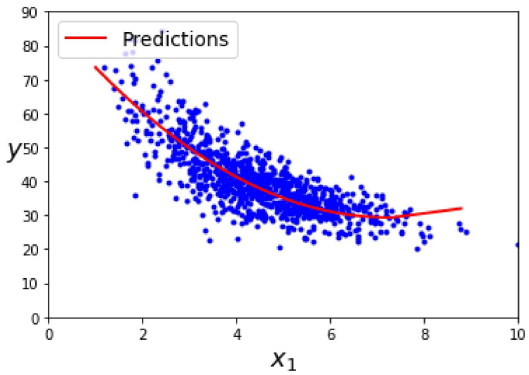
```
linear_reg = LinearRegression(fit_intercept=True)
linear_reg.fit(X_train, y_train)
X_para_regresion = X_test
y_para_regresion = linear_reg.predict(X_para_regresion)
plt.scatter(X_train, y_train)
plt.plot(X_para_regresion, y_para_regresion, "r-", linewidth=2, label="Predicciones")
plt.xlabel("$X$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14);


print('El modelo es: Y =', linear_reg.coef_,'X +', linear_reg.intercept_)
print('Error Medio Absoluto (MAE):', metrics.mean_absolute_error(y_test,y_para_regresion))
print('Error Medio Cuadrado (RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_para_regresion)))
print('r2_score:', r2_score(y_test,y_para_regresion))

```

```
El modelo es: Y = [-5.98882699] X + 66.83650741226988
Error Medio Absoluto (MAE): 5.013587781954963
Error Medio Cuadrado (RMSE): 7.108963321847682
r2_score: 0.6116251549562579
```


```
caracteristicas_para_poly = PolynomialFeatures(degree=2, include_bias=False)
X_polinomial = caracteristicas_para_poly.fit_transform(X_train)
print("Caracteristicas de entrada",caracteristicas_para_poly.n_input_features_)
print("Caracteristicas de salida",caracteristicas_para_poly.n_output_features_)
print("Exponentes de las caracteristicas de entrada",caracteristicas_para_poly.powers_)

regresion_lineal_poli = LinearRegression(fit_intercept=True)
regresion_lineal_poli.fit(X_polinomial, y_train)
print ('Punto de intercepcion',regresion_lineal_poli.coef_, regresion_lineal_poli.intercept_)
# Vemos las caracteristicas para evitar errores posteriores
X_polinomial.shape
X_polinomial_test = caracteristicas_para_poly.fit_transform(X_test)
X_polinomial_test.shape
y_con_regresion_poli = regresion_lineal_poli.predict(X_polinomial_test)
y_con_regresion_poli.shape

```

```
Caracteristicas de entrada 1
Caracteristicas de salida 2
Exponentes de las caracteristicas de entrada [[1]
 [2]]
Punto de intercepcion [-16.40638102   1.13136095] 88.80179909112496
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:103: FutureWarning: The attribute `n_input_features_` was dep
  warnings.warn(msg, category=FutureWarning)
(100,)
```

```

#Graficamos
order = np.argsort(X_test.values.ravel())

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

plt.plot(X, y, "b.")
plt.plot(sortedXPoly, sorted_predicPoly, "r-", linewidth=2, label="Predictions")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 10, 0, 90]);
```



