
#**Tecnológico de Monterrey**

##**Maestría en Inteligencia Artificial Aplicada**

###**Curso:** Ciencia y Analítica de Datos

###**Profesora:** Dra. María de la Paz Rico Fernández

###**Actividad:** K-Means

###**Alumno:** Francisco Javier Ramírez Arias

###**Matrícula:** A01316379

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
! pip install haversine
```

```
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: qeds in /usr/local/lib/python3.7/dist-
packages (0.7.0)
Requirement already satisfied: fiona in /usr/local/lib/python3.7/dist-
packages (1.8.22)
Requirement already satisfied: geopandas in
/usr/local/lib/python3.7/dist-packages (0.10.2)
Requirement already satisfied: xgboost in
/usr/local/lib/python3.7/dist-packages (0.90)
Requirement already satisfied: gensim in
/usr/local/lib/python3.7/dist-packages (3.6.0)
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: quandl in
/usr/local/lib/python3.7/dist-packages (from qeds) (3.7.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-
packages (from qeds) (1.21.6)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.7/dist-packages (from qeds) (0.11.2)
Requirement already satisfied: plotly in
/usr/local/lib/python3.7/dist-packages (from qeds) (5.5.0)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.7/dist-packages (from qeds) (3.2.2)
Requirement already satisfied: statsmodels in
/usr/local/lib/python3.7/dist-packages (from qeds) (0.12.2)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.7/dist-packages (from qeds) (1.0.2)
Requirement already satisfied: pandas in
/usr/local/lib/python3.7/dist-packages (from qeds) (1.3.5)
Requirement already satisfied: pyarrow in
/usr/local/lib/python3.7/dist-packages (from qeds) (6.0.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-
packages (from qeds) (1.7.3)
Requirement already satisfied: requests in
/usr/local/lib/python3.7/dist-packages (from qeds) (2.23.0)
Requirement already satisfied: openpyxl in
/usr/local/lib/python3.7/dist-packages (from qeds) (3.0.10)
Requirement already satisfied: pandas-datareader in
/usr/local/lib/python3.7/dist-packages (from qeds) (0.9.0)
Requirement already satisfied: quantecon in
/usr/local/lib/python3.7/dist-packages (from qeds) (0.5.3)
Requirement already satisfied: cligj>=0.5 in
/usr/local/lib/python3.7/dist-packages (from fiona) (0.7.2)
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: click>=4.0 in
/usr/local/lib/python3.7/dist-packages (from fiona) (7.1.2)
Requirement already satisfied: click-plugins>=1.0 in
/usr/local/lib/python3.7/dist-packages (from fiona) (1.1.1)
Requirement already satisfied: attrs>=17 in
/usr/local/lib/python3.7/dist-packages (from fiona) (22.1.0)
Requirement already satisfied: munch in /usr/local/lib/python3.7/dist-
packages (from fiona) (2.5.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.7/dist-packages (from fiona) (57.4.0)
Requirement already satisfied: shapely>=1.6 in
/usr/local/lib/python3.7/dist-packages (from geopandas) (1.8.5.post1)
Requirement already satisfied: pyproj>=2.2.0 in
/usr/local/lib/python3.7/dist-packages (from geopandas) (3.2.1)
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: pytz>=2017.3 in
/usr/local/lib/python3.7/dist-packages (from pandas->qeds) (2022.6)
Requirement already satisfied: smart-open>=1.2.1 in
/usr/local/lib/python3.7/dist-packages (from gensim) (5.2.1)
Requirement already satisfied: branca>=0.3.0 in
/usr/local/lib/python3.7/dist-packages (from folium) (0.5.0)
Requirement already satisfied: jinja2>=2.9 in
/usr/local/lib/python3.7/dist-packages (from folium) (2.11.3)
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: numexpr in
/usr/local/lib/python3.7/dist-packages (from pyLDAvis) (2.8.4)
Requirement already satisfied: joblib in
/usr/local/lib/python3.7/dist-packages (from pyLDAvis) (1.2.0)
Requirement already satisfied: funcy in /usr/local/lib/python3.7/dist-
packages (from pyLDAvis) (1.17)
Requirement already satisfied: sklearn in
/usr/local/lib/python3.7/dist-packages (from pyLDAvis) (0.0.post1)
Requirement already satisfied: future in
/usr/local/lib/python3.7/dist-packages (from pyLDAvis) (0.16.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib->qeds) (1.4.4)
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
matplotlib->qeds) (3.0.9)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.7/dist-packages (from matplotlib->qeds)
(0.11.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1-
>matplotlib->qeds) (4.1.1)
```

