Trabajo Programación Paralela 2022-2023

Análisis de datos con Spark de BiciMAD

UNIVERSIDAD
COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

Autores: **Javier Castellano Soria e Ignacio Fernández Sánchez-Pascuala**

Profesor: **Luis Fernando Llana**

Índice

1. Introducción.

El objetivo de esta práctica es el planteamiento, diseño e implementación de un análisis de datos reales por medio de la librería *pySpark* de *Python*. El conjunto de datos utilizado representa el uso de bicicletas de préstamo de *BiciMAD*, proporcionado por el Ayuntamiento de Madrid.

Analizaremos el uso de las bicicletas de *BiciMAD* en el año 2017, teniendo en cuenta una serie de parámetros que nos llevarán a distintas conclusiones.

2. Conjunto de datos

Los datos de los que disponemos sobre *BiciMAD* recogen los meses desde Abril de 2017 hasta junio de 2018. Los datos de cada mes vienen por separado en un archivo de extensión *.json*.

Cada línea de estos ficheros es un objeto de *JavaScript* con los siguientes parámetros:

- **_id:** Identificador de movimiento
- **user_day_code:** Código de usuario diario
- **idplug_base:** Número base en la que se engancha la bicicleta
- **user_type:** Número que indica el tipo de usuario que ha realizado el movimiento. Sus posibles valores son:
  - 0: No se ha podido determinar el tipo de usuario
  - 1: Usuario anual (poseedor de un pase anual)
  - 2: Usuario ocasional
  - 3: Trabajador de la empresa
- **idunplug_base:** Número de la base de la que se desengancha la bicicleta.
- **travel_time:** Tiempo total en segundos, entre el desenganche y el enganche de la bicicleta.
- **idunplug_station:** Número de la estación de la que se desengancha la bicicleta.
- **ageRange:** Número que indica el rango de edad del usuario que ha realizado el movimiento. Sus posibles valores son:
  - 0: No se ha podido determinar el rango de edad del usuario
  - 1: El usuario tiene entre 0 y 16 años
  - 2: El usuario tiene entre 17 y 18 años
  - 3: El usuario tiene entre 19 y 26 años
  - 4: El usuario tiene entre 24 y 40 años
  - 5: El usuario tiene entre 41 y 65 años

    - 6: El usuario tiene 66 años o más

- **idplug_station:** Número de la estación de la que se engancha la bicicleta.
- **unplug_hourTime:** Franja horaria en la que se realiza el desenganche de la bicicleta. Todos los movimientos iniciados durante la misma hora, tendrán el mismo dato de inicio.
- **zip_code:** Texto que indica el código postal del usuario que ha realizado el movimiento.

3. Problema a Resolver

Para realizar un estudio que pueda tener interés para la sociedad, hemos seleccionado todos los datos desde Abril de 2017 hasta Diciembre de 2017 y hemos analizado distintos puntos que hemos considerado interesantes sobre este conjunto de datos:

- Media de usuarios únicos para cada día de la semana.
- Media de usuarios únicos para cada mes de los analizados.
- Tiempo medio por edad para cada viaje.
- Tiempo medio de viaje para cada día de la semana.
- Tiempo medio de viaje según la edad.
- Por cada día de la semana, enganches y desenganches de cada estación.
- Por cada estación, enganches y desenganches según el día de la semana.

Como podemos observar, para los estudios solo serán necesarios los siguientes parámetros de los datos: user_day_code, travel_time, idunplug_station, ageRange, idplug_station y unplug_hourTime.

4. Solución del Problema y Análisis Resultados

Para trabajar de forma paralela con los datos, trabajaremos con *RDDs*. A continuación veremos cómo convertir los datos de partida en un *rdd* de tal forma que recoja la información que necesitamos. Posteriormente llevamos a cabo los estudios introducidos en el apartado anterior.

