estudioExploracion

August 26, 2021

1 Análisis exploratorio

- Alumno: Javier Berlmar Tevar
- Tutor: Jose Norberto Mazón
- Cotutor: Jose Jacobo Zubcoff

1.1 —

1.2 0. Librerías

[1]:
```
#### Librerías generales
import pandas as pd
from pathlib import Path
import plotly.graph_objects as go
import numpy as np

#### Clustering
from sklearn.cluster import KMeans, DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import silhouette_samples
from matplotlib import cm
import matplotlib.pyplot as plt
import plotly.figure_factory as ff
from scipy.cluster.hierarchy import dendrogram, linkage
```

1.3 —

1.4 1. Carga de datos

A continución se realiza la carga de datos de dos ficheros:

1. Un fichero con extensión `xlsx` (CuestionarioTouristInfoAlcoi.xlsx): Contiene las respuestas de las personas que acuden a la oficina de Turismo.
2. Un fichero con extensión `csv` (resultados_cw08.csv)

[2]:
```
path = os.path.dirname(os.path.realpath('__file__'))

p = Path(path)
p = str(p.parent)
```

```
############################ CUESTIONARIO ####################################


p1 = p + '/Datos/CuestionarioTouristInfoAlcoi.xlsx'
p1 = p1.replace("\\", "/")

# se ha instalado el parquete openpyxl para que el motor de la función soporte␣
 ↪el archivo
# pip install openpyxl (en conda prompt)
df_cuest = pd.read_excel(
     p1,
     engine='openpyxl',
)

# renombramos campos para que sea mas sencillo operar con ellos
mapping = {df_cuest.columns[0]:'timestamp', df_cuest.columns[1]: 'residencia',
           df_cuest.columns[2]:'origen', df_cuest.columns[3]: 'num_personas',
           df_cuest.columns[4]:'num_personas_oficina', df_cuest.columns[5]:␣
 ↪'wifi_abierta_tlf'}
df_cuest = df_cuest.rename(columns=mapping)


df_cuest['date'] = df_cuest['timestamp'].dt.date

############################ SENSOR ##########################################


p2 = p + '/Datos/resultados_cw08.csv'
p2 = p2.replace("\\", "/")

df_sensor = pd.read_csv(p2, sep = ';')

df_sensor = df_sensor.rename(columns={'date': 'timestamp'})
df_sensor.timestamp = pd.to_datetime(df_sensor.timestamp)

df_sensor['date'] = df_sensor['timestamp'].dt.date

############################ SSID's Alcoy ####################################

p3 = p + '/Datos/resultados_ssid_alcoi.csv'
p3 = p3.replace("\\", "/")

df_ssid_alcoy = pd.read_csv(p3, sep = ';')
```

1.5 —

1.6 2.Visualización del estado de los conjuntos

A continuación se obtiene una vista previa de lo que nos vamos a encontrar en los datasets.

1.6.1 1. Datos recogidos por la oficina de turismo:

[3]:
```
df_cuest.head()
```

[3]:
```
                 timestamp            residencia     origen  num_personas  \
0 2020-11-27 12:22:02.791                 Alcoi        NaN             1
1 2020-11-27 17:17:22.383                 Alcoi        NaN             1
2 2020-11-27 18:02:33.662  Comunidad Valenciana   Valencia             2
3 2020-11-28 10:16:21.630  Comunidad Valenciana  Valencia              5
4 2020-11-28 11:05:25.625                 Alcoi        NaN             2

   num_personas_oficina wifi_abierta_tlf        date
0                     1               Sí  2020-11-27
1                     1               Sí  2020-11-27
2                     2               No  2020-11-27
3                     3               Sí  2020-11-28
4                     2               No  2020-11-28
```

1.6.2 2. Datos recogidos por el sensor de Hopu.

[4]:
```
df_sensor.head()
```

[4]:
```
                 timestamp sensor                                      ssid  \
0 2020-12-21 12:30:26.336   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
1 2020-12-21 12:30:26.336   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
2 2020-12-21 12:30:27.443   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
3 2020-12-21 12:30:27.443   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
4 2020-12-21 12:30:31.944   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a

                                     mac visitor        date
0  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21
1  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21
2  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21
3  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21
4  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21
```

1.6.3 3. Datos de todos los ssid's existentes recogidos por los sensores.

