Clustering de geopoints por DBSCAN

Objetivos:

- Aplicar el algoritmo DBSCAN para obtener zonas de alta densidad de accidentes vehiculares en la CDMX.

Desarrollo:

- Indicar un radio $\varepsilon = 500$m que tendrá la vecindad.
- Indicar el número mínimo de accidentes que tendrá la vecindad $min = 5$ para ser conciderada una coordenada **significativa**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
df = pd.read_csv("data/incidentes-viales-c5-limpio.csv", sep="$", index_col=0)
```

```
df.tail()
```

	folio	codigo_cierre	delegacion_inicio	incidente_c4	latitud
693675	C5/210228/09218	N	MIGUEL HIDALGO	accidente-motociclista	19.392430
693688	C5/210228/09309	N	IZTAPALAPA	accidente-choque sin lesionados	19.349940
693689	C5/210228/09401	N	GUSTAVO A. MADERO	lesionado-atropellado	19.491660

Conversión de GPS a matriz de distancias y clustering con DBSCAN

```
from sklearn.cluster import DBSCAN
from geopy.distance import great_circle
from shapely.geometry import MultiPoint
```

```
coords = df[['latitud', 'longitud']].values
kms_per_radian = 6371.0088
epsilon = 0.5 / kms_per_radian
```

```
db = DBSCAN(eps=epsilon, min_samples=5, algorithm='ball_tree', metric='haversine').fi
cluster_labels = db.labels_
num_clusters = len(set(cluster_labels)) # Number of cluster with no noise
# num_clusters = len(set(labels)) - (1 if -1 in labels else 0) # Number of cluster wi
clusters = pd.Series([coords[cluster_labels == n] for n in range(num_clusters)])
print('Number of clusters: {}'.format(num_clusters))
```

```
Number of clusters: 44
```

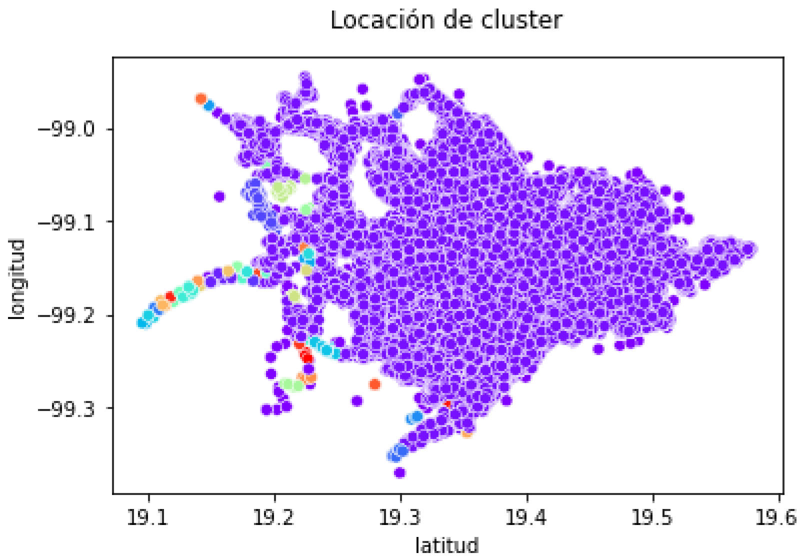
```
fig = plt.figure()
ax = fig.add_subplot()

ax.set_title('Locación de cluster', pad=15)
ax.set_xlabel('latitud')
ax.set_ylabel('longitud')

sns.scatterplot(df['latitud'], df['longitud'], ax=ax, hue=cluster_labels, palette='ra
# sns.scatterplot(labels[:,0], labels[:,1], ax=ax, s=100, color='black');

ax.get_legend().remove()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
  FutureWarning
```



Encontrar el punto más cercano a cada cluster

```
def get_centermost_point(cluster):
    centroid = (MultiPoint(cluster).centroid.x, MultiPoint(cluster).centroid.y)
    centermost_point = min(cluster, key=lambda point: great_circle(point, centroid).m
    return tuple(centermost_point)
```