4.1. Lectura de los archivos JSON y filtrado de parámetros.

En primer lugar, queremos transferir los datos desde los archivos *JSON* a un *RDD* mediante la biblioteca *Spark*, para trabajar paralelamente con ellos. Para ello, creamos un contexto de *Spark* como punto de entrada principal, que lo llamaremos *sc*:

```
conf = SparkConf().setAppName("Bicimad Estudios")
sc = SparkContext(conf=conf)
sc.setLogLevel("ERROR")
```

Inicializamos nuestro RDD vacío a partir del *sc* creado mediante el método *.parallelize()* y añadimos a este el contenido de los archivos *.json* de los meses y años de las listas *years* y *months*. La instrucción *sc.textFile(filename)* lee el contenido de un archivo y lo carga en un RDD. La función *union()* combina los RDDs de cada archivo. Así conseguimos construir nuestro "rdd" con toda la información, donde cada elemento representa una linea de texto:

```
def main(sc, years, months, datadir):

    rdd = sc.parallelize([])
    for y in years:
        for m in months:
            filename = f"{datadir}/{y}{m:02}_movements.json"
            print(f"Adding {filename}")
            rdd = rdd.union(sc.textFile(filename))
```

Para filtrar los parámetros que necesitamos para el estudio, creamos la función *datos* que dada una linea de archivo en formato *JSON*, nos devuelve la información deseada. Utilizamos el método *json.loads* de la librería *json* que transforma una linea en un diccionario y la librería *datetime* para manejar las fechas de forma más práctica, con sus funciones predefinidas *weekday()*, *month()* y *strptime()*:

```
def datos(line):

    data = json.loads(line)
    origen = data['idunplug_station']
    destino=data['idplug_station']
    fecha = datetime.strptime(data['unplug_hourTime']['$date']\
                              , "%Y-%m-%dT%H:%M:%S.%f%z")
    mes=fecha.month
    weekday=fecha.weekday()
    tiempo = data['travel_time']
    edad = data['ageRange']
    codigo_usuario = data['user_day_code']
    return origen, destino, weekday, edad, tiempo, codigo_usuario, mes
```

Una vez definida la función *datos*, se la aplicamos a cada una de los elementos de nuestro "rdd" mediante el método *map*, creando un "rdd_base" de partida para los futuros estudios. Además calculamos el número total de semanas aproximado en los que se lleva a cabo el estudio para hallar luego algunos promedios por día de la semana:

```
    n_semanas = len(years)*len(months)*4
    n_years = len(years)
    rdd_base=rdd.map(datos)
```

4.2. Media usuarios únicos por día de la semana y mes.

Queremos ver el promedio de usuarios únicos por día de la semana y mes. El estudio de los días de la semana lo realizamos con la función auxiliar *usua-*

rios_unicos_dias, que dado el "rdd_base" y el número de semanas, crea otro RDD cuyos elementos son tuplas, donde el primer elemento es el día de la semana y el segundo su promedio de usuarios únicos. Además, guarda estos datos en un archivo de texto:

```
def usuarios_unicos_dias(rdd_base,n_semanas):

    rdd_usuarios_dias = rdd_base.map(lambda x: (x[2],x[5]))\
        .distinct()\
        .map(lambda x: (x[0],1)).reduceByKey(lambda a, b:a+b)\
        .map(lambda x: (x[0],x[1]/n_semanas))
    lista_usuarios_dias=rdd_usuarios_dias.collect()

    f = open('usuarios_unicos_por_dia.txt', 'w')
    for (weekday, media) in lista_usuarios_dias:
        print(f'Dia de la semana {weekday} ---->\
              Media de usuarios unicos {media}')
        f.write(f'Dia de la semana {weekday} ---->\
                Media de usuarios unicos {media}\n')
    f.close()
```

Con el primer *map* seleccionamos los datos que queremos: weekday y user_day_code. Como no quiero contar los usuarios que repiten día hago *.distinct()*. Luego me quedo con los días de la semana (clave) y pongo un 1 como valor para hacer el conteo con *reduceByKey()*. Después divido entre el número de semanas para tener el promedio por día de la semana.

Este es el resultado del estudio:

```
Día de la semana 0 ----> Media de usuarios únicos 6145.305555555556
Día de la semana 1 ----> Media de usuarios únicos 6587.361111111111
Día de la semana 2 ----> Media de usuarios únicos 6776.833333333333
Día de la semana 3 ----> Media de usuarios únicos 6753.972222222223
Día de la semana 4 ----> Media de usuarios únicos 6795.027777777777
Día de la semana 5 ----> Media de usuarios únicos 5411.583333333333
Día de la semana 6 ----> Media de usuarios únicos 4985.361111111111
```

