Density-Based Clustering

Objetivos

- Utilizar DBSCAN
- Utilizar Matplotlib para graficar clusters

La mayoría de las técnicas tradicionales de clustering, como k-means, jerárquico y fuzzy clustering pueden ser utilizadas para agrupar datos sin supervisión.

Sin embargo, cuando se aplican a tareas con clusteres de forma arbitraria, o con clusteres dentro de clusteres, las técnicas tradicionales podrían no lograr buenos resultados. Esto es, elementos en el mismo cluster podrían no compartir suficientes similaridades o la performance ser mala.

Adicionalmente, DBSCAN ubica regiones de alta densidad que están separadas de regiones de baja densidad. La densidad en este contexto, es definida como el número de puntos dentro de un radio especificado.

In [39]:
```
# Notice: For visualization of map, you need basemap package.
# if you dont have basemap install on your machine, you can use the following line to
#!conda install -c conda-forge  basemap matplotlib==3.1 -y
# Notice: you maight have to refresh your page and re-run the notebook after installa
```

In [40]:
```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

Generación de datos

La función de abajo generará puntos de datos y requiere las siguientes entradas:

- **centroidLocation**: Coordenadas de los centroides que generarán los datos aleatorios.
  - Ejemplo: input: [[4,3], [2,-1], [-1,4]]
- **numSamples**: El número de puntos de datos que queremos generar dividido sobre el número de centroides (# de centroides definido en centroidLocation)
  - Ejemplo: 1500
- **clusterDeviation**: La desviación estándar entre clusters. Mientras más grande sea el número más separados estarán.
  - Ejemplo: 0.5

In [41]:
```
def createDataPoints(centroidLocation, numSamples, clusterDeviation):
    # Creamos puntos de datos aleatorios y los almacenamos en la matriz de caracterís
    X, y = make_blobs(n_samples=numSamples, centers=centroidLocation,
                                cluster_std=clusterDeviation)

    # Estandarizamos las características removiendo la media y escalando a varianza u
    X = StandardScaler().fit_transform(X)
    return X, y
```

Usamos **createDataPoints** con las 3 entradas y almacenamos la salida en las variables X e y.

In [42]:
```
X, y = createDataPoints([[4,3], [2,-1], [-1,4]] , 1500, 0.5)
```

Modelando

DBSCAN significa "Density-Based Spatial Clustering of Applications with Noise". Es una de las técnicas más comunes que trabaja basado en densidades de un objeto. La idea es que si un punto particular pertenece a un cluster, debe estar cerca a muchos otros puntos en ese cluster.

Funciona basado en 2 parámetros: Epsilon y Minimum Points.

Epsilon determina un radio especificado, que si incluye un número suficiente de puntos, es llamado área densa **minimumSamples** determina el mínimo número de puntos de datos que queremos en una vecindad para definir el cluster

In [43]:
```
epsilon = 0.3
minimumSamples = 7
db = DBSCAN(eps=epsilon, min_samples=minimumSamples).fit(X)
labels = db.labels_
labels
```

Out[43]:
```
array([0, 0, 0, ..., 2, 0, 0], dtype=int64)
```

Distinguiendo outliers

Reemplacemos todos los elementos con "True" en core_samples_mask que están el cluster y con "False" si los puntos son outliers.

In [44]:
```
# Creamos las etiquetas de booleanos usando las etiquetas desde db
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
core_samples_mask
```

Out[44]:
```
array([ True,  True,  True, ...,  True,  True,  True])
```

In [45]:
```
# Número de clusters en etiquetas, ignoramos el ruido si está presente
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_clusters_
```

Out[45]:
```
3
```

In [46]:
```
# Removemos la repetición en las etiquetas convirtiendo a un conjunto
unique_labels = set(labels)
unique_labels
```

Out[46]:
```
{0, 1, 2}
```

Visualización de datos

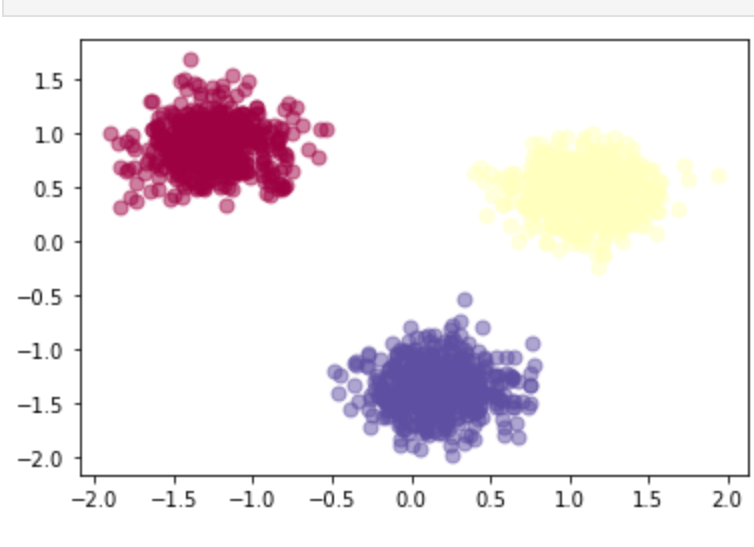
In [47]:
```
# Creamos colores para los clusters
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
```

In [48]:
```
# Imprimimos los puntos con colores
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Negro para el ruido
        col = 'k'