Cruzado de datos con el fichero que contiene las ssid's que son de Alcoi (capturadas por el dispositivo). A continuación se muestra el fichero que contiene las ssid's de las redes Wifi localizadas en Alcoi

[5]:
```
# nos quedamos con la columna relevante
df_ssid_alcoy = df_ssid_alcoy[["ssid"]]
df_ssid_alcoy = df_ssid_alcoy.dropna(subset=["ssid"])
df_ssid_alcoy.head()
```

[5]:

	ssid
2	00ccba5c6f7925e1fc70b2e0893852bf596c88c2
3	0143a4fa9f588990add630753c676a2c5961232d
4	015075bd5ebf650fa28d9016af6a4a50e25cb153
5	01593e9505926488c67e8cc24b8fc3de53f7dace
6	0185884ec4753ec27be08b3ee0bc7086b4db5ee4

Se obtienen los visitantes (visitor_v2) según ssid's recogidos por los sensores de Alcoi.

[6]:
```
list_ssid = df_ssid_alcoy['ssid'].tolist()

def f(row):
    if row['ssid'] in list_ssid:
        val = 'Y'
    else:
        val = 'N'
    return val

#df_sensor['visitor_v2'] = np.where(df_sensor['ssid'] in list_ssid, "Y", "N")
df_sensor['visitor_v2'] = df_sensor.apply(f, axis=1)

df_sensor.head()
```

[6]:

	timestamp	sensor	ssid	mac	visitor	date	visitor_v2
0	2020-12-21 12:30:26.336	CW08	cada330f25b98b5e0d44da5fc3decb161d337b4a	b627731c6c589cdabd7b02e979f7019c1d1efcf4	Y	2020-12-21	N
1	2020-12-21 12:30:26.336	CW08	cada330f25b98b5e0d44da5fc3decb161d337b4a	b627731c6c589cdabd7b02e979f7019c1d1efcf4	Y	2020-12-21	N
2	2020-12-21 12:30:27.443	CW08	cada330f25b98b5e0d44da5fc3decb161d337b4a	b627731c6c589cdabd7b02e979f7019c1d1efcf4	Y	2020-12-21	N
3	2020-12-21 12:30:27.443	CW08	cada330f25b98b5e0d44da5fc3decb161d337b4a	b627731c6c589cdabd7b02e979f7019c1d1efcf4	Y	2020-12-21	N
4	2020-12-21 12:30:31.944	CW08	cada330f25b98b5e0d44da5fc3decb161d337b4a	b627731c6c589cdabd7b02e979f7019c1d1efcf4	Y	2020-12-21	N

1.7 —

1.8 3. Análisis exploratorio

1.8.1 Nulos: Primero comprobamos los nulos para cada conjunto:

1. Cuestionario:

[7]:
```
df_cuest.isna().sum()
```

[7]:
```
timestamp                0
residencia               0
origen                 520
num_personas             0
num_personas_oficina     0
wifi_abierta_tlf         0
date                     0
dtype: int64
```

Se observan nulos en el campo de origen del cuestionario, en este caso no hay problema, pues que sea nulo depende directamente de la pregunta anterior del cuestionario. Concretamente, si en la anterior pregunta (segunda pregunta): "¿Cúal es su lugar de residencia?" se contesta Alcoi, la respuesta a la siguiente pregunta (tercera) figurará vacía, ya que se deberá especificar el origen en ese caso. Por tanto, lo que va a realizar es un relleno de los campos nulos con "Alcoi".

[8]:
```
df_cuest = df_cuest.fillna("Alcoi")
df_cuest.isna().sum()
```

[8]:
```
timestamp              0
residencia             0
origen                 0
num_personas           0
num_personas_oficina   0
wifi_abierta_tlf       0
date                   0
dtype: int64
```

2. Sensor:

[9]:
```
df_sensor.isna().sum()
```

[9]:
```
timestamp         0
sensor            0
ssid              2
mac               1
visitor       68444
date              0
visitor_v2        0
dtype: int64
```

Se observa que gran parte de los nulos cae sobre el campo visitor, de lo cual no hay que preocuparse porque es nuestra tarea principal la estimación del mismo. Se procederá a eliminar los registros en los que en los campos de ssid y mac haya algún nulo.

[10]:
```
# df_sensor = df_sensor.drop(['visitor'], axis=1) # primero borramos la columna␣
↪irrelevante
df_sensor = df_sensor.dropna(subset=["ssid", "mac"])
```

```
df_sensor.isna().sum()
```

[10]:
```
timestamp         0
sensor            0
ssid              0
mac               0
visitor       68442
date              0
visitor_v2        0
dtype: int64
```

1.8.2 Filtrado: Se procede a realizar un filtrado para que coincidan en rango horario tanto el conjunto del sensor como el de la oficina.

Horario: Lunes a Viernes: 10-14h. Sábados, domingos y festivos: 10-14h. 4/01 y 5/01:10-14 y 17-19h. Festivos cerrados: 1,6/01, 1/05 y 25/12.

