PRÁCTICAS (APARTADO II)

Análisis, Diseño y Procesamiento de Datos Aplicados a las Ciencias y a las Tecnologías(ADP)

Máster Universitario en Inteligencia Computacional e Internet de las Cosas

Universidad de Córdoba, EPSC

2022/2023

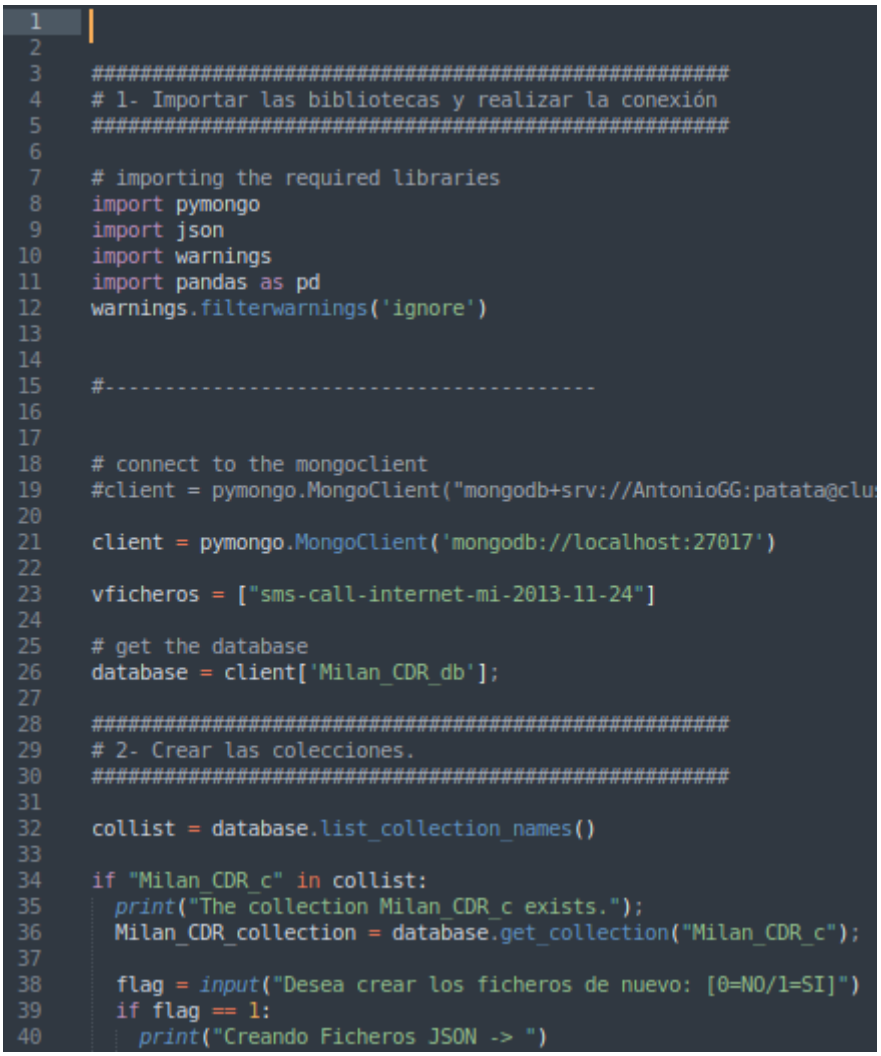
Autor:

Antonio Gómez Giménez (i72gogia@uco.es)

Índice:

1. Ejercicio:

Para el primer ejercicio hay que descargar los ficheros “.txt” correspondientes, en mi casos desde el lunes 18 hasta el domingo 24, y crear un programa en Python para cargar los datos en la colección “Milan_CDR_c” de la base de datos “Milan_CDR_db”. Para lograrlo se ha creado el siguiente código:



```
1
2
3   ##################################################
4   # 1- Importar las bibliotecas y realizar la conexión
5   ##################################################
6
7   # importing the required libraries
8   import pymongo
9   import json
10  import warnings
11  import pandas as pd
12  warnings.filterwarnings('ignore')
13
14
15  #---------------------------------------
16
17
18  # connect to the mongoclient
19  #client = pymongo.MongoClient("mongodb+srv://AntonioGG:patata@clu
20
21  client = pymongo.MongoClient('mongodb://localhost:27017')
22
23  vficheros = ["sms-call-internet-mi-2013-11-24"]
24
25  # get the database
26  database = client['Milan_CDR_db'];
27
28  ##################################################
29  # 2- Crear las colecciones.
30  ##################################################
31
32  collist = database.list_collection_names()
33
34  if "Milan_CDR_c" in collist:
35    print("The collection Milan_CDR_c exists.");
36    Milan_CDR_collection = database.get_collection("Milan_CDR_c");
37
38    flag = input("Desea crear los ficheros de nuevo: [0=NO/1=SI]")
39    if flag == 1:
40      print("Creando Ficheros JSON -> ")
```

```
        print("Creando Ficheros JSON -> ")
        for x in vficheros:
          datos = pd.read_csv(x+".txt",
                              sep='\t',
                              header=0,
                              names=['cellid', 'time', 'countrycode', 'smsin','smsout','callin','callout','internet'])
          datos.to_json(x+".json", orient = "records")

      print("Introduciendo Ficheros JSON a la BD -> ")

      for x in vficheros:
        print("Introduciendo Ficheros  -> ")
        print(x)
        with open(x+".json") as f:
          file_data = json.load(f);
          # insert the data into the collection
          Milan_CDR_collection.insert_many(file_data);


    else:
      # create weekly demand collection
      database.create_collection("Milan_CDR_c")
      Milan_CDR_collection = database.get_collection("Milan_CDR_c");

      flag = input("Desea crear los ficheros de nuevo: [0=NO/1=SI]")
      if flag == 1:
        print("Creando Ficheros JSON -> ")
        for x in vficheros:
          datos = pd.read_csv(x+".txt",
                              sep='\t',
                              header=0,
                              names=['cellid', 'time', 'countrycode', 'smsin','smsout','callin','callout','internet'])
          datos.to_json(x+".json", orient = "records")

      print("Introduciendo Ficheros JSON a la BD -> ")

      for x in vficheros:
        print("Introduciendo Ficheros  -> ")
        print(x)
        with open(x+".json") as f:
          file_data = json.load(f);
          # insert the data into the collection
          Milan_CDR_collection.insert_many(file_data);
```

Con dicho código se crea la bd y se comprueba si ya existía la conexión. En el caso de no existir, se recorren todos los ficheros leyendo su contenido y transformándolos a JSON, para que posteriormente se realice la subida de los datos a la colección Milan_CDR_c de la bd.

Como este método daba problemas se dividieron los archivos, insertándose uno a uno. Primero se creaba la bd con la carga del primer día (su respectivo json). Posteriormente se introducía el json de cada día, teniendo en cuenta la existencia de la colección.

No se podía realizar la carga de todos los datos a la vez ya que el almacenamiento del programa era superior a la ram de mi ordenador y se ejecutaba el proceso. Finalmente se consiguió introducir todos los datos a la base de datos.

```
Milan_CDR_db> db.Milan_CDR_c.count()
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or esti
matedDocumentCount.
37217322
```

2. Ejercicio:

En el segundo apartado se pedía encontrar los países con los que se interactúa. Para ello es necesario fijarse en el country code ya que nos ofrece el código de los países con los que interactúa. Cabe destacar que el país 0 no es ningún país.

Para obtener estos datos, se ha creado el siguiente código para acceder a la bd creada con anterioridad:

```
print("Modo peticiones")
my_set=set({0})
#busqueda de paises con los que interacciona
for i in Milan_CDR_collection.find({},{"_id": 0, "countrycode": 1}):
  my_set.add(i['countrycode'])

print(my_set)
```