```
centermost_points = clusters[:len(clusters)-1].map(get_centermost_point)
```

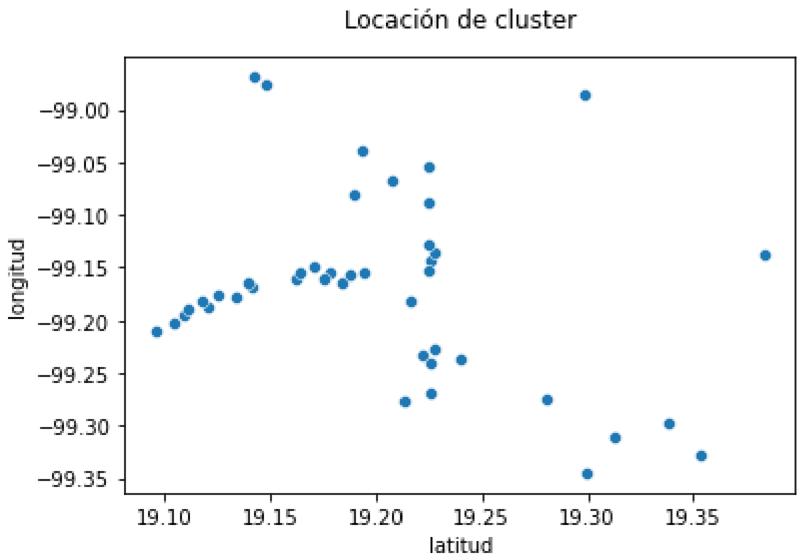
```
lats, lons = zip(*centermost_points)
rep_points = pd.DataFrame({'latitud':lats, 'longitud':lons})


fig = plt.figure()
ax = fig.add_subplot()

ax.set_title('Locación de cluster', pad=15)
ax.set_xlabel('latitud')
ax.set_ylabel('longitud')

sns.scatterplot(rep_points['latitud'], rep_points['longitud'], ax=ax);
# sns.scatterplot(labels[:,0], labels[:,1], ax=ax, s=100, color='black');
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
  FutureWarning
```




```
# Guardamos las coordenadas de los puntos con mayor densidad
rep_points.to_csv("most_dangerous_geop.csv")
```

Graficar Gmap

```
import gmaps
```

```
f = open("key.txt")
api_token = f.read()
gmaps.configure(api_key=api_token)
```

Heatmap

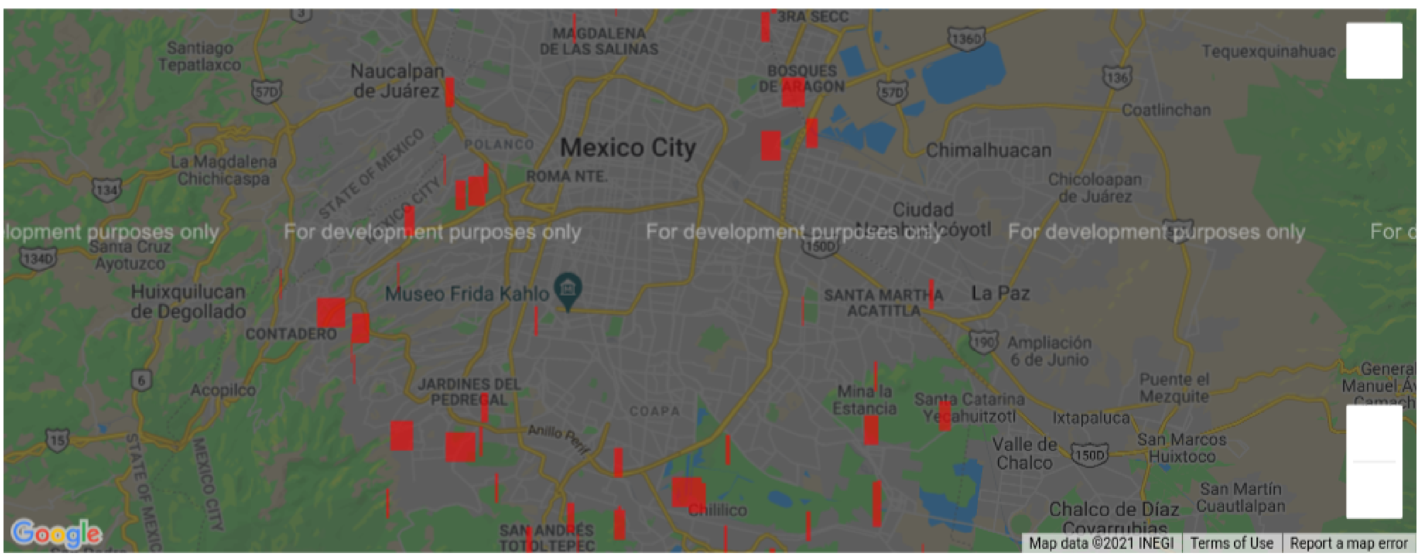
```
#Cargamos labels
dbscan_labels = pd.read_csv("data/dbscan_labels.csv", sep=",", index_col=0)


def labels_to_weights(label):
    if label == -1:
        return 0
    elif label == 0:
        return 1
    else:
        return label


df['labels'] = dbscan_labels['label'].map(labels_to_weights)


locations = df[['latitud', 'longitud']]
weights = dbscan_labels
fig = gmaps.figure()
fig.add_layer(gmaps.heatmap_layer(locations, weights=df['labels']))
fig
```