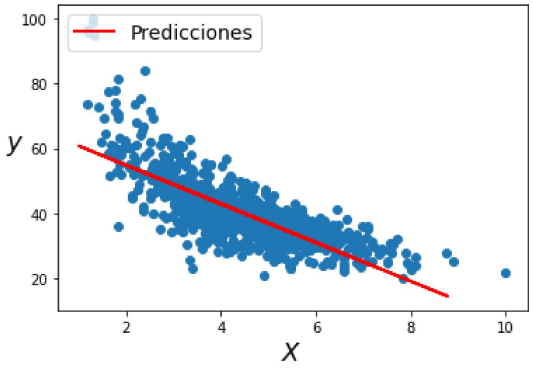
CODIGO PARA REGRESION MULTIPLE

```
print('El modelo es: Y =', regresion_lineal_poli.coef_[1],'X^2 +', regresion_lineal_poli.coef_[0],'X +',regresion_lineal_poli.inter
```

```
2
3 metrica_mae = metrics.mean_absolute_error(y_test, y_con_regresion_poli)
4 r2Score = r2_score(y_test, y_con_regresion_poli)
5 print('Error medio Absoluto (MAE):', metrica_mae)
6 print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_con_regresion_poli)))
7 print('r2_score',r2Score)
```

```
El modelo es: Y = 1.1313609537119216 X^2 + -16.406381017212386 X + 88.80179909112496
Error medio Absoluto (MAE): 4.3833025759681075
Root Mean Squared Error: 5.832771301068423
r2_score 0.7385501224942537
```

```
1 #Imprimrimos nuestra regresion
2 mi_ridge = Ridge(alpha=5.0,fit_intercept=True)
3 mi_ridge.fit(X_train, y_train)
4 X_para_ridge = X_test
5 y_para_ridge = mi_ridge.predict(X_para_ridge)
6 plt.scatter(X_train, y_train)
7 plt.plot(X_para_ridge, y_para_ridge, "r-", linewidth=2, label="Predicciones")
8 plt.xlabel("$X$", fontsize=18)
9 plt.ylabel("$y$", rotation=0, fontsize=18)
10 plt.legend(loc="upper left", fontsize=14);
```




```
1 metrica_mae_ridge = metrics.mean_absolute_error(y_test, y_para_ridge)
2 r2Score = r2_score(y_test, y_para_ridge)
3 print('Error medio Absoluto (MAE):', metrica_mae_ridge)
4 print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_para_ridge)))
5 print('r2_score',r2Score)
6
7 print('El modelo es: Y =', mi_ridge.coef_,'X +', mi_ridge.intercept_)
```

```
Error medio Absoluto (MAE): 5.0162057389928325
Root Mean Squared Error: 7.1111119498200965
r2_score 0.6113903530239646
El modelo es: Y = [-5.97003397] X + 66.75243237759665
```

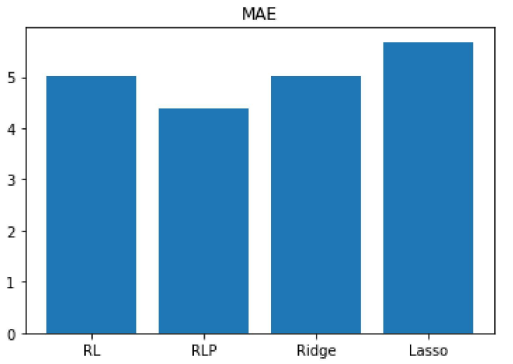
```
1 mi_lasso = Lasso(alpha=5.0,fit_intercept=True)
2 mi_lasso.fit(X_train, y_train)
3 X_para_lasso = X_test
4 y_para_lasso = mi_lasso.predict(X_para_ridge)
5 plt.scatter(X_train, y_train)
6 plt.plot(X_para_lasso, y_para_lasso, "r-", linewidth=2, label="Predicciones")
7 plt.xlabel("$X$", fontsize=18)
8 plt.ylabel("$y$", rotation=0, fontsize=18)
9 plt.legend(loc="upper left", fontsize=14);
```

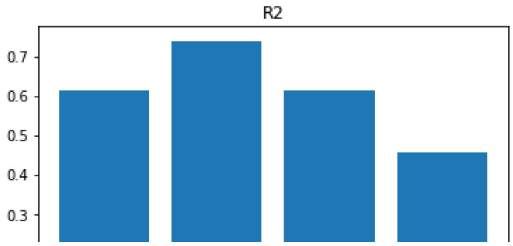

```
metrica_mae_lasso = metrics.mean_absolute_error(y_test, y_para_lasso)
r2Score = r2_score(y_test, y_para_lasso)
print('Error medio Absoluto (MAE):', metrica_mae_lasso)
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_para_lasso)))
print('r2_score',r2Score)

print('El modelo es: Y =', mi_lasso.coef_,'X +', mi_lasso.intercept_)
```

```
Error medio Absoluto (MAE): 5.681207654677401
Root Mean Squared Error: 8.409660991642687
r2_score 0.456505036516648
El modelo es: Y = [-3.15572458] X + 54.16195119377413
```