    class_member_mask = (labels == k)

    # Graficamos los puntos de datos que está en el cluster
    xy = X[class_member_mask & core_samples_mask]
    plt.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)

    # Graficamos los outliers
    xy = X[class_member_mask & ~core_samples_mask]
    plt.scatter(xy[:, 0], xy[:, 1],s=50, c=[col], marker=u'o', alpha=0.5)
```

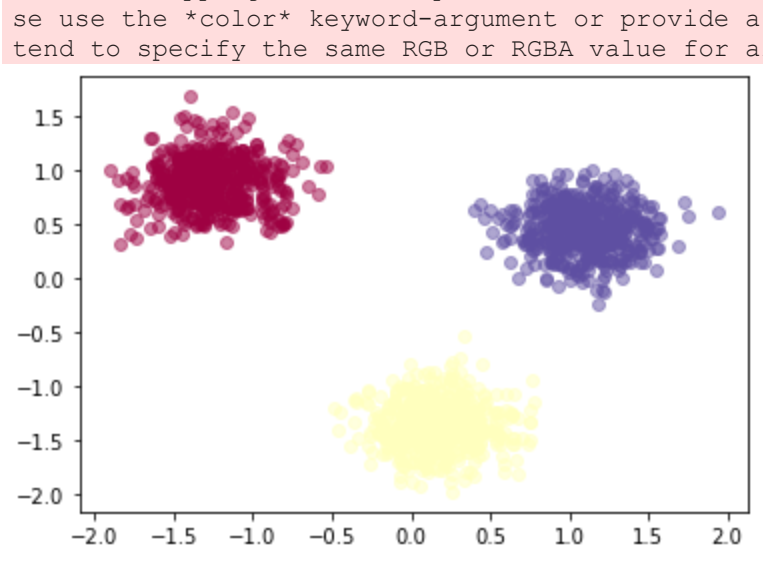


Práctica

Para entender mejor las diferencias entre clustering particional y clustering basado en densidades intente clusterizar el dataset de arriba en 3 clusters usando k-means.

In [49]:
```
from sklearn.cluster import KMeans
k = 3
k_means3 = KMeans(init = "k-means++", n_clusters = k, n_init = 12)
k_means3.fit(X)
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot(1, 1, 1)
for k, col in zip(range(k), colors):
    my_members = (k_means3.labels_ == k)
    plt.scatter(X[my_members, 0], X[my_members, 1],  c=col, marker=u'o', alpha=0.5)
plt.show()
```

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



Weather Station Clustering utilizando DBSCAN & scikit-learn

DBSCAN es especialmente bueno para tareas como identificación de clases en un contexto espacial. Lo maravilloso de DBSCAN es que puede encontrar clusters de forma arbitraria sin ser afectado por el ruido. Por ejemplo, el siguiente ejemplo clusteriza las ubicaciones de las estaciones de tiempo en Canadá. DBSCAN puede ser utilizado aquí para por ejemplo encontrar grupos de estaciones que muestren las mismas condiciones climáticas. No sólo encuentra clusters de forma arbitraria diferente, también puede encontrar la parte más densa de las muestras ignorando áreas menos densas o ruidos.

Trabajaremos de acuerdo al siguiente workflow:

1. Cargar los datos

- Overview de los datos
- Limpieza de los datos
- Selección de datos
- Clustering

Acerca del dataset

Environment Canada Monthly Values for July - 2015

Name in the table	Meaning
Stn_Name	**Station Name**
Lat	**Latitude (North+, degrees)**
Long	**Longitude (West - , degrees)**
Prov	Province
Tm	Mean Temperature (°C)
DwTm	Days without Valid Mean Temperature
D	Mean Temperature difference from Normal (1981-2010) (°C)
Tx	Highest Monthly Maximum Temperature (°C)
DwTx	Days without Valid Maximum Temperature
Tn	Lowest Monthly Minimum Temperature (°C)
DwTn	Days without Valid Minimum Temperature
S	Snowfall (cm)
DwS	Days without Valid Snowfall
S%N	Percent of Normal (1981-2010) Snowfall
P	**Total Precipitation (mm)**
DwP	Days without Valid Precipitation
P%N	Percent of Normal (1981-2010) Precipitation
S_G	Snow on the ground at the end of the month (cm)
Pd	Number of days with Precipitation 1.0 mm or more
BS	Bright Sunshine (hours)
DwBS	Days without Valid Bright Sunshine
BS%	Percent of Normal (1981-2010) Bright Sunshine
HDD	Degree Days below 18 °C
CDD	Degree Days above 18 °C
Stn_No	Climate station identifier (first 3 digits indicate drainage basin, last 4 characters are for sorting alphabetically).
NA	Not Available

1. Descargando los datos

In [50]:
```
#!wget -O weather-stations20140101-20141231.csv https://cf-courses-data.s3.us.cloud-o