```
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->qeds)
(1.24.3)
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: sympy in /usr/local/lib/python3.7/dist-
packages (from quantecon->qeds) (1.7.1)
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-
packages (from quantecon->qeds) (0.56.4)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from numba->quantecon->qeds)
(4.13.0)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in
/usr/local/lib/python3.7/dist-packages (from numba->quantecon->qeds)
(0.39.1)
Requirement already satisfied: zipp>=0.5 in
/usr/local/lib/python3.7/dist-packages (from importlib-metadata-
>numba->quantecon->qeds) (3.10.0)
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)
Requirement already satisfied: mpmath>=0.19 in
/usr/local/lib/python3.7/dist-packages (from sympy->quantecon->qeds)
(1.2.1)
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Collecting haversine
  Downloading haversine-2.7.0-py2.py3-none-any.whl (6.9 kB)
Installing collected packages: haversine
Successfully installed haversine-2.7.0
```

```
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
import seaborn as sns
from IPython.display import set_matplotlib_formats
from sklearn.datasets import make_blobs
from sklearn.metrics import pairwise_distances_argmin
from geopy.geocoders import Nominatim
```

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.

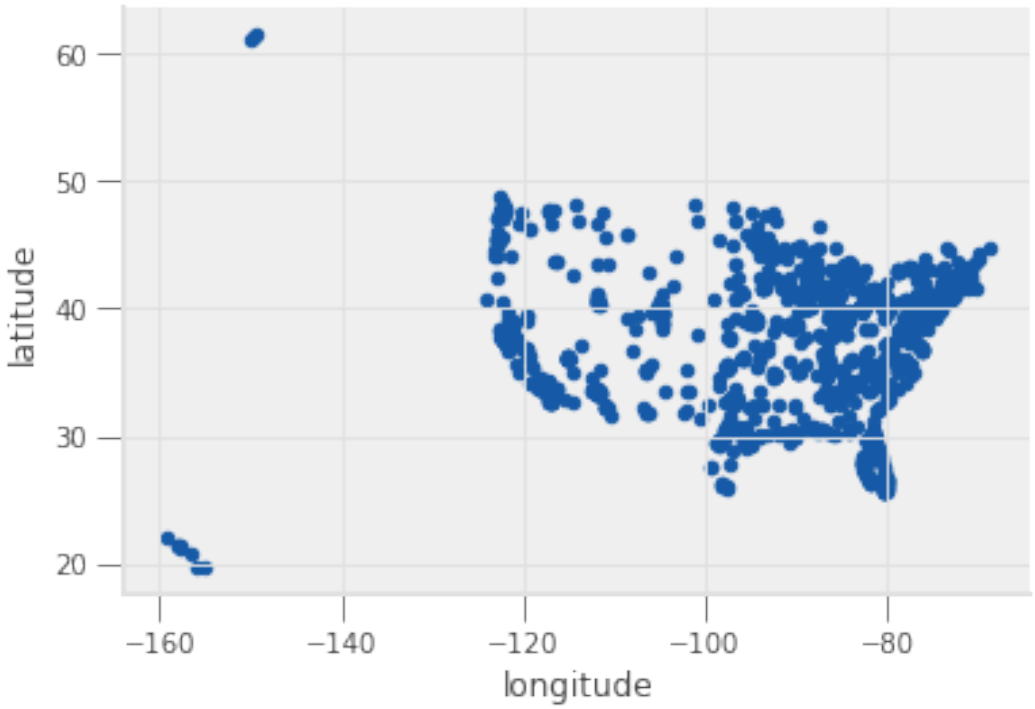
```
latlong=df[["latitude","longitude"]]
```

¡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 0x7f75a27b8510>
```




```
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.