GRAFICAMOS NUESTRAS METRICAS

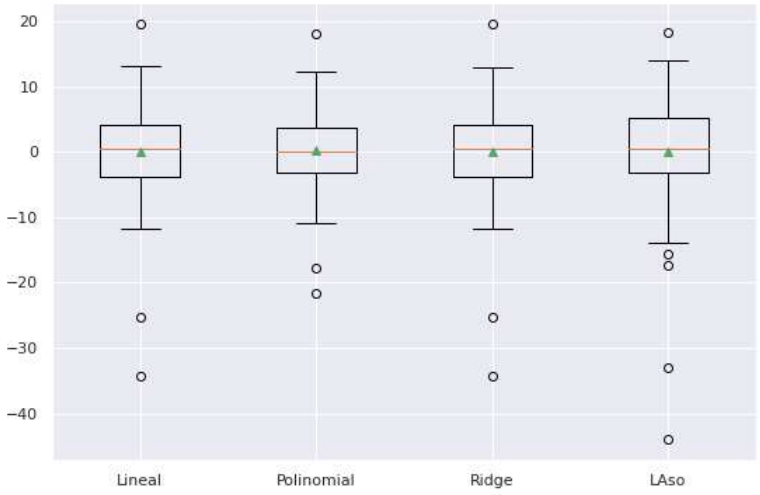
```

#Anadimos las metricas a la lista - Regresion Simple
mae_regresion_lineal_simple = metrics.mean_absolute_error(y_test,y_para_regresion)
lista_para_mae.append(mae_regresion_lineal_simple)
r2_regresion_lineal_simple = r2_score(y_test,y_para_regresion)
lista_para_r2.append(r2_regresion_lineal_simple)

#Anadimos las metricas a la lista - Regresion Polinomial
mae_regresion_lineal_multiple = metrics.mean_absolute_error(y_test,y_con_regresion_poli)
lista_para_mae.append(mae_regresion_lineal_multiple)
r2_regresion_lineal_multiple = r2_score(y_test,y_con_regresion_poli)
lista_para_r2.append(r2_regresion_lineal_multiple)

#Anadimos las metricas a la lista - Ridge
mae_ridge = metrics.mean_absolute_error(y_test,y_para_ridge)
lista_para_mae.append(mae_ridge) ,
r2_ridge= r2_score(y_test,y_para_ridge)
lista_para_r2.append(r2_ridge)

#Anadimos las metricas a la lista - Lasso
mae_lasso = metrics.mean_absolute_error(y_test,y_para_lasso)
lista_para_mae.append(mae_lasso)
r2_lasso= r2_score(y_test,y_para_lasso)
lista_para_r2.append(r2_lasso)


nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')

plt.bar(nombres, lista_para_mae)
plt.title('MAE')
plt.show()

nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')

plt.bar(nombres, lista_para_r2)
plt.title('R2')
plt.show()