De forma análoga, se realiza el estudio para el promedio de usuarios únicos por mes:

```
def usuarios_unicos_meses(rdd_base,n_years):

    rdd_usuarios_meses = rdd_base.map(lambda x: (x[6],x[5]))\
        .distinct()\
        .map(lambda x: (x[0],1)).reduceByKey(lambda a, b:a+b)\
        .map(lambda x: (x[0],x[1]/n_years))

    f = open('usuarios_unicos_por_mes.txt', 'w')
    for (mes, media) in rdd_usuarios_meses.collect():
        print(f'Mes {mes} ----> Media de usuarios unicos {media}')
        f.write(f'Mes {mes} ----> Media de usuarios unicos {media}\n')
    f.close()
```

Obteniendo el siguiente resultado del estudio para los meses seleccionados:

```
Mes 5 ----> Media de usuarios únicos 184842.0
Mes 6 ----> Media de usuarios únicos 205266.0
Mes 7 ----> Media de usuarios únicos 184696.0
Mes 8 ----> Media de usuarios únicos 147230.0
Mes 9 ----> Media de usuarios únicos 217755.0
Mes 10 ----> Media de usuarios únicos 224996.0
Mes 11 ----> Media de usuarios únicos 189369.0
Mes 12 ----> Media de usuarios únicos 141922.0
```

Adjuntamos unas gráficas ilustrativas de los estudios anteriormente realizados.

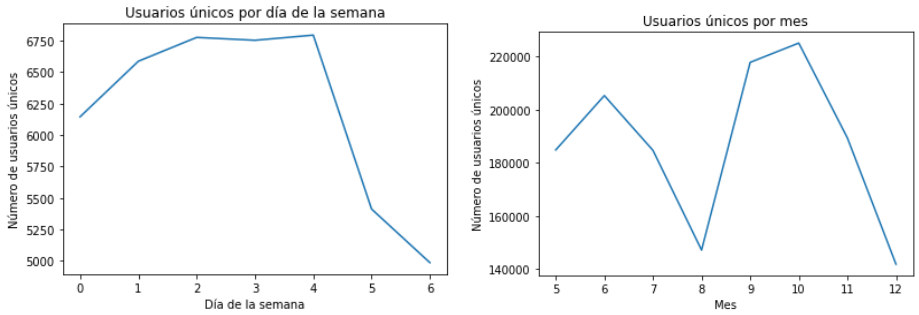

Figura 1: *Plots* estudios media usuarios únicos por día y mes

Como podemos observar en 1a, es más frecuente el uso de las bicicletas de préstamo durante los días de diario. Además, también podemos observar en 1b que hay menos usuarios en los meses donde hace más calor, como Julio y Agosto, y también cuando hace más frío, como en Diciembre.

4.3. Análisis Tiempos Medios según edad, día de la semana y viaje.

Primero veamos la duración media de cada viaje realizado en función de la edad. Para ello creamos una función que dado el "rdd_base" hallamos otro cuyos elementos son tuplas con el viaje y una lista de tuplas (edad,tiempo). Además, guardamos los datos en un fichero de texto.

```
def tiempo_medio_viaje_por_edad(rdd_base):

    rdd_tiempo_medio_viaje_por_edad = rdd_base\
        .map(lambda x: ((x[0], x[1], x[3]), x[4]))\
        .filter(lambda x: x[0][2]>0)\
        .mapValues(lambda x: (x,1))\
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))\
        .mapValues(lambda x: x[0]/x[1])\
        .map( lambda x: ( ((x[0][0],x[0][1]), (x[0][2], x[1]))))\
        .groupByKey()\
        .mapValues(lambda x: sorted(list(x), key=lambda y: y[0]))
```

```
f = open("tiempo_medio_viaje_por_edad.txt", "w")
for (viaje, edad_tiempo) in rdd_tiempo_medio_viaje_por_edad\
    .collect():
    print(f'Origen {viaje[0]} - Destino {viaje[1]}')
    f.write(f'Origen {viaje[0]} - Destino {viaje[1]}\n')
    for (edad, tiempo) in edad_tiempo:
        print(f'Grupo de edad {edad} ----> \
              Tiempo promedio: {tiempo}')
        f.write(f'Grupo de edad {edad} ---->\
                Tiempo promedio: {tiempo}\n')
    f.write('\n\n')
    print()
f.close()
```

Con el primer *map* seleccionamos los datos que nos interesan: origen, destino, edad y tiempo, dejando los tres primeros argumentos como una tupla. Filtramos aquellos cuyo grupo de edad sea > 0, es decir, el grupo de edad es conocido. Después hago un *map* a los valores (tiempo) para ponerlo como tuplas con un 1 que hará de contador. Luego, hago un reduceByKey sumando los valores y un mapValues que coge el tiempo total y lo divide entre el contador obteniendo así el tiempo promedio. Después hacemos un map y un groupByKey para organizar los datos por tupla (origen, destino) (clave). Por último, ordeno los tiempos de cada viaje por grupo de edad.