```
import geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from shapely.geometry import Point

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

```
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)
```

```
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()
```

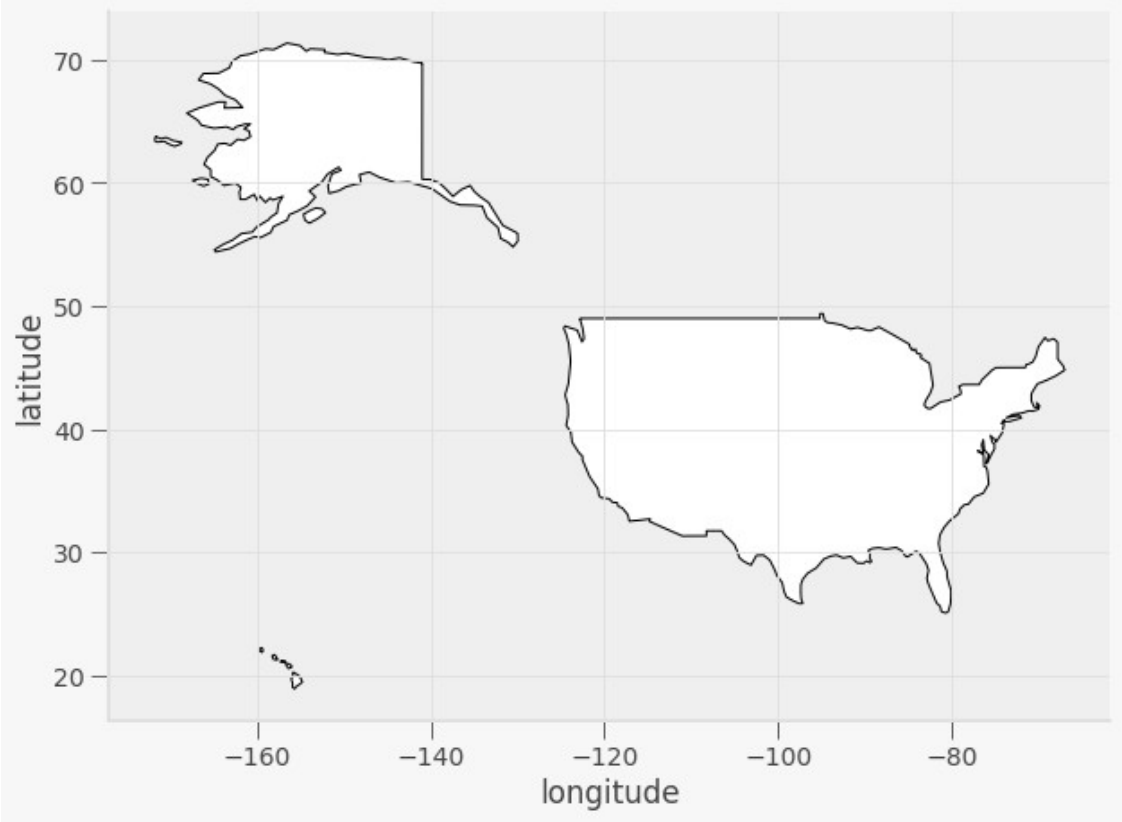
```
          pop_est      continent                      name  gdp_md_est
\
iso_a3

FJI        920938        Oceania                      Fiji      8374.0

TZA      53950935         Africa                  Tanzania    150600.0

ESH        603253         Africa                W. Sahara       906.5
```

```
CAN      35623680  North America                  Canada   1674000.0

USA     326625791  North America  United States of America  18560000.0


                                                  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...
```