```





HAcemos el box plot de errores

```
residuo_regresion_simple=y_para_regresion - y_test
residuo_regresion_poliforme= y_con_regresion_poli - y_test
residuo_ridge=y_para_ridge - y_test
residuo_lasso=y_para_lasso - y_test

lista_residuos= [residuo_regresion_simple,residuo_regresion_poliforme,residuo_ridge,residuo_lasso]

nombres=list()
nombres.append('Lineal')
nombres.append('Polinomial')
nombres.append('Ridge')
nombres.append('LAso')

#grafica del MAE (de los cuatro métodos)
sns.set(rc={'figure.figsize':(9,6)})

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



Explica tus resultados, que método conviene más a la empresa, ¿por que?, ¿que porcentajes de entrenamiento y evaluación usaste?, ¿que error tienes?, ¿es bueno?, ¿cómo lo sabes?

CONCLUSIONES

- Que método conviene más a la empresa Considerando que la empresa tiene el objetivo de estimar el costo de los bienes vendidos a medida que varían el volumen de producción (número de unidades producidas). Se tiene unicamente dos caracteristicas principales, sin embargo con base en lo observado el metodo Polinomial es el mejor ***- ¿por que?*** Porque Los datos observados en el r2 que es el erro absoluto, y que entre mayor es el porcentaje de precicion, el modelo es mas exacto, al observarse un r2 mayor en ese modelo, se puede concluir que es el optimo, no obstande que solamente hay dos caractersiticas principales que definen el sistema, que es costo y volumen de produccion, por lo que se podria pensar que un metodo simple era optimo ***- ¿que porcentajes de entrenamiento y evaluación usaste?*** Los que definio por desault el ejercicio que es un 10% de datos de entrenamiento ***- ¿que error tienes?*** Todos los errores MAE calculados oscilan el 50% ***- ¿es bueno?*** No, porque el error debe de bajar para que le modelos ea efectivo, sin llegar a ser cero, pero si acercandose lo mas posible ***- ¿cómo lo sabes?*** Por la literatura que hemos usado en la materia, la cual nos indica que el RMSE es la desviación estándar de los valores residuales (errores de predicción). Los valores residuales son una medida de la distancia de los puntos de datos de la línea de regresión; RMSE es una medida de cuál es el nivel de dispersión de estos valores residuales. En otras palabras, le indica el nivel de concentración de los datos en la línea de mejor ajuste.

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	sqft_living	sqft_lot	floors	waterfront	view	...	grade
5954	7852020250	20140602T000000	725995.0	4	2.50	3190	7869	2.0	0	2	...	9
8610	6392002020	20150324T000000	559000.0	3	1.75	1700	6500	1.0	0	0	...	8
7650	626049058	20150504T000000	275000.0	5	2.50	2570	17234	1.0	0	0	...	7
5683	2202500255	20150305T000000	335000.0	3	2.00	1210	9926	1.0	0	0	...	7
20773	7304301231	20140617T000000	345000.0	3	2.50	1680	2229	2.0	0	0	...	7
6959	723000114	20140505T000000	1395000.0	5	3.50	4010	8510	2.0	0	1	...	9
10784	4104900340	20150204T000000	710000.0	4	2.50	3220	18618	2.0	0	1	...	10
21529	2487200490	20140623T000000	670000.0	3	2.50	3310	5300	2.0	0	2	...	8
12319	2386000070	20141029T000000	795127.0	4	3.25	4360	91158	1.0	0	0	...	10
19948	293070090	20140711T000000	859990.0	4	2.75	3520	5500	2.0	0	0	...	9

10 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21613 non-null  int64  
 1   date           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
```

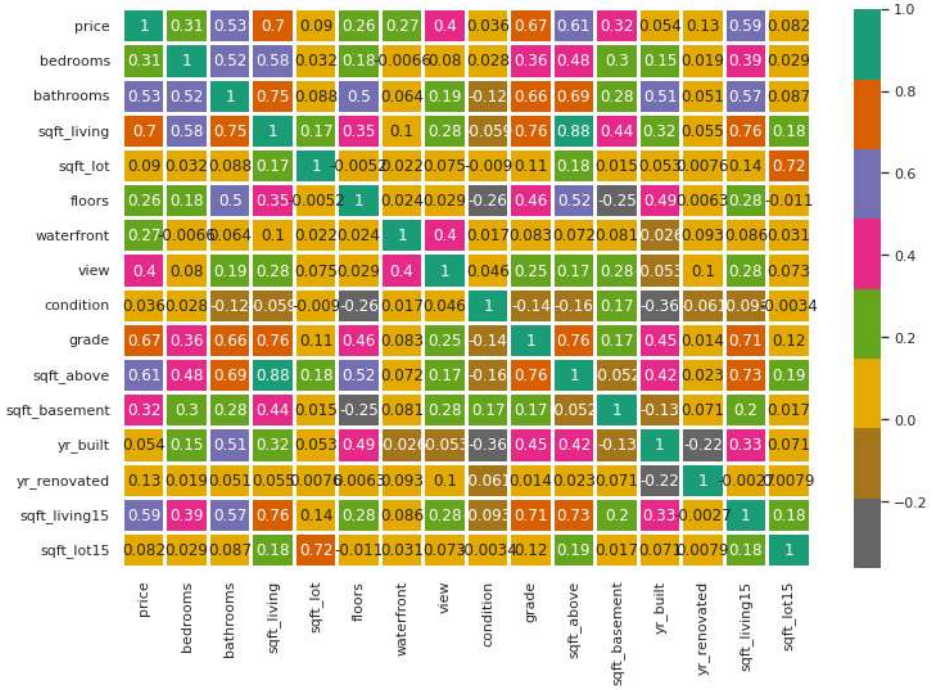
```
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
df.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000