Una parte de los datos obtenidos es:

```
Origen 52 - Destino 38
Grupo de edad 1 ----> Tiempo promedio: 524.0
Grupo de edad 2 ----> Tiempo promedio: 2033.0
Grupo de edad 3 ----> Tiempo promedio: 745.56
Grupo de edad 4 ----> Tiempo promedio: 684.2045454545455
Grupo de edad 5 ----> Tiempo promedio: 628.6153846153846


Origen 40 - Destino 129
Grupo de edad 3 ----> Tiempo promedio: 815.0
Grupo de edad 4 ----> Tiempo promedio: 1117.9777777777779
Grupo de edad 5 ----> Tiempo promedio: 843.3770491803278
```

Ahora estudiemos los tiempos medios de viaje por día de la semana. El resultado va a ser un archivo de texto con el día de la semana y el tiempo promedio.

```
def tiempo_medio_por_dia_semana(rdd_base):

    rdd_tiempo_medio_por_dia = rdd_base\
        .map(lambda x: (x[2], (x[4], 1)))\
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))\
        .mapValues(lambda x: x[0]/x[1])

    f = open("tiempo_medio_por_dia_semana.txt", "w")
    for (dia, tiempo) in \
    sorted(list(rdd_tiempo_medio_por_dia.collect()), key=lambda y: y[0]):
        print(f'Dia {dia} ----> Tiempo promedio: {tiempo}')
        f.write(f'Dia {dia} ----> Tiempo promedio: {tiempo}\n')
    print()
    f.close()
```

Con el primer *map* seleccionamos los datos que nos interesan: weekday y tiempo. Además aprovechamos para poner weekday como clave y la tupla (tiempo, 1) como valor. Hacemos un *reduceByKey* para obtener el tiempo total por día de la semana y número de viajes. Terminamos con un *mapValues* para hallar el promedio.

Los datos obtenidos son:

```
Día 0 ----> Tiempo promedio: 1151.0067408231369
Día 1 ----> Tiempo promedio: 1192.7087450706426
Día 2 ----> Tiempo promedio: 1145.2980669149995
Día 3 ----> Tiempo promedio: 1134.3081269003603
Día 4 ----> Tiempo promedio: 1179.4564551412177
Día 5 ----> Tiempo promedio: 1281.2865189377396
Día 6 ----> Tiempo promedio: 1355.3398103184174
```

Por último realizamos el mismo estudio pero ahora por edad. El código es análogo al anterior:

```
def tiempo_medio_por_edad(rdd_base):

    rdd_tiempo_medio_por_edad = rdd_base\
        .map(lambda x: (x[3], (x[4], 1)))\
        .filter(lambda x: x[0] > 0)\
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))\
        .mapValues(lambda x: x[0]/x[1])

    f = open("tiempo_medio_por_edad.txt", "w")
    for (edad, tiempo) in \
    sorted(list(rdd_tiempo_medio_por_edad.collect()), key=lambda y: y[0]):
        print(f'Grupo de edad {edad} ---->\
              Tiempo promedio: {tiempo}')
        f.write(f'Grupo de edad {edad} ---->\
                Tiempo promedio: {tiempo}\n')
    print()
    f.close()
```

Los datos obtenidos son:

```
Grupo de edad 1 ----> Tiempo promedio: 1343.5273299019534
Grupo de edad 2 ----> Tiempo promedio: 1114.4512439040834
Grupo de edad 3 ----> Tiempo promedio: 958.651384896252
Grupo de edad 4 ----> Tiempo promedio: 1177.5107162656661
Grupo de edad 5 ----> Tiempo promedio: 1194.906066205867
Grupo de edad 6 ----> Tiempo promedio: 1162.5616527256996
```

Las gráficas de los dos últimos estudios son las siguientes:

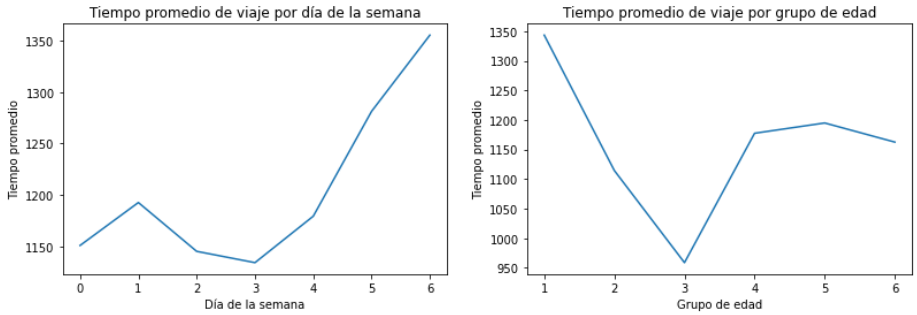

Figura 2: *Plots* estudios tiempos promedios

Como podemos observar en 2a, los fines de semana los viajes son más largos, ya que los usuarios usan las bicicletas de forma más lúdica y menos funcional. Por otro lado, podemos ver en 2b que los usuarios más rápidos son los de entre 19 y 26 años, debido a sus mejores capacidades físicas probablemente.

4.4. Usos de bicicletas con misma estación de enganche y desenganche.

Ahora vamos a calcular la cantidad de viajes cuya estación de enganche es la misma que de desenganche por cada estación. Esto podría ser un indicativo de qué estaciones suelen usarse para darse un paseo en bicicleta. Como anteriormente, usamos una función que tomará como entrada el "rdd_base" y escribirá en un archivo de texto la cantidad de viajes de este estilo por cada estación.

```
def viajes_mismo_origen_destino_por_estacion(rdd_base):

    rdd_viajes_mismo_origen_destino = rdd_base\
        .map(lambda x: (x[0], x[1]))\
        .filter(lambda x: x[0] == x[1])\
        .map(lambda x: (x[0], 1))\
        .reduceByKey(lambda x, y: x + y)

    f = open("viajes_mismo_origen_destino_por_estacion.txt", "w")
    for (estacion, n_viajes) in \
    sorted(list(rdd_viajes_mismo_origen_destino.collect())\
            , key=lambda y: y[1], reverse=True):
        print(f'Numero de estacion {estacion} ---->\
            Numero de viajes con mismo origen y destino: {n_viajes}')
        f.write(f'Numero de estacion {estacion} ---->\
            N mero de viajes con mismo origen y destino: {n_viajes}\n')
    print()
    f.close()
```

Con el primer *map* seleccionamos los datos que nos interesan: origen y destino. Después con el *filter* nos quedamos solo con las tuplas que tienen origen y destino idénticos. Con el *map* nos quedamos con el número de estación y ponemos 1 como valor para contar el número de viajes por estación con el *reduceByKey*.

Una parte de los datos obtenidos (ordenados de mayor a menor número de viajes) es:

```
Número de estación 64 ----> Número de viajes con mismo origen y destino: 3134
Número de estación 135 ----> Número de viajes con mismo origen y destino: 2962
Número de estación 132 ----> Número de viajes con mismo origen y destino: 2525
Número de estación 90 ----> Número de viajes con mismo origen y destino: 2171
Número de estación 163 ----> Número de viajes con mismo origen y destino: 1901
Número de estación 43 ----> Número de viajes con mismo origen y destino: 1868
Número de estación 129 ----> Número de viajes con mismo origen y destino: 1849
Número de estación 168 ----> Número de viajes con mismo origen y destino: 1809
Número de estación 57 ----> Número de viajes con mismo origen y destino: 1679
Número de estación 149 ----> Número de viajes con mismo origen y destino: 1587
Número de estación 74 ----> Número de viajes con mismo origen y destino: 1563
Número de estación 78 ----> Número de viajes con mismo origen y destino: 1540
Número de estación 175 ----> Número de viajes con mismo origen y destino: 1527
Número de estación 83 ----> Número de viajes con mismo origen y destino: 1523
Número de estación 75 ----> Número de viajes con mismo origen y destino: 1490
```

4.5. Relaciones enganches y desenganches por estación y día de la semana.

En este apartado vamos a ver las relaciones entre enganches y desenganches según la estación y día de la semana.