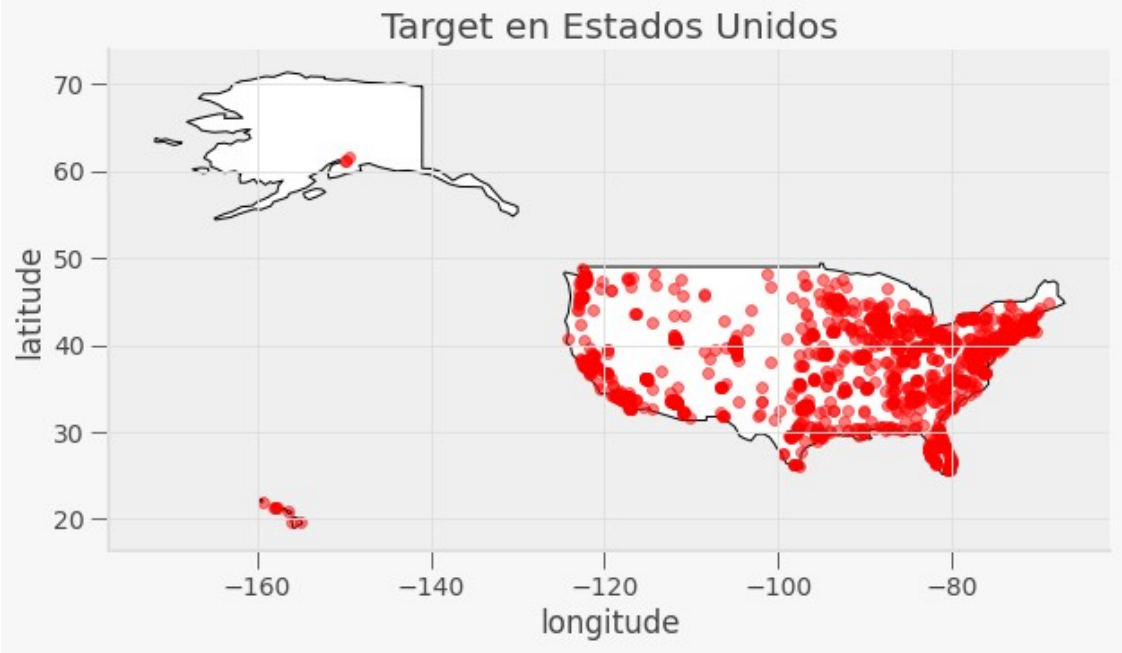
```
#graficar el mapa
world.name.unique()
```

```
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.', '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.

```
import random
import numpy as np
import pandas as pd
import scipy.spatial
from haversine import haversine
from sklearn.cluster import KMeans
```

```
latlong.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1839 entries, 0 to 1838
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   latitude   1839 non-null   float64
 1   longitude  1839 non-null   float64
dtypes: float64(2)
memory usage: 28.9 KB
```

```
#Ajuste del modelo de vecinos cercanos
#Se prueba el ajuste para obtener un
#numero especifico de vecinos cercanos
```

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(latlong)
```

```
KMeans(n_clusters=3)
```

```
#Obtenemos el punto centras de cluster
centers = kmeans.cluster_centers_
print("'kmeans' model intances is trained and the cluster centroids
are stored in 'centers'")
print("Number of iterations the model run to converge :
{}".format(kmeans.n_iter_))
```