```
df.drop('id', axis = 1, inplace = True)
df.drop('date', axis = 1, inplace = True)
df.drop('zipcode', axis = 1, inplace = True)
df.drop('lat', axis = 1, inplace = True)
df.drop('long', axis = 1, inplace = True)

```

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




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

features = columns
```

```
label = ['price']

X = df[features] #15 columnas
y = df[label]
y
```

	price
0	221900.0
1	538000.0
2	180000.0
3	604000.0
4	510000.0
...	...
21608	360000.0
21609	400000.0
21610	402101.0
21611	400000.0
21612	325000.0

21613 rows × 1 columns

```
from sklearn.model_selection import train_test_split
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size = 0.1, random_state = 101)

print(f'Numero total de registros en la bdd: {len(X)}')
print("*****"*10)
print(f'Numero total de registros en el training set: {len(X_train2)}')
print(f'Tamaño de X_train: {X_train2.shape}')
print("*****"*10)
print(f'Mumero total de registros en el test dataset: {len(X_test2)}')
print(f'Tamaño del X_test: {X_test2.shape}')
print(f'Numero total de registros en el training set: {len(X_train2)}')
print(f'Tamaño de y_train: {y_train2.shape}')
print("*****"*10)
print(f'Mumero total de registros en el test dataset: {len(X_test2)}')
print(f'Tamaño del y_test: {y_test2.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)
Numero total de registros en el training set: 19451
Tamaño de y_train: (19451, 1)
**************************************************
Mumero total de registros en el test dataset: 2162
Tamaño del y_test: (2162, 1)
```

```
#Lineal
linear_reg2 = LinearRegression(fit_intercept=True)
linear_reg2.fit(X_train2, y_train2)
X_para_regresion2 = X_test2
y_para_regresion2 = linear_reg2.predict(X_para_regresion2)


#print('El modelo es: Y =', linear_reg2.coef_,'X +', linear_reg2.intercept_)
print('Error Medio Absoluto (MAE):', metrics.mean_absolute_error(y_test2,y_para_regresion2))
print('Error Medio Cuadrado (RMSE):', np.sqrt(metrics.mean_squared_error(y_test2, y_para_regresion2)))
print('r2_score:', r2_score(y_test2,y_para_regresion2))
```

```
Error Medio Absoluto (MAE): 137480.1388273178
Error Medio Cuadrado (RMSE): 232133.3676240749
r2_score: 0.6579723205007814
```

```
#Polinomial
caracteristicas_para_poly2 = PolynomialFeatures(degree=2, include_bias=False)
X_polinomial2 = caracteristicas_para_poly2.fit_transform(X_train2)
print("Caracteristicas de entrada",caracteristicas_para_poly2.n_input_features_)
print("Caracteristicas de salida",caracteristicas_para_poly2.n_output_features_)
print("Exponentes de las caracteristicas de entrada",caracteristicas_para_poly2.powers_)

regresion_lineal_poli2 = LinearRegression(fit_intercept=True)
regresion_lineal_poli2.fit(X_polinomial2, y_train2)
print ('Punto de intercepcion',regresion_lineal_poli2.coef_, regresion_lineal_poli2.intercept_)
# Vemos las caracteristicas para evitar errores posteriores
X_polinomial2.shape
X_polinomial_test2 = caracteristicas_para_poly2.fit_transform(X_test2)
X_polinomial_test2.shape
y_con_regresion_poli2 = regresion_lineal_poli2.predict(X_polinomial_test2)
y_con_regresion_poli2.shape

metrica_mae2 = metrics.mean_absolute_error(y_test2, y_con_regresion_poli2)
r2Score2 = r2_score(y_test2, y_con_regresion_poli2)
print('Error medio Absoluto (MAE):', metrica_mae2)
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test2, y_con_regresion_poli2)))