```
Figure(layout=FigureLayout(height='420px'))
```

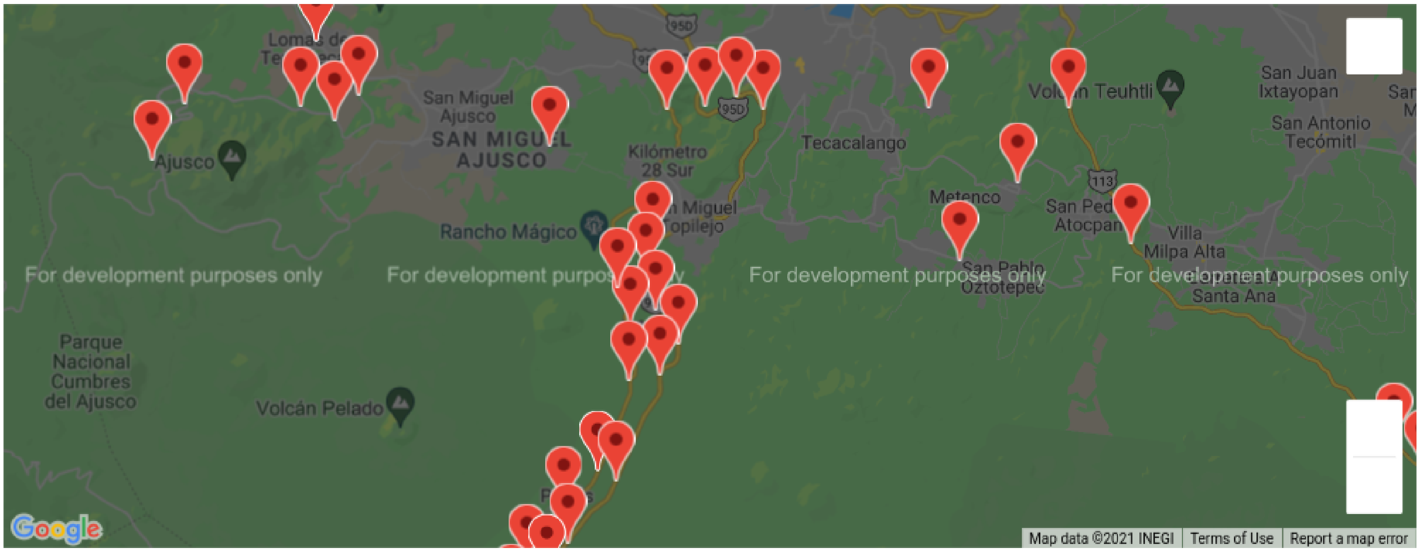


Top puntos significativos

```
#Cargamos coordenadas significativas
df = pd.read_csv("data/most_dangerous_geop.csv", sep=",", index_col=0)


top_dangerous_geopoints=tuple(zip(df['latitud'], df['longitud']))
fig = gmaps.figure()
markers = gmaps.marker_layer(top_dangerous_geopoints)
fig.add_layer(markers)
fig
```

```
Figure(layout=FigureLayout(height='420px'))
```



Se observa que los puntos con mayor densidad de accidentes ocurren en carreteras.

Reverse geocoding

```
import googlemaps
```

```
f = open("key.txt")
api_token = f.read()
gsdk = googlemaps.Client(key=api_token)
```

Double-click (or enter) to edit

```
reverse_geocode_result = gsdk.reverse_geocode(top_dangerous_geopoints[0])
reverse_geocode_result
```

```
---------------------------------------------------------------------------
ApiError                                  Traceback (most recent call last)
<ipython-input-46-7f2e89352a04> in <module>
----> 1 reverse_geocode_result =
gsdk.reverse_geocode(top_dangerous_geopoints[0])
      2 reverse_geocode_result

~/github/data_proyects/BEDU-M4-DataAnalysisProject-
CarAccidents/env/lib/python3.9/site-packages/googlemaps/client.py in
wrapper(*args, **kwargs)
    416     def wrapper(*args, **kwargs):
    417         args[0]._extra_params = kwargs.pop("extra_params", None)
--> 418         result = func(*args, **kwargs)
    419         try:
    420             del args[0]._extra_params

~/github/data_proyects/BEDU-M4-DataAnalysisProject-
CarAccidents/env/lib/python3.9/site-packages/googlemaps/geocoding.py in
reverse_geocode(client, latlng, result_type, location_type, language)
    107         params["language"] = language
    108
--> 109     return client._request("/maps/api/geocode/json",
params).get("results", [])

~/github/data_proyects/BEDU-M4-DataAnalysisProject-
```

Con una geo decodificación se podría obtener dada un par de coordenas una descripción de las direcciones y avenidas. Sin embargo, ésta información tiene un costo asociado con el servidor de google. De obtener la información se podrían usar algoritmos de NPL

```
--> 313                 result = self._get_body(response)



~/github/data_proyects/BEDU-M4-DataAnalysisProject-
CarAccidents/env/lib/python3.9/site-packages/googlemaps/client.py in
_get_body(self, response)
```