Los resultados obtenidos son los siguientes:

```
{0, 1, 7, 20, 27, 30, 31, 32, 33, 34, 36, 39, 40, 41, 43, 44, 45, 46, 47, 48, 49, 51, 52, 53, 54,
55, 56, 57, 58, 60, 61, 62, 63, 64, 65, 66, 14413, 81, 82, 84, 86, 90, 91, 92, 93, 94, 95, 98, 882
16, 88233, 88239, 211, 212, 213, 216, 218, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230,
231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 248, 249, 250, 251
, 252, 18683, 254, 255, 256, 257, 258, 18684, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 29
1, 297, 298, 18762, 18763, 18765, 18768, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 370, 37
1, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 385, 386, 387, 389, 12684, 12687, 420, 4
21, 423, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 12843, 12845, 590, 591, 592, 593, 594,
595, 596, 597, 598, 599, 8816, 8817, 8818, 670, 672, 673, 674, 675, 676, 677, 678, 679, 682, 683,
685, 687, 688, 689, 690, 692, 850, 852, 853, 855, 856, 870, 880, 881, 882, 886, 960, 961, 962, 963
, 964, 965, 966, 967, 968, 970, 971, 972, 973, 974, 975, 976, 977, 992, 993, 994, 995, 996, 998, 9
7259, 29773, 29774, 1129, 1204, 1214, 1226, 1235, 1242, 1246, 1250, 253, 1289, 17676, 1306, 1340,
1345, 1365, 1403, 1416, 1418, 1438, 1441, 1450, 1473, 1514, 1519, 7700, 7701, 7702, 7705, 7707, 77
11, 7712, 7713, 7714, 7715, 7717, 7721, 1579, 1581, 7725, 7726, 7727, 7729, 1587, 1604, 1613, 7771
, 7775, 7776, 7777, 7778, 1647, 1664, 1670, 1671, 1705, 18092, 1709, 18094, 18093, 18096, 18097, 1
8098, 18099, 1721, 1758, 1767, 1778, 50931, 1780, 1784, 50936, 50938, 1787, 50947, 50948, 1808, 18
09, 1819, 1829, 1849, 18686, 18687, 1867, 1902, 1905, 1907, 1924, 1929, 1930, 1938, 1939}
```

Como podemos observar nos devuelve el countrycode de muchos países, aun así parece que existen valores outliers ya que countrycode como 18686 no parece un código válido, aun así se comprobó que existía en la bd. Se puede apreciar en la siguiente imagen:

```
test> use Milan_CDR_db
switched to db Milan_CDR_db
Milan_CDR_db> db.Milan_CDR_c.find({"countrycode": 18686},{"_id": 0, "countrycode
": 1})
[
  { countrycode: 18686 },
  { countrycode: 18686 },
  { countrycode: 18686 },
  { countrycode: 18686 }
```

Por poner algún ejemplo válido de país nos encontramos los siguientes:
7 -> Russia
20 -> Egipto
27 -> Sudáfrica
30 -> Grecia
31 -> Países Bajos
32 -> Bélgica
33 -> Francia
34 -> España
etc…

3. Ejercicio:

Para el tercer apartado se pide encontrar qué país es con el que más se interactúa además de Italia. Para ello se ha creado el siguiente código:

```
result = Milan_CDR_collection.aggregate([
## stage 1
{
    "$group" :
            {"_id": "$countrycode",
            "total": { "$sum": 1 }}
},
{"$sort": {"total":-1}},#ordeno de mayor a menor
{"$limit": 3}#cojo solo dos
])

for i in result:
  print(i)
```

Los resultados obtenidos son los siguientes:

```
Modo peticiones
{'_id': 39, 'total': 10079537}
{'_id': 0, 'total': 7908739}
{'_id': 33, 'total': 1951585}
```

Si obviamos al primero que es Italia y el segundo que el el countrycode cuando no se sabe el país, podemos afirmar que el país con el que más interactúa Milan es Francia con el código de 33.