En primer lugar, vamos a calcular, para cada día de la semana, por un lado la frecuencia media de enganches y por otro la frecuencia media de desenganches de cada estación. Para ello, usaremos una función que dado el "rdd_base" y el número de semanas, hallamos otros 2 cuyos elementos son tuplas, donde el primer elemento es el día de la semana, y el segundo una lista de tuplas (origen/destino, frecuencia). Además, guardamos estos datos en 2 archivos de texto según sean enganches o desenganches.

```
def estudio_frecuencias_dias(rdd_base,n_semanas):

    rdd_frec_origen = rdd_base.map(lambda x: ((x[2],x[0]), 1))\
        .reduceByKey(lambda a, b: a + b)

    rdd_orden_origen = rdd_frec_origen\
        .map(lambda x: (x[0][0], (x[0][1], x[1]/n_semanas)))\
        .groupByKey()\
        .mapValues(lambda x: sorted\
                   (list(x), key=lambda y: y[1], reverse=True))

    f = open('desenganches_por_dia_por_estacion.txt', 'w')
    for (weekday, origen_frec) in rdd_orden_origen.collect():
        print(f"Dia: {weekday}")
        f.write(f"Dia: {weekday}\n")
        for (origen, frec) in origen_frec:
            print(f"Estacion de desenganche: {origen} --->\
                  Frecuencia: {frec}")
            f.write(f"Estacion de desenganche: {origen} --->\
                    Frecuencia: {frec}\n")
        print()
        f.write('\n\n')
    f.close()
```

```
rdd_frec_destino = rdd_base.map(lambda x: ((x[2],x[1]), 1))\
    .reduceByKey(lambda a, b: a + b)

rdd_orden_destino = rdd_frec_destino\
    .map(lambda x: (x[0][0], (x[0][1], x[1]/n_semanas)))\
    .groupByKey()\
    .mapValues(lambda x: sorted\
               (list(x), key=lambda y: y[1], reverse=True))

f = open('enganches_por_dia_por_estacion.txt', 'w')
for (weekday, destino_frec) in rdd_orden_destino.collect():
    print(f"Dia: {weekday}")
    f.write(f"Dia: {weekday}\n")
    for (destino, frec) in destino_frec:
        print(f"Estacion de enganche: {destino} --->\
              Frecuencia: {frec}")
        f.write(f"Estacion de enganche: {destino} ---\
               > Frecuencia: {frec}\n")
    print()
    f.write('\n\n')
f.close()
```

Con el primer *map* seleccionamos los datos que queremos: weekday y origen. Aprovechamos para poner (weekday,origen) como clave y un 1 como valor. Después hacemos *reduceByKey* para obtener la frecuencia de desenganches.

Luego hago un *map* para poner el día de la semana como clave y el par (origen, promedio) como valor al haber dividido la frecuencia entre el número de semanas. Por último agrupo por día de la semana y ordeno las estaciones y promedio de mayor a menor según el promedio. La segunda parte del código es análoga a la anterior pero ahora con destino (enganches).

Un ejemplo de los resultados obtenidos (solo para un día de la semana), ordenados de mayor a menor según la frecuencia, tanto para desenganches como enganches:

```
Día: 0
Estación de desenganche: 163 ---> Frecuencia: 151.22222222222223
Estación de desenganche: 43 ---> Frecuencia: 146.58333333333334
Estación de desenganche: 129 ---> Frecuencia: 140.36111111111111
Estación de desenganche: 57 ---> Frecuencia: 129.47222222222223
```

```
Día: 0
Estación de enganche: 163 ---> Frecuencia: 156.36111111111111
Estación de enganche: 43 ---> Frecuencia: 155.75
Estación de enganche: 129 ---> Frecuencia: 146.61111111111111
Estación de enganche: 135 ---> Frecuencia: 134.88888888888889
```