Alcance de sensor: Nombre del parámetro: WiFi RSSI Detection Threshold: - -120 (máximo alcance). - Se cambia a valor -60 el 10 de mayo (mínimo alcance). - Se cambia a valor -80 (alcance intermedio) el 22 junio

[11]:
```
pd.to_datetime(df_sensor['timestamp']).dt.weekday # .value_counts().
↪sort_values()
```

[11]:
```
0        0
1        0
2        0
3        0
4        0
        ..
83071    3
83072    3
83073    3
83074    3
83075    3
Name: timestamp, Length: 83074, dtype: int64
```

[12]:
```
df_sensor[(pd.to_datetime(df_sensor['timestamp']).dt.day == 4)]
```

[12]:
```
                    timestamp sensor  \
6828  2021-01-04 00:12:19.384   CW08
6829  2021-01-04 00:37:36.364   CW08
6830  2021-01-04 00:54:27.056   CW08
6831  2021-01-04 01:02:53.370   CW08
6832  2021-01-04 01:19:44.891   CW08
…                          …    …
80529 2021-07-04 09:52:16.510   CW08
80530 2021-07-04 10:37:08.315   CW08
```

```
80531 2021-07-04 14:05:22.381   CW08
80532 2021-07-04 14:07:57.084   CW08
80533 2021-07-04 14:12:41.829   CW08

                                          ssid  \
6828   2086eb1dccef68d850e0063c5340302b616177b2
6829   2086eb1dccef68d850e0063c5340302b616177b2
6830   2086eb1dccef68d850e0063c5340302b616177b2
6831   2086eb1dccef68d850e0063c5340302b616177b2
6832   2086eb1dccef68d850e0063c5340302b616177b2
…                                           …
80529  cca78a1e963f3ba3443bf3e54a9aa939f73d223d
80530  a9a49a9f9fa949e1de2879df0c09d83ebd0aa58e
80531  6bd998e6f05877f7222519de751d2b7c7d201ad3
80532  6bd998e6f05877f7222519de751d2b7c7d201ad3
80533  6bd998e6f05877f7222519de751d2b7c7d201ad3

                                           mac visitor        date visitor_v2
6828   d1b09fcbf788d755c76d8ec4803fd73aebf95246       Y  2021-01-04          Y
6829   d1b09fcbf788d755c76d8ec4803fd73aebf95246       Y  2021-01-04          Y
6830   d1b09fcbf788d755c76d8ec4803fd73aebf95246       Y  2021-01-04          Y
6831   d1b09fcbf788d755c76d8ec4803fd73aebf95246       Y  2021-01-04          Y
6832   d1b09fcbf788d755c76d8ec4803fd73aebf95246       Y  2021-01-04          Y
…                                           …     …          …        …
80529  5cbcffeab5fa9206a8c2712ef085daab688dd086     NaN  2021-07-04          Y
80530  253f75802600eb70118d14acee8e83e281f3e271     NaN  2021-07-04          N
80531  215776a4509318d0d2dae66a01ab0bf459817da8     NaN  2021-07-04          Y
80532  215776a4509318d0d2dae66a01ab0bf459817da8     NaN  2021-07-04          Y
80533  215776a4509318d0d2dae66a01ab0bf459817da8     NaN  2021-07-04          Y

[3186 rows x 7 columns]
```

[13]:
```
# Función que se encarga del filtrado (rango horario y festivos)
# realizamos filtrado para hacer coincidir el horario y días del sensor
# con los de la apertura de la oficina

def filtroHorario (df):

    """
    Filtramos para obtener información de los registros de 10 a 14h, y de 17 a
↪19h en los días correspondientes (4/01 y 5/01)
    """
    df = df[(((pd.to_datetime(df['timestamp']).dt.hour > 9) & (pd.
↪to_datetime(df['timestamp']).dt.hour < 14)) |
            ((pd.to_datetime(df['timestamp']).dt.hour > 16) & (pd.
↪to_datetime(df['timestamp']).dt.hour < 19) &
```

```
            (((pd.to_datetime(df['timestamp']).dt.month == 1) & (pd.
↪to_datetime(df['timestamp']).dt.day == 4)) |
            ((pd.to_datetime(df['timestamp']).dt.month == 1) & (pd.
↪to_datetime(df['timestamp']).dt.day == 5))))
        )]

    """
    Días en los que la oficina de turismo está cerrada
    1,6/01, 1/05 y 25/12
    """
    # Lista con los días a eliminar
    lista = ['01-01', '06-01', '01-05', '25-12']

    # uso de ~ para la negación de la condición, desarrollo de la máscara
    mask = ~df['timestamp'].dt.strftime("%d-%m").isin(lista)
    df = df[mask]
    return df
```

[14]:
```
# Aplicamos el filtro
test = df_sensor.copy()
df_sensor = filtroHorario(test)
df_sensor.head()
```