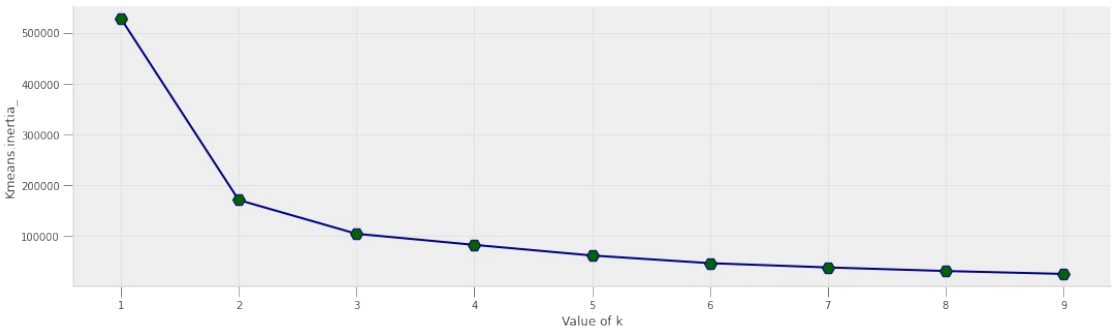
```
'kmeans' model intances is trained and the cluster centroids are
stored in 'centers'
Number of iterations the model run to converge : 9
```

```
#Se despliegan los puntos centrales de los agrupamientos
#Estas son las coordenada de los almacenes
centers
```

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

```
#Se utiliza el metodo de "Elbow", para obtener
#el numero de almacenes adecuado para abastecer
#a las diferentes tiendas
sum_square = {}

for k in range(1, 10):
  kmeans = KMeans(n_clusters=k).fit(latlong)
  sum_square[k] = kmeans.inertia_

fig,ax = plt.subplots(figsize=(18,5))
ax.plot(list(sum_square.keys()),
list(sum_square.values()),ls='-',marker='H', color='DarkBlue', lw=2,
        markersize=12,markerfacecolor = 'DarkGreen')
ax.set_xlabel("Value of k")
ax.set_ylabel("Kmeans.inertia_")
```

```
Text(0, 0.5, 'Kmeans.inertia_')
```



```
#Con el valor optimo de K, se entrena de nuevo el
#modelo y se obtienen las coordenas de los centro
#de distribución
kmeans = KMeans(n_clusters=4)
kmeans.fit(latlong)

centers = kmeans.cluster_centers_
centers
```

```
array([[  36.557344  ,  -84.8119487 ],
       [  37.57757741, -119.14645561],
       [  36.99505521,  -96.97278809],
       [  40.33224701,  -75.27553283]])
```

```
#Se realiza un conteopara determinar
#cuantas tiendas atendea cada centro de distribución
conteo=kmeans.labels_
conteo
```

```
array([0, 0, 0, ..., 2, 2, 1], dtype=int32)
```

```
np.bincount(conteo)
```

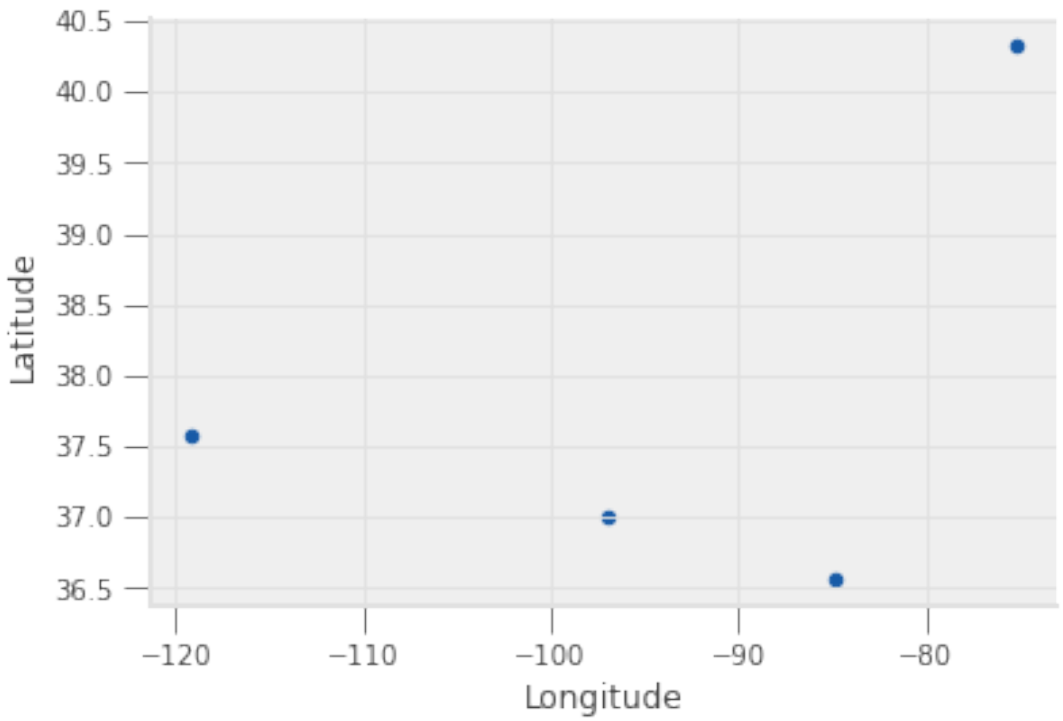
```
array([615, 369, 401, 454])
```

```
#Se obtienen las nuevas coordenas
new_latlong = pd.DataFrame(centers, columns =['Latitude','Longitude'])
new_latlong
```

```
    Latitude   Longitude
0  36.557344  -84.811949
1  37.577577 -119.146456
2  36.995055  -96.972788
3  40.332247  -75.275533
```

```
#Se graficas en una grafica de dispersion
#al parecer no tienen mucho sentido
new_latlong.plot.scatter( "Longitude","Latitude")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f75a2b3eb90>
```