Por otro lado, vamos a ver, dada una estación cuál es el promedio de enganches y desenganches para cada día de la semana. Para ello creamos una función que dado el "rdd_base" y el número de semanas, nos devuelve otro cuyos elementos son tuplas donde el primer elemento es la estación y el segundo una lista de tuplas cuyas componentes son día de la semana y una tupla correspondiente a las frecuencias de enganche y desenganche.

```
def estudio_frecuencias_estacion(rdd_base, n_semanas):

    rdd_llegadas = rdd_base.map(lambda x: ((x[2], x[1]), 1))\
        .reduceByKey(lambda a, b: a + b)

    rdd_salidas = rdd_base.map(lambda x: ((x[2], x[0]), 1))\
        .reduceByKey(lambda a, b: a + b)

    rdd_resultado = rdd_salidas.join(rdd_llegadas)

    rdd_frec_total = rdd_resultado\
        .map(lambda x: (x[0][1], (x[0][0],\
                        (x[1][0]/n_semanas, x[1][1]/n_semanas))))\
        .groupByKey()

    f = open("enganches_desenganches_por_estacion_por_dia.txt", 'w')
    for (station, weekday_frec) in rdd_frec_total.collect():
        print(f"Estacion {station}:")
        f.write(f"Estacion {station}:\n")
        for (weekday, frec) in weekday_frec:
            (frec_origen, frec_destino)=frec
            print(f"Dia {weekday} ----> Frec_desenganche: \
                  {frec_origen}, Frec_enganche: {frec_destino}")
            f.write(f"Dia {weekday} ----> Frec_desenganche: \
                    {frec_origen}, Frec_enganche: {frec_destino}\n")
        f.write("\n\n")
    f.close()
```

Calculamos el número total de llegadas (enganches) por día y estación, calculamos el número total de salidas (desenganches) por día y estación y los unimos los RDD de salidas y llegadas por día y estación con el método *join*. Finalmente aplicamos un *map* para calcular el promedio y agruparlos por su estación con un *groupByKey*.

Algunos de los resultados obtenidos son (solo para una estación):

```
Estacion 154:
Día 2 ----> Frec_desenganche: 59.05555555555556, Frec_enganche: 56.80555555555556
Día 4 ----> Frec_desenganche: 56.97222222222222, Frec_enganche: 54.25
Día 0 ----> Frec_desenganche: 57.80555555555556, Frec_enganche: 53.833333333333336
Día 6 ----> Frec_desenganche: 47.80555555555556, Frec_enganche: 44.72222222222222
Día 5 ----> Frec_desenganche: 40.138888888888886, Frec_enganche: 38.888888888888886
Día 1 ----> Frec_desenganche: 59.19444444444444, Frec_enganche: 53.94444444444444
Día 3 ----> Frec_desenganche: 61.638888888888886, Frec_enganche: 58.083333333333336
```

Este estudio puede ser útil para hacer una previsión sobre el tráfico que va a haber en una estación en un día concreto. Y así decidir en cuáles es necesario aumentar el número de bicicletas de préstamo disponibles.