[14]:
```
                 timestamp sensor                                      ssid  \
0 2020-12-21 12:30:26.336   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
1 2020-12-21 12:30:26.336   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
2 2020-12-21 12:30:27.443   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
3 2020-12-21 12:30:27.443   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a
4 2020-12-21 12:30:31.944   CW08  cada330f25b98b5e0d44da5fc3decb161d337b4a

                                       mac visitor        date visitor_v2
0  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21          N
1  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21          N
2  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21          N
3  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21          N
4  b627731c6c589cdabd7b02e979f7019c1d1efcf4       Y  2020-12-21          N
```

1.8.3 Descripción de los datos: percentiles y desviación estándar:

1. Cuestionario:

[15]:
```
data = df_cuest
perc =[.20, .40, .60, .80]
desc = data.describe(percentiles=perc, include='all', datetime_is_numeric=True)
desc
```

[15]:

	timestamp	residencia	origen	\
count	1410	1410	1410	
unique	NaN	4	239	
top	NaN	Comunidad Valenciana	Alcoi	
freq	NaN	729	520	
mean	2021-03-15 02:23:07.244278784	NaN	NaN	
min	2020-11-27 12:22:02.791000	NaN	NaN	
20%	2020-12-19 14:04:55.633400064	NaN	NaN	
40%	2021-03-13 13:54:27.152800256	NaN	NaN	
50%	2021-04-02 13:59:58.656999936	NaN	NaN	
60%	2021-04-10 12:53:59.550000128	NaN	NaN	
80%	2021-05-22 11:51:51.611400192	NaN	NaN	
max	2021-07-07 13:59:22.824000	NaN	NaN	
std	NaN	NaN	NaN	

	num_personas	num_personas_oficina	wifi_abierta_tlf	date
count	1410.000000	1410.000000	1410	1410
unique	NaN	NaN	3	183
top	NaN	NaN	Sí	2021-04-04
freq	NaN	NaN	848	41
mean	1.118440	0.858865	NaN	NaN
min	0.000000	0.000000	NaN	NaN
20%	0.000000	0.000000	NaN	NaN
40%	1.000000	1.000000	NaN	NaN
50%	1.000000	1.000000	NaN	NaN
60%	1.000000	1.000000	NaN	NaN
80%	2.000000	1.000000	NaN	NaN
max	10.000000	5.000000	NaN	NaN
std	1.143755	0.865273	NaN	NaN

2. Sensor

[16]:
```
data = df_sensor
desc = data.describe(datetime_is_numeric=True)
desc
```

[16]:

	timestamp
count	45816
mean	2021-03-24 16:31:42.815804928
min	2020-12-21 12:30:26.336000
25%	2021-03-01 12:16:54.264499968
50%	2021-03-27 12:21:37.189000192
75%	2021-04-21 10:48:59.488000
max	2021-07-08 10:08:15.358000

1.8.4 Campos cuestionario: A continuación, se realiza un breve análisis exploratorio sobre los campos del conjunto de datos del cuestionario.

date: Fecha (día) de realización del cuestionario.

[17]:
```
df_cuest['date'].value_counts().sort_values()
```

[17]:
```
2021-01-24     1
2021-02-05     1
2021-06-17     1
2021-01-07     1
2021-02-15     1
              ..
2021-04-02    26
2020-12-06    29
2021-01-04    29
2021-04-03    33
2021-04-04    41
Name: date, Length: 183, dtype: int64
```

---

residencia: Lugar de residencia de los visitantes a la oficina de turismo de Alcoi.

Obtenemos el número de residentes por cada categoría de localización.

[18]:
```
df_cuest['residencia'].value_counts().sort_values()
```

[18]:
```
Internacional            68
España                  102
Alcoi                   511
Comunidad Valenciana    729
Name: residencia, dtype: int64
```

---

origen: Lugar originario del/de los visitantes.

[19]:
```
df_cuest['origen'].value_counts().sort_values()
```

[19]:
```
Benaguacil      1
Adzeneta        1
Vinaroz         1
Colombia        1
Guardamar       1
               ..
Alicante       28
Alicante       52
Valencia      117
Valencia      190
Alcoi         520
```

```
Name: origen, Length: 239, dtype: int64
```

---

num_personas: Número de personas del grupo con el que ha viajado.

[20]:
```
df_cuest['num_personas'].value_counts().sort_values()
```

[20]:
```
8       1
7       1
10      2
9       2
6       2
5       9
4      34
3     106
2     162
0     402
1     689
Name: num_personas, dtype: int64
```

---

num_personas_oficina: Número de personas con el que ha viajado y ha entrado a la oficina.