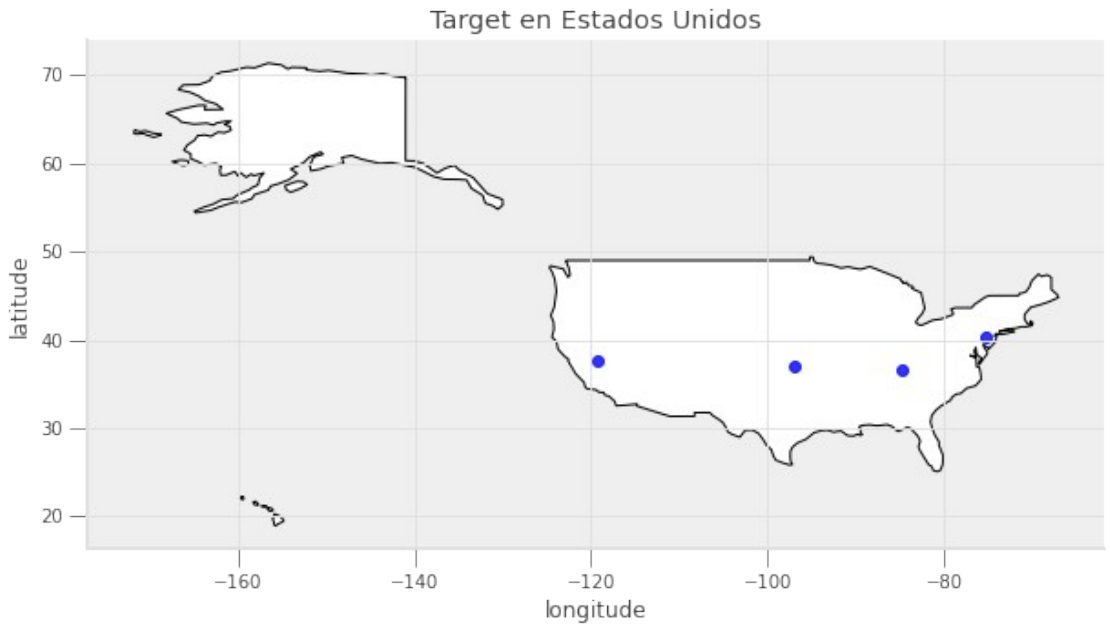

```
#Se agregan datos ala nueva tabla de coordenadas
new_latlong["Coordinates"] = list(zip(new_latlong.Longitude,
new_latlong.Latitude))
new_latlong["Coordinates"] = new_latlong["Coordinates"].apply(Point)
new_latlong.head()
```

```
    Latitude   Longitude
Coordinates
0  36.557344  -84.811949   POINT (-84.81194869642276
36.55734399853659)
1  37.577577 -119.146456  POINT (-119.14645561138212
37.57757741219512)
2  36.995055  -96.972788         POINT (-96.97278808975
36.99505520875)
3  40.332247  -75.275533   POINT (-75.27553282527472
40.33224701032967)
```

```
#Se genera el nuevo dataframe de datos geograficos con las coordenadas
nuevas
gedf = gpd.GeoDataFrame(new_latlong, geometry="Coordinates")
gedf.head()
```