4. Ejercicio:

En este ejercicio se nos pregunta qué celda comunica más con el extranjero, siendo X. Para llegar a esta conclusión se realizó el siguiente código:

```
result = Milan_CDR_collection.aggregate([
{
    "$match" :
            {"countrycode": {"$not": {"$eq": 0}}}
},
{
    "$match" :
            {"countrycode": {"$not": {"$eq":33}}}
},
{
    "$group" :
            {"_id": "$cellid",
            "total": { "$sum": 1 }}
},
{"$limit": 1}#cojo solo una
])

for i in result:
  print(i)
```

Se realizaron dos match para eliminar todos los datos relacionados con el countrycode 0 y 33 que son los datos relacionados con el propio país y los que no se saben de qué país son. Sobre los datos restantes se cuenta para cada celda cuantos datos hay y así obtener la celda con más comunicación con el extranjero. El resultado es el siguiente:

```
Modo peticiones
{'_id': 1416, 'total': 1899}
```

Por lo tanto, la celda 1416 es la celda con mayor cantidad de comunicaciones con el exterior, con un total de 1899.

5. Ejercicio:

En este apartado se busca encontrar la celda con más actividad de smsin, smsout callin, callout, internet y la total, para ello se realizó el siguiente código:

```
result = Milan_CDR_collection.aggregate([
{
    "$group" :
            {"_id": "$cellid",
            "maxsmsin": { "$max": "$smsin" }}
},
{"$sort": {"maxsmsin":-1}},#ordeno de mayor a menor
{"$limit": 1}#cojo solo dos
])
for i in result:
  print(i)
result = Milan_CDR_collection.aggregate([
{
    "$group" :
            {"_id": "$cellid",
            "maxsmsout": { "$max": "$smsout" }}
},
{"$sort": {"maxsmsout":-1}},#ordeno de mayor a menor
{"$limit": 1}#cojo solo dos
])
for i in result:
  print(i)
result = Milan_CDR_collection.aggregate([
{
    "$group" :
            {"_id": "$cellid",
            "maxcallin": { "$max": "$callin" }}
},
{"$sort": {"maxcallin":-1}},#ordeno de mayor a menor
{"$limit": 1}#cojo solo dos
])
```

```
for i in result:
  print(i)
result = Milan_CDR_collection.aggregate([
{
    "$group" :
            {"_id": "$cellid",
            "maxcallout": { "$max": "$callout" }}
},
{"$sort": {"maxcallout":-1}},#ordeno de mayor a menor
{"$limit": 1}#cojo solo dos
])
for i in result:
  print(i)
result = Milan_CDR_collection.aggregate([
{
    "$group" :
            {"_id": "$cellid",
            "maxinternet": { "$max": "$internet" }}
},
{"$sort": {"maxinternet":-1}},#ordeno de mayor a menor
{"$limit": 1}#cojo solo dos
])
for i in result:
  print(i)
```

Para este código, donde se agrupan las celdas y se busca por cada parámetro indicado, los resultados fueron los siguientes:

```
{'_id': 4874, 'maxsmsin': 649.4864990436}
{'_id': 7355, 'maxsmsout': 541.3159225827}
{'_id': 5259, 'maxcallin': 260.3757649662}
{'_id': 5059, 'maxcallout': 304.6983936848}
{'_id': 5061, 'maxinternet': 5882.4640404064}
```

Como podemos ver, la celda 4874 es la celda con mayor sms de entrada, con un total de 649.48 sms.
La celda 7355 es la celda con mayor sms de salida, con un total de 541.31 sms.
La celda 5259 es la celda con mayor llamadas de entrada, con un total de 260.37 llamadas.
La celda 5059 es la celda con mayor llamadas de salida, con un total de 304.69 llamadas.
La celda 5061 es la celda con mayor uso de internet, con un total de 5882.46.

Respecto a la celda con mayor actividad, se ha realizado la suma de todas las columnas con actividad para cada celda, se realizó con el siguiente código:

```
result = Milan_CDR_collection.aggregate([
{ "$group" :
    {"_id": "$cellid",
    "sumactividad": { "$sum": { "$add" : [
        '$smsin', "$smsin","$smsout","$callin","$callout","$internet"
    ]}}}
},
{"$sort": {"sumactividad":-1}},#ordeno de mayor a menor
{"$limit": 1}#cojo solo dos
])
for i in result:
  print(i)
```

El resultado obtenido fue el siguiente:

```
Modo peticiones
{'_id': 5059, 'sumactividad': 1723194.0117722927}
```

Como se puede observar, la celda con mayor actividad es la celda 5059, con una actividad total de 1723194 actividades ya sean sms, llamadas o internet.