[21]:
```
df_cuest['num_personas_oficina'].value_counts().sort_values()
```

[21]:
```
5      2
4     17
3     58
2    146
0    520
1    667
Name: num_personas_oficina, dtype: int64
```

---

wifi_abierta_tlf: Marca (Sí/No) si la persona tiene la red wifi del teléfono activa

[22]:
```
df_cuest['wifi_abierta_tlf'].value_counts().sort_values()
```

[22]:
```
No entiendo la pregunta    123
No                         439
Sí                         848
Name: wifi_abierta_tlf, dtype: int64
```

1.8.5 Campos sensor: A continuación, se realiza un breve análisis exploratorio sobre los campos del conjunto de datos del sensor.

date: Fecha de registro del dispositivo.

[23]:
```
df_sensor['date'].value_counts().sort_values()
```

[23]:
```
2021-07-04       1
2021-06-20       1
2021-06-19       2
2021-05-23       2
2021-07-08       4
                ...
2021-04-07    1033
2021-03-18    1047
2020-12-29    1237
2021-02-24    1383
2021-04-15    1511
Name: date, Length: 153, dtype: int64
```

---

sensor: Id del sensor que ha refgistrado el dispositivo

[24]:
```
df_sensor['sensor'].value_counts().sort_values()
```

[24]:
```
CW08    45816
Name: sensor, dtype: int64
```

Se está haciendo el análisis para un solo sensor que está ubicado en la oficina de turismo

---

1.9 ssid:

ssid de la red Wifi preferida por el dispositivo registrado.

[25]:
```
df_sensor['ssid'].value_counts().sort_values()
```

[25]:
```
2e366797b4cfa6eaea33ded77e0de3d0545ab207        1
eb374b99cc6071372b3310256148c8e761efd855        1
a59ed7c74633766620709f88654eb51027ce9a04        1
b59f1729484a2a12ee03ac982e281ae1d3100b23        1
8f0bda519a34296413f92abed11db67166c76776        1
                                              ...
87b1965f588f3dc856ecaac238aaeaac1a71f247      514
2e4d03b0c09a73f874b341a33152129c5bba5011     1241
d75f15af8afffe8c66e9863c1a4dfcad8e98a056     2226
ffd2edd79fe4bac98610a2ad6892b53395e44e34     8493
cca78a1e963f3ba3443bf3e54a9aa939f73d223d    24775
Name: ssid, Length: 1975, dtype: int64
```

Hay ssid's registradas que destacan (en número) sobre las demás. Utilizar para algoritmo de clasificación (e.g. rpart). Gráfica –> regla del codo. Estas ssid's nos servirán para compararlas con las ssid's que recogidas que solo pertenecen a Alcoi

[26]:
```
df_aux = pd.DataFrame(df_sensor['ssid'].value_counts().sort_values())
df_aux.reset_index(inplace=True)
df_aux.rename(columns={"index": "ssid", "ssid": "counts"}, inplace=True)
df_aux['ssid_id'] = range(0, len(df_aux))

df_aux.head()
```

[26]:

	ssid	counts	ssid_id
0	2e366797b4cfa6eaea33ded77e0de3d0545ab207	1	0
1	eb374b99cc6071372b3310256148c8e761efd855	1	1
2	a59ed7c74633766620709f88654eb51027ce9a04	1	2
3	b59f1729484a2a12ee03ac982e281ae1d3100b23	1	3
4	8f0bda519a34296413f92abed11db67166c76776	1	4

[27]:
```
fig = go.Figure(
    data=[go.Scatter(x=df_aux['ssid_id'], y=df_aux['counts'])],

)



fig.update_layout(
    title = dict(text = "SSID Frecuencias"),
    xaxis=dict(rangeslider=dict(visible=True)
    )
)
```

Esta gráfica muestra las frecuencias de los ssid's registrados por el sensor, en el eje de ordendas están situados de los id's para cada ssid. Al final de la gráfica se puede observar una subida en la frecuencia de ssid's (estas han sido ordenadas por frecuencias previamente), esto nos puede ayudar para discernir de visitantes y residentes.