import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevelope
filename = 'weather-stations20140101-20141231.csv'
urllib.request.urlretrieve(url, filename)
```

Out[50]:
```
('weather-stations20140101-20141231.csv',
 <http.client.HTTPMessage at 0x24e52890e20>)
```

2. Cargando el dataset

Importaremos el .csv y crearemos columnas para year, month y day.

In [51]:
```
import csv
import pandas as pd
import numpy as np

filename='weather-stations20140101-20141231.csv'

#Read csv
pdf = pd.read_csv(filename)
pdf.head(5)
```

Out[51]:

	Stn_Name	Lat	Long	Prov	Tm	DwTm	D	Tx	DwTx	Tn	...	DwP	P%N	S_G	Pd	BS	DwBS	BS%	HDD	CDD	Stn_No
0	CHEMAINUS	48.935	-123.742	BC	8.2	0.0	NaN	13.5	0.0	1.0	...	0.0	NaN	0.0	12.0	NaN	NaN	NaN	273.3	0.0	1011500
1	COWICHAN LAKE FORESTRY	48.824	-124.133	BC	7.0	0.0	3.0	15.0	0.0	-3.0	...	0.0	104.0	0.0	12.0	NaN	NaN	NaN	307.0	0.0	1012040
2	LAKE COWICHAN	48.829	-124.052	BC	6.8	13.0	2.8	16.0	9.0	-2.5	...	9.0	NaN	NaN	11.0	NaN	NaN	NaN	168.1	0.0	1012055
3	DISCOVERY ISLAND	48.425	-123.226	BC	NaN	NaN	NaN	12.5	0.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1012475
4	DUNCAN KELVIN CREEK	48.735	-123.728	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573

5 rows × 25 columns

3. Limpieza

Removamos las filas que no tengan ningún valor en el campo **Tm**.

In [52]:
```
pdf = pdf[pd.notnull(pdf["Tm"])]
pdf = pdf.reset_index(drop=True)
pdf.head(5)
```

Out[52]:

	Stn_Name	Lat	Long	Prov	Tm	DwTm	D	Tx	DwTx	Tn	...	DwP	P%N	S_G	Pd	BS	DwBS	BS%	HDD	CDD	Stn_No
0	CHEMAINUS	48.935	-123.742	BC	8.2	0.0	NaN	13.5	0.0	1.0	...	0.0	NaN	0.0	12.0	NaN	NaN	NaN	273.3	0.0	1011500
1	COWICHAN LAKE FORESTRY	48.824	-124.133	BC	7.0	0.0	3.0	15.0	0.0	-3.0	...	0.0	104.0	0.0	12.0	NaN	NaN	NaN	307.0	0.0	1012040
2	LAKE COWICHAN	48.829	-124.052	BC	6.8	13.0	2.8	16.0	9.0	-2.5	...	9.0	NaN	NaN	11.0	NaN	NaN	NaN	168.1	0.0	1012055
3	DUNCAN KELVIN CREEK	48.735	-123.728	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573
4	ESQUIMALT HARBOUR	48.432	-123.439	BC	8.8	0.0	NaN	13.1	0.0	1.9	...	8.0	NaN	NaN	12.0	NaN	NaN	NaN	258.6	0.0	1012710

5 rows × 25 columns

4. Visualización

Visualización de estaciones en el mapa usando el paquete basemap. La herramienta basemap de matplotlib es una librería para realizar gráficos 2D sobre mapas en Python. Basemap no realiza gráficas por cuenta propia, pero brinda facilidades para transformar las coordenadas en proyecciones de mapa.

Observe que el tamaño de cada punto de datos representa el promedio de la máxima temperatura para cada estación en el año.

In [54]:
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

llon=-140
ulon=-50
llat=40
ulat=65

pdf = pdf[(pdf['Long'] > llon) & (pdf['Long'] < ulon) & (pdf['Lat'] > llat) &(pdf['La

my_map = Basemap(projection='merc',
            resolution = 'l', area_thresh = 1000.0,
            llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (

my_map.drawcoastlines()
my_map.drawcountries()