6. Ejercicio:

En este apartado se pide que se cree una colección con una documento por celda en el que aparezcan los acumulados de los diferentes campos, para lograrlo se realizó el siguiente código:

```
153     result = Milan_CDR_collection.aggregate([
154     { "$group" :
155         {"_id": "$cellid",
156         "smsinacum": { "$sum": "$smsin" },
157         "smsoutacum": { "$sum": "$smsout" },
158         "callinacum": { "$sum": "$callin" },
159         "calloutacum": { "$sum": "$callout" },
160         "internetacum": { "$sum": "$internet" },
161         }
162     },
163     { "$out" : "Ej6" }
```

Para comprobar que se realizó la colección correctamente se ha comprobado con mongosh dicha colección y cuantos documentos se han insertado en dicha colección. En las siguientes imágenes se puede apreciar:

```
  {
    _id: 8513,
    smsinacum: 6874.2472540055,
    smsoutacum: 3419.4545857677,
    callinacum: 4395.3273010045,
    calloutacum: 4696.5527194327,
    internetacum: 75085.5850595784
  },
  {
    _id: 9544,
    smsinacum: 1268.6941605621,
    smsoutacum: 586.7003123192,
    callinacum: 936.3709252864,
    calloutacum: 1927.0100588192,
    internetacum: 32872.729708815
  }
]
Type "it" for more
Milan_CDR_db> db.Ej6.count({})
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or esti
matedDocumentCount.
10000
```

Como podemos apreciar se han insertado 10000 documentos que hacen referencia a cada celda con sus respectivos acumulados para cada atributo.

7. Ejercicio:

En este ejercicio se pide crear una colección con una documento por celda y hora en el que aparezcan los acumulados de los diferentes campos, el código para lograrlo fue el siguiente:

```
result = Milan_CDR_collection.aggregate([
{ "$group" :
    {"_id": {"cellid":"$cellid","hour":{"$hour": {"$toDate": "$time"}}},
    "smsinacum": { "$sum": "$smsin" },
    "smsoutacum": { "$sum": "$smsout" },
    "callinacum": { "$sum": "$callin" },
    "calloutacum": { "$sum": "$callout" },
    "internetacum": { "$sum": "$internet" },
    }
},
{ "$out" : "Ej7" }
], allowDiskUse=True)
```

Como podemos ver es similar al caso anterior pero con la diferencia de tener que trabajar con datetime tras transformar el dato long. Se añade allowDiskUse, esto permite usar el disco en el caso de que la operación group ocupe demasiado espacio.

En la siguiente imagen podemos ver como se ha creado correctamente la colección con los datos pedidos anteriormente.

```
test> use Milan_CDR_db
switched to db Milan_CDR_db
Milan_CDR_db> db.Ej7.count({})
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or esti
matedDocumentCount.
240000
```