print('r2_score',r2Score2)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:103: FutureWarning: The attribute `n_input_features_` was dep
  warnings.warn(msg, category=FutureWarning)
Caracteristicas de entrada 15
Caracteristicas de salida 135
Exponentes de las caracteristicas de entrada [[1 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 0 2 0]
 [0 0 0 ... 0 1 1]
 [0 0 0 ... 0 0 2]]
Punto de intercepcion [[ 9.33912834e+05 -1.09330730e+06 -4.56449932e+02 -2.78686288e+01
  -2.07407238e+06 -3.95431870e+06 -2.88638026e+05  5.58697570e+05
   1.09864510e+06 -2.19758415e+02 -3.82034272e+02 -8.59630908e+04
  -2.95351595e+03  3.99965710e+03 -3.40106316e+01  9.57541696e+02
   7.49958190e+03 -1.37969466e+01 -1.84926695e-02  8.57699569e+03
  -1.14965893e+04 -3.06588900e+02 -5.24628836e+03 -5.36788065e+03
  -1.20757669e+00 -1.46068182e+01 -4.67099451e+02 -9.13191519e+00
   1.72800642e+01  2.15500198e-01 -9.63429892e+03  1.53879172e+01
  -1.35842369e-01 -2.59067293e+04  4.33707238e+04  3.82942459e+03
  -1.07592927e+03  2.22971689e+04  1.44292325e+01  1.33928361e+00
   5.03401980e+02 -1.70798805e+01 -1.72895923e+01 -5.56785248e-02
   1.56185890e+00 -4.34972758e+00  4.83299456e+00  1.60303193e+02
  -1.42711737e+01  1.31138593e+01  2.07283397e+01  2.61335089e+00
  -4.41381412e+00 -2.20765903e+00  6.36282684e-02  3.53629072e+00
   1.88176819e+01  3.06637958e-07  4.40839907e-01 -9.84216136e-02
  -9.75869659e-02  9.17233241e-02  1.51960354e-01  4.34904161e+00
   4.34926327e+00  1.37650380e-02 -1.48516006e-04  1.76864211e-04
   1.05303479e-06  2.16116740e+04 -1.31320713e+05  1.44412938e+04
   2.02989005e+04 -4.27844150e+03 -2.82201875e+00  7.66330754e+00
   1.03664758e+03  2.59349503e+00 -3.20401882e+01 -5.28612817e-01
  -3.95431899e+06 -1.60488762e+04  9.19611219e+03 -1.61637325e+05
   1.83103328e+02 -2.28001023e+01  4.42833048e+03 -2.99177258e+01
   1.70604193e+02 -8.13533327e-01  7.98448565e+03  7.33999963e+03
   1.79556171e+04 -1.24963418e+01 -1.77332663e+00  7.20704241e+01
  -9.96831512e+00  4.63359903e+00 -5.24194412e-02 -5.41914699e+02
  -6.03821219e+03  1.16309467e+00  1.19544378e+01 -3.07074418e+02
  -2.01997367e+01  4.63198881e+01 -2.76877580e-01  7.36624125e+03
   9.60665725e+00  1.11459299e+01 -5.71989956e+02 -1.01345126e+01
  -2.27554341e+01 -5.08577049e-01 -4.16987537e+00 -1.28789962e+00
   2.44718109e+00 -2.54496408e-02 -3.53836487e+00 -1.88174913e+01
   2.79020491e+00  2.54821210e+00 -1.88342368e-02 -3.53312599e+00
  -1.88181978e+01  2.30599022e+01  4.06969503e-01 -2.04695891e+00
   1.88936366e-02  1.11629336e+00  4.71190049e-02  2.55405757e-04
   3.31517292e-02  1.11144269e-04  2.35741027e-06]] [80227028.3104757]
Error medio Absoluto (MAE): 121314.06948173394
Root Mean Squared Error: 186261.28575138954
r2_score 0.7797929072570198
```

```
#ridge
mi_ridge2 = Ridge(alpha=5.0,fit_intercept=True)
mi_ridge2.fit(X_train2, y_train2)
```

```
X_para_ridge2 = X_test2
y_para_ridge2 = mi_ridge2.predict(X_para_ridge2)
metrica_mae_ridge2 = metrics.mean_absolute_error(y_test2, y_para_ridge2)
r2Score2 = r2_score(y_test2, y_para_ridge2)
print('Error medio Absoluto (MAE):', metrica_mae_ridge2)
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test2, y_para_ridge2)))
print('r2_score',r2Score2)
print('El modelo es: Y =', mi_ridge2.coef_,'X +', mi_ridge2.intercept_)
```