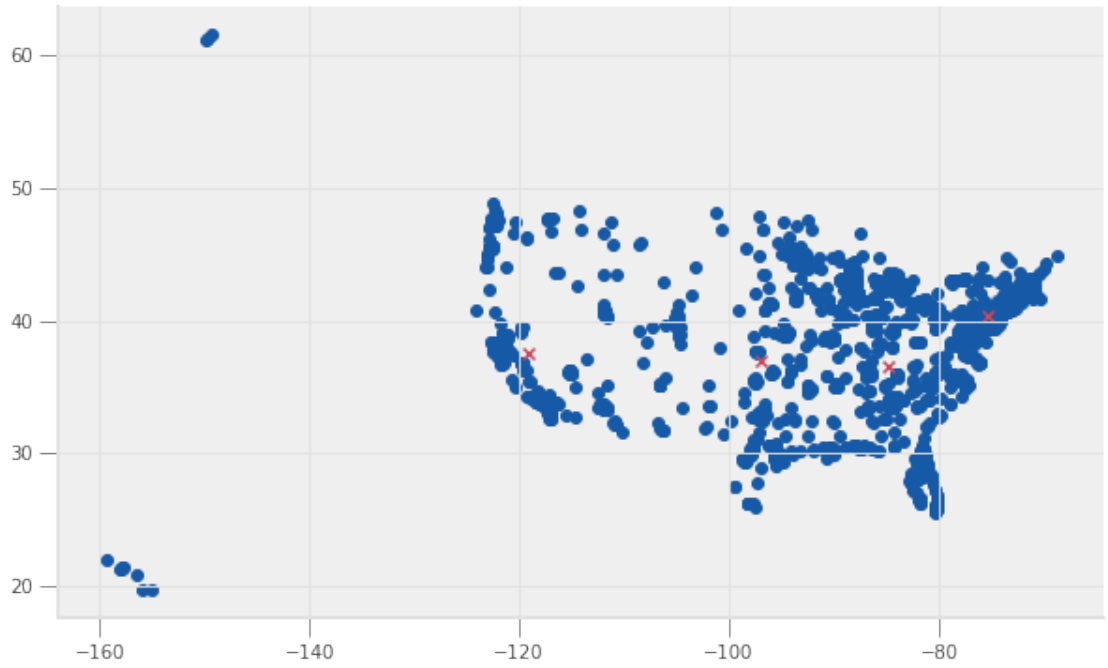
```
    Latitude   Longitude                  Coordinates
0  36.557344  -84.811949   POINT (-84.81195 36.55734)
1  37.577577 -119.146456  POINT (-119.14646 37.57758)
2  36.995055  -96.972788   POINT (-96.97279 36.99506)
3  40.332247  -75.275533   POINT (-75.27553 40.33225)
```

```
#Se grafica solo los almacenes
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.
gedf.plot(ax=gax, color='blue', alpha = 0.8)

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()
```




```
#Posicion de las tienda y de los almacenes
#Grafica de dispersión
fig, ax = plt.subplots(figsize=(10, 6))
plt.scatter(latlong.longitude,latlong.latitude)
plt.scatter(new_latlong.Longitude, new_latlong.Latitude,marker='x')
```

```
<matplotlib.collections.PathCollection at 0x7f75a27cd690>
```




```
#Ubicacion o dirección de los almacenes
geolocator = Nominatim(user_agent="geoapiExercises")
location_1 = geolocator.reverse("36.557344, -84.811949")
print(location_1)
location_2 = geolocator.reverse("37.577577, -119.146456")
print(location_2)
location_3 = geolocator.reverse("36.995055, -96.972788")
print(location_3)
location_4 = geolocator.reverse("40.332247, -75.275533")
print(location_4)
```

```
Spaughs Ridge Road, Pickett County, Tennessee, United States
Mammoth Trail, Madera County, California, United States
332nd Road, Cowley County, Kansas, United States
Diamond Street, Hilltown Township, Bucks County, Pennsylvania, 18962,
United States
```

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?