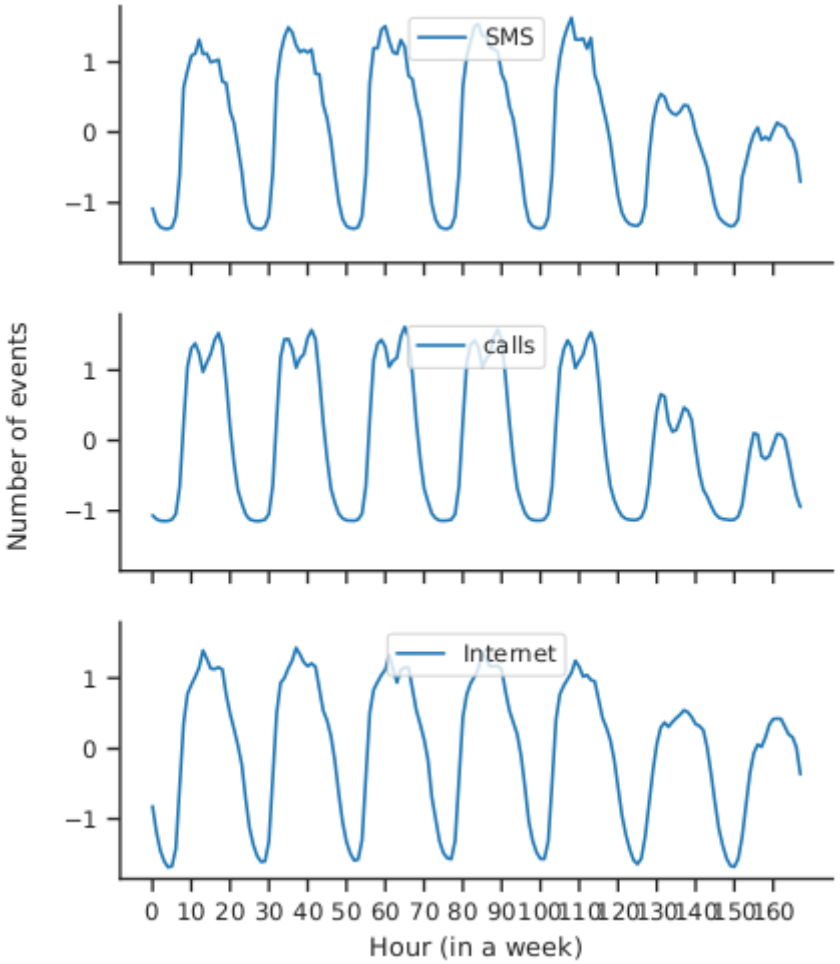
En la siguiente imagen se puede ver la estructura:

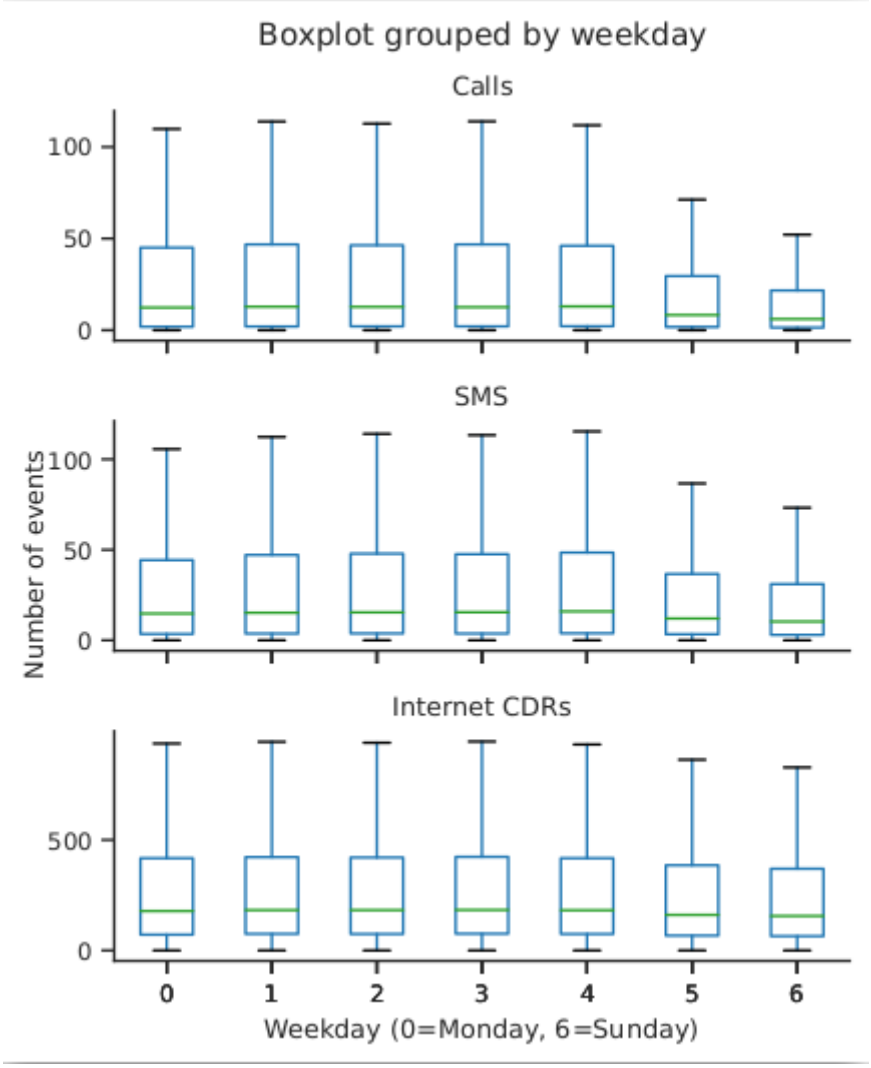
```
{
  _id: { cellid: 1, hour: 18 },
  smsinacum: 43.1327726816,
  smsoutacum: 26.5954633283,
  callinacum: 30.8095613096,
  calloutacum: 29.7620182723,
  internetacum: 611.2828221606001
},
{
  _id: { cellid: 1, hour: 19 },
  smsinacum: 36.8426125841,
  smsoutacum: 29.0124723154,
  callinacum: 20.8850791662,
  calloutacum: 23.8983004525,
  internetacum: 665.0795532749
}
]
```