# my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To collect data based on stations

xs,ys = my_map(np.asarray(pdf.Long), np.asarray(pdf.Lat))
pdf['xm']= xs.tolist()
pdf['ym'] =ys.tolist()

#Visualization1
for index,row in pdf.iterrows():
#   x,y = my_map(row.Long, row.Lat)
   my_map.plot(row.xm, row.ym,markerfacecolor =([1,0,0]),  marker='o', markersize= 5,
#plt.text(x,y,stn)
plt.show()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-54-6b98a7110c83> in <module>
----> 1 from mpl_toolkits.basemap import Basemap
      2 import matplotlib.pyplot as plt
      3 from pylab import rcParams
      4 get_ipython().run_line_magic('matplotlib', 'inline')
      5 rcParams['figure.figsize'] = (14,10)

ModuleNotFoundError: No module named 'mpl_toolkits.basemap'
Building graph of deps:   0%|          | 0/5 [00:00<?, ?it/s]
Examining @/win-64::__win==0=0:   0%|          | 0/5 [00:00<?, ?it/s]
Examining @/win-64::__archspec==1=x86_64:  20%|##        | 1/5 [00:00<?, ?it/s]
Examining python=3.8:  40%|####      | 2/5 [00:00<?, ?it/s]
Examining basemap:  60%|######    | 3/5 [00:00<00:00,  7.72it/s]
Examining basemap:  80%|########  | 4/5 [00:00<00:00, 10.30it/s]
Examining matplotlib==3.1:  80%|########  | 4/5 [00:02<00:00, 10.30it/s]
Examining matplotlib==3.1: 100%|##########| 5/5 [00:02<00:00,  1.52it/s]


Determining conflicts:   0%|          | 0/5 [00:00<?, ?it/s]
Examining conflict for python basemap matplotlib:   0%|          | 0/5 [00:00<?, ?it/
s]
Examining conflict for python basemap:  20%|##        | 1/5 [00:00<00:01,  2.18it/s]

Examining conflict for python basemap:  40%|####      | 2/5 [00:00<00:00,  4.36it/s]
Examining conflict for basemap matplotlib:  40%|####      | 2/5 [00:00<00:00,  4.36it/
s]
Examining conflict for basemap matplotlib:  60%|######    | 3/5 [00:00<00:00,  3.93it/
s]



UnsatisfiableError: The following specifications were found
to be incompatible with the existing python installation in your environment:

Specifications:

  - matplotlib==3.1 -> python[version='>=3.6,<3.7.0a0|>=3.7,<3.8.0a0']

Your python: python=3.8

If python is on the left-most side of the chain, that's the version you've asked for.
When python appears to the right, that indicates that the thing on the left is somehow
not available for the python version you are constrained to. Note that conda will not
change your python version to a different minor version unless you explicitly specify
that.

The following specifications were found to be incompatible with each other:

Output in format: Requested package -> Available versions

Package ca-certificates conflicts for:
basemap -> python[version='>=2.7,<2.8.0a0'] -> ca-certificates
python=3.8 -> openssl[version='>=1.1.1i,<1.1.2a'] -> ca-certificates
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with fle
xible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retrying with fle
xible solve.
Solving environment: ...working...
Found conflicts! Looking for incompatible packages.
This can take several minutes.  Press CTRL-C to abort.
failed
```