1.9.1 Clustering

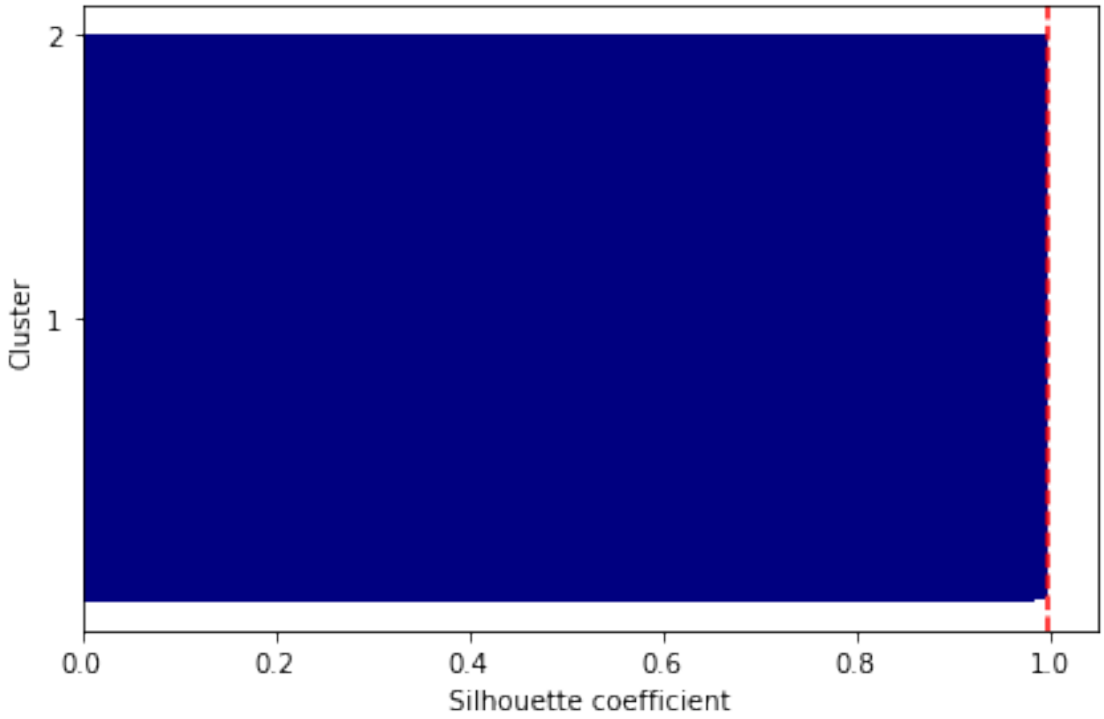
Regla del codo Para determinar el parámetro K (número de clusters) , uno de los métodos a utilizar es la regla del codo en el SSE (Error Sum of Squares).

[28]:
```
distortions = []
for i in range(1, 14):
    km_model = KMeans(n_clusters=i)
    km_model.fit(df_aux[['counts']])
    distortions.append(km_model.inertia_)



x = np.arange(14)
fig = go.Figure(data=go.Scatter(x=x, y=distortions))
fig.update_layout(
    title="Regla del codo SSID clustering óptimo",
    xaxis_title="Número de clusters",
```

```
    yaxis_title="SSE",
    legend_title="SSE",
)
```

Método de la silueta El otro método que se empleará es selección de cluster por puntuación de la silueta, este sirve para evaluar la calidad de los clusters.

[29]:
```
km = KMeans(n_clusters=2)
y_km = km.fit_predict(df_aux[['counts']])

cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(df_aux[['counts']], y_km,␣
 ↪metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```



[30]:
```
km = KMeans(n_clusters=2)

y_km = km.fit_predict(df_aux[['counts']])
df_y_km = pd.DataFrame(y_km, columns = ['counts_clust']).reset_index(drop=True)

# clusters
df_y_km['counts_clust'].value_counts().sort_values()
#df_y_km.tail(10)
```

[30]:
```
1       1
0    1974
Name: counts_clust, dtype: int64
```

2 Dendograma Frecuencia SSID

[31]:
```
df_aux['counts']
```

[31]:
```
0           1
1           1
2           1
3           1
4           1
        ...
```

```
1970      514
1971     1241
1972     2226
1973     8493
1974    24775
Name: counts, Length: 1975, dtype: int64
```

[32]:
```
"""
OTRA FORMA

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(data, method='ward'))
"""


linked = linkage(df_aux[['counts']], 'single')

labelList = range(0, len(df_aux.ssid_id))



"""
VERSION matplot

plt.figure(figsize=(10, 7))

dendrogram(linked,
            orientation='top',
            labels=labelList,
            distance_sort='descending',
            show_leaf_counts=True)
plt.show()

"""



"""
VERSION PLOTLY
"""

np.random.seed(1)