8. Ejercicio:

Por último, en este apartado se pide realizar un estudio de las celdas 4259 (Bocconi), 4456 (Navigli), 5060 (Duomo), 1419 (terreno de agricultura), 2436 (área industrial), 4990 (aeropuerto de Linate), 945 (residencial aislado) y 5048 (residencial céntrico):

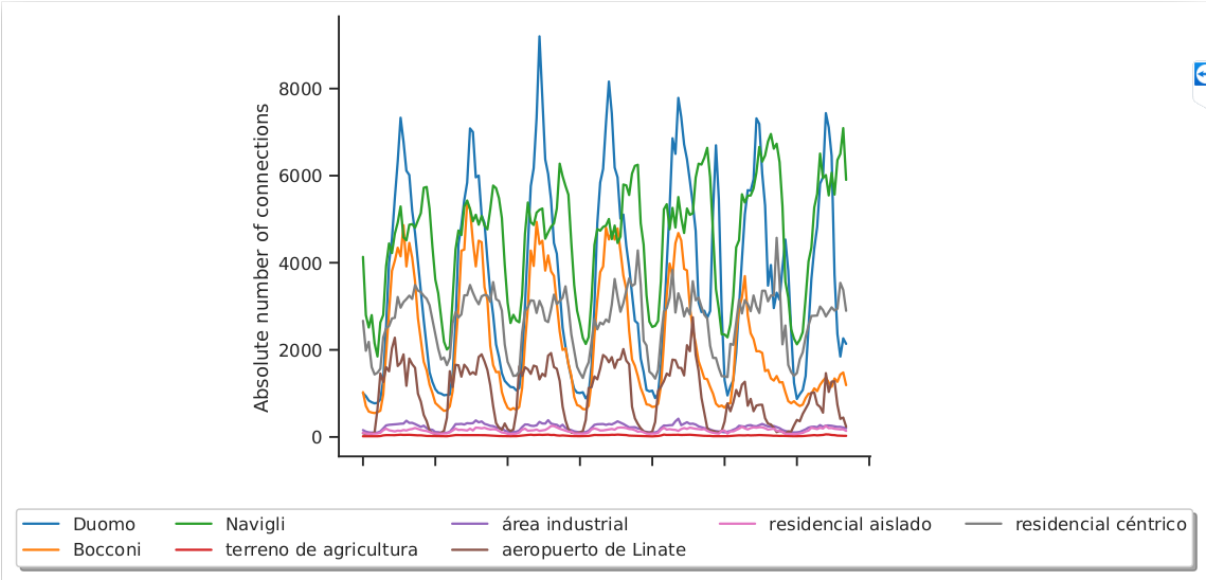
Para la realización de este apartado se ha realizado una modificación del código dado por Marco De Nadai, ajustándose a las necesidades de este ejercicio. Antes de analizar cada celda se crearon la siguiente gráfica y boxplot para analizar la actividad general (llamadas, sms e internet). Las gráficas son las siguientes:





Como podemos observar en las gráficas se nos muestra cada tipo de actividad en una semana, obviando el tipo de celda. Como se puede observar, la actividad en general es muy similar, es decir, que tanto sms como llamadas e internet suelen tener la misma cantidad de actividad. Si acaso, internet si que tiene un poco más de actividad, sobre todo en los fines de semana.