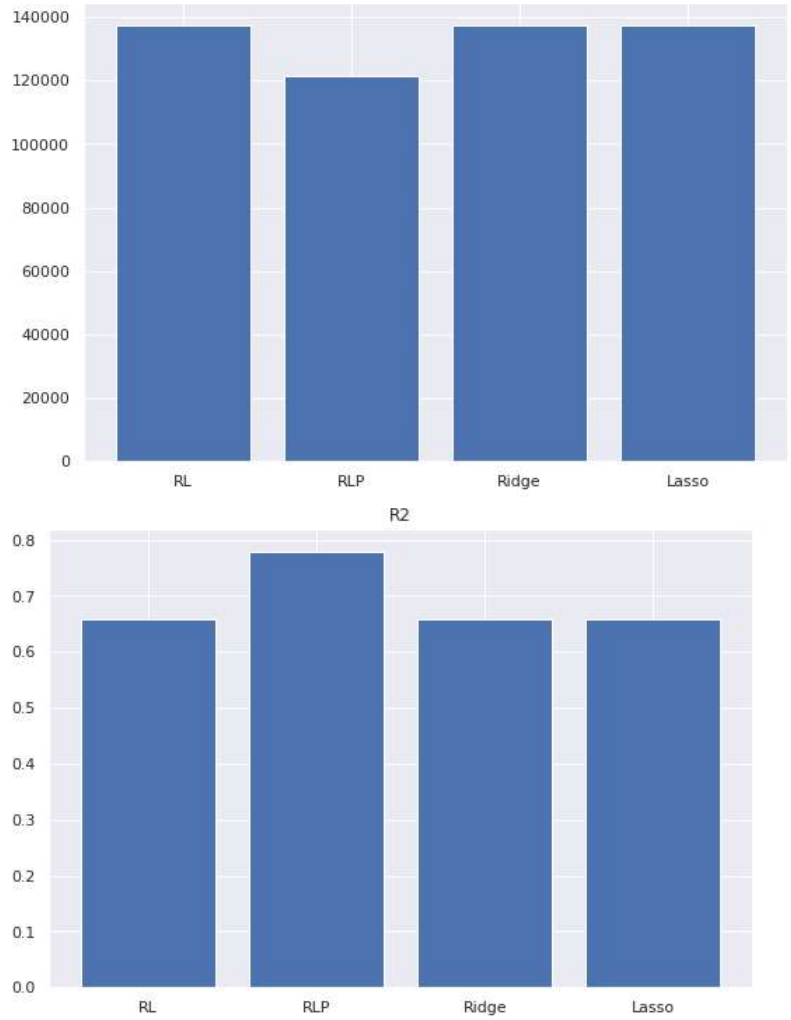
```
Error medio Absoluto (MAE): 137533.65574257114
Root Mean Squared Error: 232291.8879538845
r2_score 0.6575050300500245
El modelo es: Y = [[-3.82896171e+04  4.13914306e+04  1.08096090e+02  1.61486363e-02
   3.16788057e+04  5.31220189e+05  4.22492400e+04  2.12424198e+04
   1.19381757e+05  4.79302812e+01  6.01658037e+01 -3.54893952e+03
   1.35365987e+01  2.89064352e+01 -5.47487907e-01]] X + [6148385.38096804]
```

```
#Lasso
mi_lasso2 = Lasso(alpha=5.0,fit_intercept=True)
mi_lasso2.fit(X_train2, y_train2)
X_para_lasso2 = X_test2
y_para_lasso2 = mi_lasso2.predict(X_para_ridge2)
metrica_mae_lasso2 = metrics.mean_absolute_error(y_test2, y_para_lasso2)
r2Score2 = r2_score(y_test2, y_para_lasso2)
print('Error medio Absoluto (MAE):', metrica_mae_lasso2)
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test2, y_para_lasso2)))
print('r2_score',r2Score2)

print('El modelo es: Y =', mi_lasso.coef_,'X +', mi_lasso.intercept_)
```

```
Error medio Absoluto (MAE): 137482.30243495226
Root Mean Squared Error: 232139.17278486004
r2_score 0.6579552135202583
El modelo es: Y = [-3.15572458] X + 54.16195119377413
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_descent.py:648: ConvergenceWarning: Objective did not con
  coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
```