5- Clustering de estaciones basado en su ubicación i.e. Lat & Lon

DBSCAN de sklearn puede ejecutar DBSCAN desde un arreglo vectorial o una matriz de distancia. En nuestro caso, pasamos el arreglo de NumPy Clus_dataSet para encontrar las muestras de núcleo de alta densidad y expandimos los clusters desde ellas.

In [ ]:
```
from sklearn.cluster import DBSCAN
import sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN
db = DBSCAN(eps=0.15, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"]=labels

realClusterNum=len(set(labels)) - (1 if -1 in labels else 0)
clusterNum = len(set(labels))


# A sample of clusters
pdf[["Stn_Name","Tx","Tm","Clus_Db"]].head(5)
```

Puede verse que la para los outliers, la etiqueta en el cluster es -1.

In [ ]:
```
set(labels)
```

6. Visualización de clusters basados en ubicación

Ahora visualizamos los clusters utilizando basemap:

In [ ]:
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
            resolution = 'l', area_thresh = 1000.0,
            llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))



#Visualization1
for clust_number in set(labels):
    c=(([0.4,0.4,0.4]) if clust_number == -1 else colors[np.int(clust_number)])
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color =c,  marker='o', s= 20, alpha = (
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red',)
        print ("Cluster "+str(clust_number)+', Avg Temp: '+ str(np.mean(clust_set.Tm))
```

7. Clustering de estaciones basado en su ubicación, media, máxima y mínima temperatura

Aquí ejecutamos DBSCAN sobre un dataset de 5 dimensiones:

In [ ]:
```
from sklearn.cluster import DBSCAN
import sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym','Tx','Tm','Tn']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"]=labels

realClusterNum=len(set(labels)) - (1 if -1 in labels else 0)
clusterNum = len(set(labels))


# A sample of clusters
pdf[["Stn_Name","Tx","Tm","Clus_Db"]].head(5)
```

8. Visualización de clusters basado en su ubicación y temperatura

In [ ]:
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
            resolution = 'l', area_thresh = 1000.0,
            llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (
            urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))



#Visualization1
for clust_number in set(labels):
    c=(([0.4,0.4,0.4]) if clust_number == -1 else colors[np.int(clust_number)])
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color =c,  marker='o', s= 20, alpha = (
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red',)
        print ("Cluster "+str(clust_number)+', Avg Temp: '+ str(np.mean(clust_set.Tm))
```