#X = linked # 15 samples, with 12 dimensions each
```

```
fig = ff.create_dendrogram(linked)
fig.update_layout(width=800, height=500)
fig.show()
```

2.1 mac:

Identificador (Media Access Control) del dispositivo identificado por el sensor, puede ser variable.

[33]:
```
df_sensor['mac'].value_counts().sort_values()
```

[33]:
```
9a6209278676991e69ee7ea08f35944c2074084b       1
c3581ab2ba5156c044eea2521af5db90b6bff10d       1
9fa70d1c5c56d9fca397022670687f6f02e18488       1
cd1048aa5a1873542a0a76137e7aeb065c03086f       1
7b64077479b5913eeed9eef94f74156b3ffb0ee1       1
                                             ...
a39a6ee87eece55a4a61453bec9448d7eb5f0f76    1983
7ecc4c8bbd75b66d0d0c04096e161e9b2f65a428    2907
6ec65bc349733bc0e85d17294d3a9cb6d8b8bdc8    4069
d29675014f618f3ef3f6b75f146b4be71c513c16    4256
292128facc8342026c849a8ae3b1fbbcb0573291    6031
Name: mac, Length: 13038, dtype: int64
```

Vamos a adaptar estos datos para obtener una gráfica que nos muestre de una manera más intuitiva lo que está sucediendo con las mac's. Primero preparamos los datos. El objetivo de la gráfica es comprobar si existe un cambio pronunciado en la frecuencia con la que aparecen las mac. De esta forma se podría intuir qué dispositivos aparecen en el sensor de la oficina y cuales no. Si aparecen con mucha frecuencia, se podría decir que son mac's de dispositivos pertenecientes a empleados de la oficina.

[34]:
```
df_aux2 = pd.DataFrame(df_sensor['mac'].value_counts().sort_values())
df_aux2.reset_index(inplace=True)

df_aux2.rename(columns={"index": "mac", "mac": "counts"}, inplace=True)

df_aux2['mac_id'] = range(0, len(df_aux2))
```

[35]:
```
df_aux.head()
```

[35]:

	ssid	counts	ssid_id
0	2e366797b4cfa6eaea33ded77e0de3d0545ab207	1	0
1	eb374b99cc6071372b3310256148c8e761efd855	1	1
2	a59ed7c74633766620709f88654eb51027ce9a04	1	2
3	b59f1729484a2a12ee03ac982e281ae1d3100b23	1	3
4	8f0bda519a34296413f92abed11db67166c76776	1	4

[36]:
```
fig = go.Figure(
    data=[go.Scatter(x=df_aux2['mac_id'], y=df_aux2['counts'])],

)



fig.update_layout(
    title = dict(text = "MAC Frecuencias"),
    xaxis=dict(rangeslider=dict(visible=True)
    )
)
```

[37]:
```
df_aux2.head(3)
#df_aux['counts2'] = df_aux['counts'] # es una artimaña para que funcione el␣
 ↪clustering, necesita al menos dos columnas
```

[37]:

	mac	counts	mac_id
0	9a6209278676991e69ee7ea08f35944c2074084b	1	0
1	c3581ab2ba5156c044eea2521af5db90b6bff10d	1	1
2	9fa70d1c5c56d9fca397022670687f6f02e18488	1	2

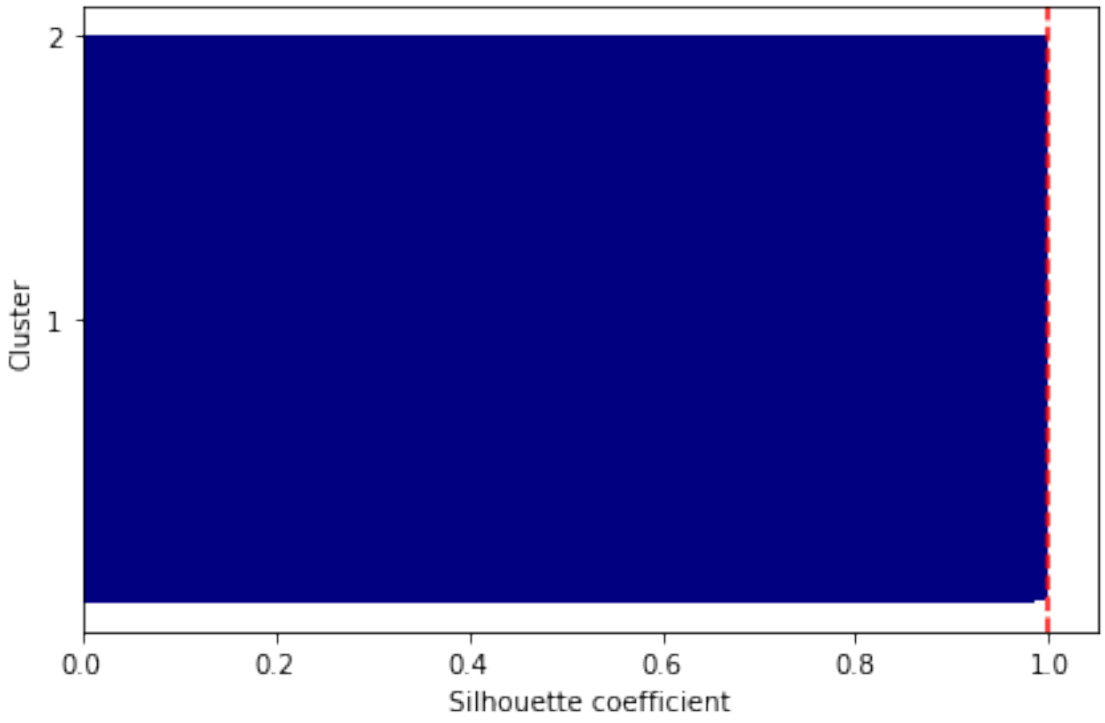
2.1.1 Clustering

[38]:
```
distortions = []
for i in range(1, 14):
    km_model2 = KMeans(n_clusters=i)
    km_model2.fit(df_aux2[['counts']])
    distortions.append(km_model2.inertia_)

