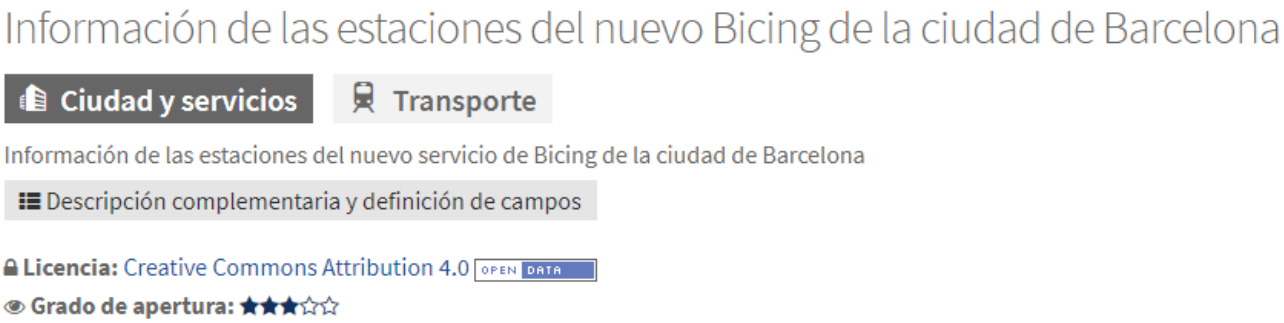
1. Datos Seleccionados

Seleccione los datos abiertos de las estaciones de bicicletas eléctricas del Servicio de datos abiertos del Ayuntamiento de Barcelona, y tiene una calificación de 3 estrellas en el grado de apertura:

Url: https://opendata-ajuntament.barcelona.cat/data/es/dataset/informacio-estacions-bicing



La Fuente de datos esta en constante actualización, y la calidad de los datos es buena porque ofrece atributos específicos y de fácil comprensión, el siguiente objeto es extraído del conjunto de datos:


```
{
    "last_updated": 1585457269,
    "ttl": 18,
    "data": {
        "stations": [
            {
                "station_id": 1,
                "name": "GRAN VIA CORTS CATALANES, 760",
                "physical_configuration": "ELECTRICBIKESTATION",
                "lat": 41.397952,
                "lon": 2.180042,
                "altitude": 21.0,
                "address": "GRAN VIA CORTS CATALANES, 760",
                "post_code": "08908",
                "capacity": 46,
                "nearby_distance": 1000.0
            },
            {
                "station_id": 2,
                "name": "C/ ROGER DE FLOR, 126",
                "physical_configuration": "ELECTRICBIKESTATION",
                "lat": 41.39553,
                "lon": 2.17706,
                "altitude": 21.0,
                "address": "C/ ROGER DE FLOR, 126",
                "post_code": "08908",
                "capacity": 27,
                "nearby_distance": 1000.0
```

La estructura del dataset es una lista que contiene diccionarios, cada diccionario es una estación con sus atributos:

```
{
"station_id": 1,                                    #Identificador de la Estación
"name": "GRAN VIA CORTS CATALANES, 760",            #Nombre de la Estación
"physical_configuration": "ELECTRICBIKESTATION",    #Tipo de Estación
"lat": 41.397952,                                   #Latitud
"lon": 2.180042,                                    #Longitud
"altitude": 21.0,                                   #Altitud
"address": "GRAN VIA CORTS CATALANES, 760",         #Dirección especifica
"post_code": "08908",                               #Codigo Postal
"capacity": 46,                                     #Capacidad
"nearby_distance": 1000.0                           #Distancia de acercamiento
},
```

Para esta actividad el dataset selecionado tiene 445 diccionarios ósea la información de 445 Estaciones.