1. 36.557344 -84.811949 Tennessee 615
2. 37.577577 -119.146456 California 309
3. 36.995055 -96.972788 Kansas 401
4. 40.332247 -75.275533 Pensilvania 454

¿Sabes a qué distancias estará? La pregunta no es muy clara o especifica, pero cada la distancia entre el almacén de California y Kansas es de 1536 millas, la distancia del almacén de Kansas al de Tennessee es de 878 millas, y del de Tennessee a Pensilvania de 833 millas. Observamos que la distancia es equidistante entre tres de los cuatro puntos. Con la excepción de la distancia de California a Kansas, pero se justifica de cierta manera porque el número de almacenes que se encuentra en área geográfica es menor.

¿Cómo elegiste el número de almacenes? Utilizando algoritmo de vecinos cercanos, específicamente método de codo "elbow", este nos indica gráficamente que el número ideal de almacenes se encuentra entre 4 y 5, para nuestro caso nosotros seleccionamos 4, nos pareció más adecuado que cuatro las distancias entre los puntos son similares, lo que pensamos minimizara los costos en logística.

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?

1. **Geopy** esta librería nos permite obtener la localización o dirección una vez que obtenemos las coordenadas de latitud y longitud, utilizada en la práctica.

2. **Plotly/Plotly Express** esta librería nos permite colocar líneas en los mapas, seleccionar áreas, colocar burbujas, sobreponer mapas de densidad, colocar información sobre punto de interés en los mapas, gráficas de dispersión sobre los mapas, entre algunas otras amenidades. Es una API de alto nivel.

3. **Folium** esta librería permite la manipulación en Python de forma interactiva los mapas o datos sobre estos. La librería permite dado un par de coordenadas generar el mapa de la zona donde esta se encuentra, y salvarla en archivo html, permite colocar diferentes tipos de marcadores sobre el mapa, diferentes tipos de vistas de los mapas, permite seleccionar zonas geográficas y etiquetarlas en burbujas, permite colocar transparencias sobre el mapa y realizar mapas de calor en una determinada zona geográfica. Además cuenta con funciones que permiten la personalización de la vista de los mapas.

¿Consideras importante que se grafique en un mapa?, ¿por qué? Si considero que es importante que estos datos se grafiquen en un mapa porque nos ayuda ver la distribución de los datos, a identificar en que regiones se concentra mayor cantidad de datos, observar en que regiones hay una distribución menor de datos, identificar datos que pensamos son outliers, y no lo son, a identificar áreas geográficas en donde no se presentan ningún dato.

Conclusiones El algoritmo de vecinos cercanos son permite realizar agrupaciones de datos y encontrar el centro de estos datos, esto lo logra por medio de la distancia entre los vecinos cercanos. Este algoritmo en comparación con los algoritmos de clasificación se encuentra en la categoría de los algoritmos sin supervisión, el algoritmo se encarga de realizar el etiquetado de las muestras, en comparación con los algoritmos supervisados en donde es necesario que los datos se encuentren etiquetados. El ejercicio desarrollado es un gran ejemplo de la capacidad que tienen este algoritmo en particular, sobre todo cuando

necesitamos determinar algo y solamente tenemos nuestros datos sin etiquetar, vecinos cercanos es la opción adecuada. Este algoritmo se complementa con la estrategia de búsqueda de "elbow", con la cual se determina el número "K" adecuado para el entrenamiento del modelo.