x = np.arange(14)
fig = go.Figure(data=go.Scatter(x=x, y=distortions))
fig.update_layout(
    title="Regla del codo MAC clustering óptimo",
    xaxis_title="Número de clusters",
    yaxis_title="SSE",
    legend_title="SSE",
)
```

Método de la silueta El otro método que se empleará es selección de cluster por puntuación de la silueta, este sirve para evaluar la calidad de los clusters.

[39]:
```
km = KMeans(n_clusters=2)
y_km = km.fit_predict(df_aux2[['counts']])

cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
```

```
silhouette_vals = silhouette_samples(df_aux2[['counts']], y_km,␣
 ↪metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```



[40]:
```
"""
df_aux2_tree_set = df_aux2.copy()
sensor_tree = DecisionTreeClassifier(random_state=0)
sensor_tree.fit(x_train_c1, y_train_c1)
"""

km2 = KMeans(n_clusters=2)

y_km2 = km2.fit_predict(df_aux2[['counts']])
df_y_km2 = pd.DataFrame(y_km2, columns = ['counts_clust']).
 ↪reset_index(drop=True)

# clusters
df_y_km2['counts_clust'].value_counts().sort_values()
#df_y_km2.tail(10)
```

[40]:
```
1        4
0    13034
Name: counts_clust, dtype: int64
```

3 Dendograma Frecuencia SSID

[41]:
```
df_aux2['counts']
```

[41]:
```
0           1
1           1
2           1
3           1
4           1
          …
13033    1983
13034    2907
13035    4069
13036    4256
13037    6031
Name: counts, Length: 13038, dtype: int64
```

[42]:
```
"""
OTRA FORMA

import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Customer Dendograms")
```

```
dend = shc.dendrogram(shc.linkage(data, method='ward'))
"""


linked2 = linkage(df_aux2[['counts']], 'single')

labelList2 = range(0, len(df_aux2.mac_id))



"""
VERSION matplot

plt.figure(figsize=(10, 7))

dendrogram(linked,
            orientation='top',
            labels=labelList,
            distance_sort='descending',
            show_leaf_counts=True)
plt.show()

"""



"""
VERSION PLOTLY
"""

np.random.seed(1)

#X = linked # 15 samples, with 12 dimensions each
fig = ff.create_dendrogram(linked2)
fig.update_layout(width=800, height=500)
fig.show()
```

Se observa como por medio del clustering, un algoritmo de aprendizaje no supervisado, separa en dos clusters las MAC's que se han repetido 427 veces o más (en este caso). Esto puede servir como apoyo a las heurístcas para discernir si el dispisitvo (identificado por su MAC) es pertenciente una persona residente en Alcoy o no.

---

visitor: Valor booleano que nos dice si es vistante o no (Y/N)

[43]:
```
df_sensor['visitor'].value_counts().sort_values()
```

[43]:
```
N     368
Y    7188
Name: visitor, dtype: int64
```

3.1 Heurísticas de detección

1. MAC repetida (alg. clasificación: rpart + gráfica + clustering): Siginificará que ese dispositivo, si se repite mucho, muy probablente sea residente en Alcoi. Se deberá de llevar cuidado con este campo pues hay algunos dispositivos que cambian de MAC.
2. ssid repetida (alg. clasificación: rpart + gráfica + clustering): Si esa id de Wifi favorita figura en las reconocidas por el dispositivo dentro de Alcoi será residente de Alcoi.

En esta visualización se obtienen las proporciones de accidentes por distrito. El distrito en el que ocurren más accidentes es el de Salamanca seguido de Chamartín. Seguramente es porque se sitúan en el centro y sean de los lugares de Madrid más concurridos.

Preguntas y Anotaciones:

- ¿Se tiene el ssid de la oficina de turismo? es para ver si coincide con el ssid que más se repite en la gráfica del ssid del sensor de la oficina de turimo.
- Debo juntar en el Dataframe del sensor (al principio) si la ssid pertenece a una localizada en el conjunto de todas la ssids recogidas de Alcoy.