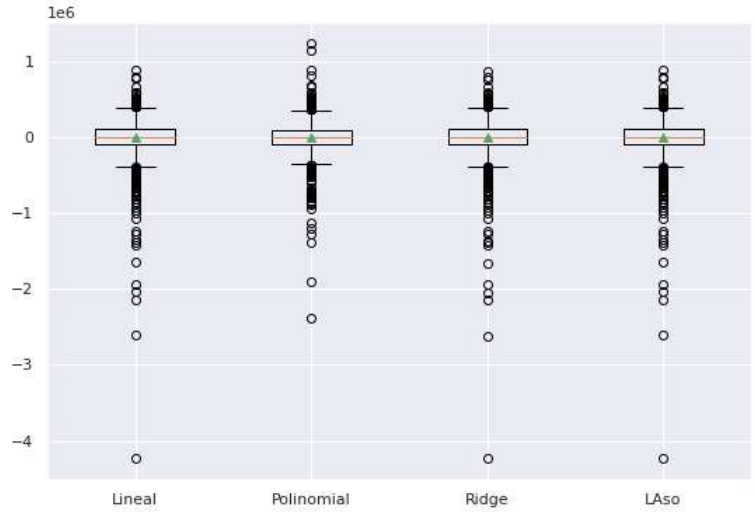
```
#graficas de residuos

lista_para_mae2=[]
lista_para_r22=[]

#Anadimos las metricas a la lista - Regresion Simple
mae_regresion_lineal_simple2 = metrics.mean_absolute_error(y_test2,y_para_regresion2)
lista_para_mae2.append(mae_regresion_lineal_simple2)
r2_regresion_lineal_simple2 = r2_score(y_test2,y_para_regresion2)
lista_para_r22.append(r2_regresion_lineal_simple2)

#Anadimos las metricas a la lista - Regresion Polinomial
mae_regresion_lineal_multiple2 = metrics.mean_absolute_error(y_test2,y_con_regresion_poli2)
lista_para_mae2.append(mae_regresion_lineal_multiple2)
r2_regresion_lineal_multiple2 = r2_score(y_test2,y_con_regresion_poli2)
lista_para_r22.append(r2_regresion_lineal_multiple2)

#Anadimos las metricas a la lista - Ridge
mae_ridge2 = metrics.mean_absolute_error(y_test2,y_para_ridge2)
lista_para_mae2.append(mae_ridge2)
r2_ridge2= r2_score(y_test2,y_para_ridge2)
lista_para_r22.append(r2_ridge2)

#Anadimos las metricas a la lista - Lasso
mae_lasso2 = metrics.mean_absolute_error(y_test2,y_para_lasso2)
lista_para_mae2.append(mae_lasso2)
r2_lasso2= r2_score(y_test2,y_para_lasso2)
lista_para_r22.append(r2_lasso2)

nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')

plt.bar(nombres, lista_para_mae2)
```

```
plt.show()

nombres=list()
nombres.append('RL')
nombres.append('RLP')
nombres.append('Ridge')
nombres.append('Lasso')

plt.bar(nombres, lista_para_r22)
plt.title('R2')
plt.show()
```




```
#Boxplot
residuo_regresion_simple2 = y_para_regresion2 - y_test2
residuo_regresion_poliforme2 = y_con_regresion_poli2 - y_test2
residuo_ridge2 = y_para_ridge2 - y_test2
residuo_lasso2 = y_para_lasso2.reshape(2162 ,1) - y_test2 #Aqui reformamos el y para lasso, porque era de 3 columnas .reshape(2162


# Unable to coerce to Series, length must be 1: given 2162, esta es la diferenfcia con el ejercicio anterior, por lo que la solucio


#lista_residuos2 = [residuo_regresion_simple2,residuo_regresion_poliforme2,residuo_ridge2,residuo_lasso2]
lista_residuos2 = [residuo_regresion_simple2.to_numpy().ravel(),residuo_regresion_poliforme2.to_numpy().ravel(),residuo_ridge2.to_n

nombres=list()
nombres.append('Lineal')
nombres.append('Polinomial')
nombres.append('Ridge')
nombres.append('LAso')

#grafica del MAE (de los cuatro métodos)
sns.set(rc={'figure.figsize':(9,6)})
```

```
22
23 plt.boxplot(lista_residuos2, labels=nombres, showmeans=True)
24 plt.show()
```



CONCLUSIONES

Para este ejercicio se utilizaron las pariciones predefiidas y de la misma formael mejor modelo sigue siendo el polinomial, ya que se observa un R2 de casi el 80%