1. Script

Todos los archivos de programación de pycharm y Jupiter Notebook, se encuentran en GitHub en la dirección:

https://github.com/alexanderbasulto/master_iot/tree/master/actividad_1

a. main:

```
if __name__ == "__main__":                          #Programa principal
    datos_json = descarga_datos()                   #Llama a la funcion descarga datos y devuelve el valor a la variab
    crear_archivo(datos_json)                       #Llama a la funcion crear archivo, para crear el documento .json
    print("Archivo creado....")                     #Imprime en consola que el documento fue creado
    url_mongo = mongodb_connect()                   #Llama a la funcion mongodb connect para cargar la dirrecion de Mo
    mongodb_send(data_to_send=datos_json, dbmongo=url_mongo)    #Llama a la funcion mongodb send para enviar laos dato
    print("Datos enviados a MongoDB....")           #Imprime en consola que los datos fueron enviados
    get_attr(url_mongo)                             #Consulta sobre un atributo numerico en todos los datos
    get_top_3(url_mongo)                            #Devuelve el top 3 de las estaciones segun el atributo escogido
    get_media(url_mongo)                            #Calcula la media de un atributo de todos los datos
    print("Programa terminado.")
```

Esta es la rutina principal, desde aquí se llama a las otras funciones.

b. descarga_datos ( ):

```
def descarga_datos():          #Descargar datos de URL y crear variable diccionario
    url = 'https://api.bsmsa.eu/ext/api/bsm/gbfs/v2/en/station_information'
    contents = urllib.request.urlopen(url).read()
    datos_0 = json.loads(contents.decode('utf-8'))
    return datos_0
```

Se carga en la variable "url" la dirección del data set en formato JSON que se va a trabajar, luego se usa la función "loads" de la librería JSON para cargar la cadena de datos en una variable, usamos el operador "decode" para normalizar el formato de los datos.

La cadena de datos descargadas del JSON se guarda y se devuelve en la variable "datos_0".

c. crear_archivo ( ):

```
def crear_archivo(datos_file):       #Crear y escribir archivo
    file = open('datos.json', 'w')
    file.write(json.dumps(datos_file, indent=4))
    file.close()
    return()
```

Se abre el archivo "datos.json" en modo escritura "w", y escribimos en el con la función de la librería JSON "dumps", que convierte los datos en una cadena JSON, luego cerramos el archivo.

d. mongodb_connect ( ):

```
def mongodb_connect():
    client = pymongo.MongoClient("URL DE MONGO DATABAES CUN USER Y PASS")
    db = client.bd_actividad
    url_dbmongo = db.actividad_1
    return url_dbmongo
```

Llamamos a esta función para cargar en una variable el direccionamiento completo a la URL de mongo, y así no tener que estar escribiendo constantemente todo.

e. Mongodb_send ( ):

```
def mongodb_send(data_to_send,dbmongo):
    datos_data = data_to_send.get('data')
    datos_stations = datos_data.get('stations')
    dbmongo.insert(datos_stations)
    return()
```

Con esta función en la variable "dbmongo" tenemos guardado el direccionamiento a nuestra base de datos de Mongo, y en "data_to_send" los datos que van a ser insertados en la base de datos. Pero antes de enviar los datos tenemos que quedarnos únicamente con los objetos o diccionarios del dataset para insertarlos por separados en la colección.

f. get_attr ( ):

```
def get_attr(dbmongo):
    print()
    print("La estaciones con una capacidad de 30 son:")
    for x in dbmongo.find({"capacity": 30}):  # Consulta a MondoGB
        print(x)
    return()
```

Con esta función vamos a realizar una consulta en Mongo, y nos va devolver las estaciones que tengan una capacidad de 30. Usamos un for para recurrer con el puntero cada objeto devuelto y luego imprimirlo.

g. get_top_3 ( ):

```
def get_top_3(dbmongo):
    print()
    print("Las 3 estaciones con mayor capacidad, de mayor a menor son:")
    for x in dbmongo.find({"capacity": {"$gt": 1}}).sort("capacity", -1).limit(3):
        print(x)
    return()
```

Esta función va buscar las 3 estaciones con mayor capacidad y las va mostrar de mayor a menor, usamos el operador ".limit( )" el cual limita la búsqueda según el numero que necesitemos.

a. get_media ( ):

```
def get_media(dbmongo):
    print()
    print("Se va calcular la media de la Altitud de todas las estaciones:")
    for x in dbmongo.aggregate([{"$group": {"_id": 1, "promedio": {"$avg":"$altitude"}, "Total de Estaciones": {"$sum": 1}}}]):
        print(x)
    return()
```

Para esta función se pretende calcular la media o promedio, de la altitud de todas las estaciones, para eso se usa la función aggregate de la librería mongo. El cual nos permite realizar cálculos y otras operaciones sobre una colección, pero para este ejemplo usamos los operadores "$avg" para calcular la media y "$sum" para saber la cantidad de valores o objetos que tenemos.