Por ello, para simplificar, se va a generar la gráficas de la celda para la actividad que se realiza con internet. La gráfica es la siguiente:



Como podemos ver en la leyenda, se nos muestra para cada color una celda distinta. Si analizamos la gráfica como tal, se puede extraer mucha información interesante.
Por ejemplo, las celdas de “terreno de agricultura”, “área industrial” y “residencial aislado”, como podemos ver tienen una actividad mínima de internet, esto es lógico ya que son zonas con una cantidad de gente bastante reducida. En el caso contrario, tanto Duomo como Navigli son celdas con gran actividad de internet, esto se debe a que, por ejemplo, Duomo es el centro de la ciudad y la atracción turística más importante, y Navigli es una zona de ocio nocturno, por ello, durante altas horas de la noche, o incluso más los fines de semana, la actividad aumenta.

Si nos fijamos más en Bocconi, como es una zona universitaria tiene bastante actividad pero cuando llegan altas horas de la noche y sobre todo el fin de semana, la actividad cae en gran medida. Algo similar ocurre con el aeropuerto de Linate, cabe destacar que el viernes hay un pico de actividad, esto puede deberse a que hay un repunte de viajes para el fin de semana.

Respecto al residencial céntrico, suele ser bastante estable ya que es el lugar donde vive la gente, como mucho se puede destacar una pequeña subida de actividad los sábados a altas horas, esto tiene sentido ya que la mayoría de gente no trabajará el domingo.

Para finalizar, si comparamos las tres celdas con menos actividad podemos decir que las celdas “área industrial” y “residencial aislado”, todavía tienen algo de actividad aunque sea mínima. La celda de “terreno de agricultura” es prácticamente una línea recta sin actividad a internet, esto es bastante lógico ya que la cantidad de gente en esta zona es muy reducida y aparte, es probable que el uso de internet en esa zona sea muy reducido si no se usa iot o alguna tecnología que necesite de internet.

Respecto al código, se adjunta la siguiente imagen del código modificado:

```
81    f, axs = plt.subplots(1, sharex=True, sharey=False, figsize=fig_size2)
82
83    axs.plot(sliceSum[sliceSum.CellID == 5060].set_index('idx')['internet'], label='Duomo')
84    axs.plot(sliceSum[sliceSum.CellID == 4259].set_index('idx')['internet'], label='Bocconi')
85    axs.plot(sliceSum[sliceSum.CellID == 4456].set_index('idx')['internet'], label='Navigli')
86
87    axs.plot(sliceSum[sliceSum.CellID == 1419].set_index('idx')['internet'], label='terreno de agricultura')
88    axs.plot(sliceSum[sliceSum.CellID == 2436].set_index('idx')['internet'], label='área industrial')
89    axs.plot(sliceSum[sliceSum.CellID == 4990].set_index('idx')['internet'], label='aeropuerto de Linate')
90    axs.plot(sliceSum[sliceSum.CellID == 945].set_index('idx')['internet'], label='residencial aislado')
91    axs.plot(sliceSum[sliceSum.CellID == 5048].set_index('idx')['internet'], label='residencial céntrico')
92    axs.set_xticklabels([])
93    sns.despine()
94    # Shrink current axis's height by 10% on the bottom
95    box = axs.get_position()
96    axs.set_position([box.x0, box.y0 + box.height * 0.1, box.width, box.height * 0.9])
97    axs.legend(loc='upper center', bbox_to_anchor=(0.5, -0.10), fancybox=True, shadow=True, ncol=5)
```

Lógicamente hay más cambios pero son menores, sobre todo de funciones inexistentes y ajustes al diseño de las gráficas, etc. En la imagen se pueden ver los cambios más notorios, donde se han añadido a la gráfica las celdas pedidas por el ejercicio.