A. Código Análisis.

```
from pyspark import SparkContext, SparkConf
import json
from datetime import datetime
import sys

conf = SparkConf().setAppName("Bicimad Estudios")
sc = SparkContext(conf=conf)
sc.setLogLevel("ERROR")

def datos(line):

    data = json.loads(line)
    origen = data['idunplug_station']
    destino=data['idplug_station']
    fecha = datetime.strptime(data['unplug_hourTime']['$date']\
                              , "%Y-%m-%dT%H:%M:%S.%f%z")
    mes=fecha.month
    weekday=fecha.weekday()
    tiempo = data['travel_time']
    edad = data['ageRange']
    codigo_usuario = data['user_day_code']
    return origen, destino, weekday, edad, tiempo, codigo_usuario, mes

def main(sc, years, months, datadir):

     rdd = sc.parallelize([])
     for y in years:
         for m in months:
             filename = f"{datadir}/{y}{m:02}_movements.json"
             print(f"Adding {filename}")
             rdd = rdd.union(sc.textFile(filename))

     n_semanas = len(years)*len(months)*4
     n_years = len(years)
     rdd_base=rdd.map(datos)

     #Estudio media usuarios unicos por dia de la semana y mes:
     usuarios_unicos_dias(rdd_base,n_semanas)

     usuarios_unicos_meses(rdd_base,n_years)

     #Estudio frecuencias:
     estudio_frecuencias_dias(rdd_base, n_semanas)

     estudio_frecuencias_estacion(rdd_base, n_semanas)

     #Estudio tiempo medio:
     tiempo_medio_viaje_por_edad(rdd_base)
```

```
    tiempo_medio_por_dia_semana(rdd_base)

    tiempo_medio_por_edad(rdd_base)

    #Estudio de viajes:
    viajes_mismo_origen_destino_por_estacion(rdd_base)

def usuarios_unicos_dias(rdd_base,n_semanas):

    rdd_usuarios_dias = rdd_base.map(lambda x: (x[2],x[5]))\
        .distinct()\
        .map(lambda x: (x[0],1)).reduceByKey(lambda a, b:a+b)\
        .map(lambda x: (x[0],x[1]/n_semanas))
    lista_usuarios_dias=rdd_usuarios_dias.collect()

    f = open('usuarios_unicos_por_dia.txt', 'w')
    for (weekday, media) in lista_usuarios_dias:
        print(f'Dia de la semana {weekday} ---->\
              Media de usuarios unicos {media}')
        f.write(f'Dia de la semana {weekday} ---->\
                Media de usuarios unicos {media}\n')
    f.close()

def usuarios_unicos_meses(rdd_base,n_years):

    rdd_usuarios_meses = rdd_base.map(lambda x: (x[6],x[5]))\
        .distinct()\
        .map(lambda x: (x[0],1)).reduceByKey(lambda a, b:a+b)\
        .map(lambda x: (x[0],x[1]/n_years))

    f = open('usuarios_unicos_por_mes.txt', 'w')
    for (mes, media) in rdd_usuarios_meses.collect():
        print(f'Mes {mes} ----> Media de usuarios unicos {media}')
        f.write(f'Mes {mes} ----> Media de usuarios unicos {media}\n')
    f.close()

def estudio_frecuencias_dias(rdd_base,n_semanas):

    rdd_frec_origen = rdd_base.map(lambda x: ((x[2],x[0]), 1))\
        .reduceByKey(lambda a, b: a + b)

    rdd_orden_origen = rdd_frec_origen\
        .map(lambda x: (x[0][0], (x[0][1], x[1]/n_semanas)))\
        .groupByKey()\
        .mapValues(lambda x: sorted\
                   (list(x), key=lambda y: y[1], reverse=True))

    f = open('desenganches_por_dia_por_estacion.txt', 'w')
    for (weekday, origen_frec) in rdd_orden_origen.collect():
        print(f"Dia: {weekday}")
        f.write(f"Dia: {weekday}\n")
```

```
            for (origen, frec) in origen_frec:
                print(f"Estacion de desenganche: {origen} --->\
                      Frecuencia: {frec}")
                f.write(f"Estacion de desenganche: {origen} --->\
                        Frecuencia: {frec}\n")
            print()
            f.write('\n\n')
        f.close()

        rdd_frec_destino = rdd_base.map(lambda x: ((x[2],x[1]), 1))\
            .reduceByKey(lambda a, b: a + b)

        rdd_orden_destino = rdd_frec_destino\
            .map(lambda x: (x[0][0], (x[0][1], x[1]/n_semanas)))\
            .groupByKey()\
            .mapValues(lambda x: sorted\
                       (list(x), key=lambda y: y[1], reverse=True))

        f = open('enganches_por_dia_por_estacion.txt', 'w')
        for (weekday, destino_frec) in rdd_orden_destino.collect():
            print(f"Dia: {weekday}")
            f.write(f"Dia: {weekday}\n")
            for (destino, frec) in destino_frec:
                print(f"Estacion de enganche: {destino} --->\
                      Frecuencia: {frec}")
                f.write(f"Estacion de enganche: {destino} ---\
                        > Frecuencia: {frec}\n")
            print()
            f.write('\n\n')
        f.close()

    def estudio_frecuencias_estacion(rdd_base, n_semanas):

        rdd_llegadas = rdd_base.map(lambda x: ((x[2], x[1]), 1))\
            .reduceByKey(lambda a, b: a + b)

        rdd_salidas = rdd_base.map(lambda x: ((x[2], x[0]), 1))\
            .reduceByKey(lambda a, b: a + b)

        rdd_resultado = rdd_salidas.join(rdd_llegadas)

        rdd_frec_total = rdd_resultado\
            .map(lambda x: (x[0][1], (x[0][0],\
                            (x[1][0]/n_semanas, x[1][1]/n_semanas))))\
            .groupByKey()

        f = open("enganches_desenganches_por_estacion_por_dia.txt", 'w')
        for (station, weekday_frec) in rdd_frec_total.collect():
            print(f"Estacion {station}:")
            f.write(f"Estacion {station}:\n")
            for (weekday, frec) in weekday_frec:
```

```
            (frec_origen, frec_destino)=frec
            print(f"Dia {weekday} ----> Frec_desenganche: \
                  {frec_origen}, Frec_enganche: {frec_destino}")
            f.write(f"Dia {weekday} ----> Frec_desenganche: \
                    {frec_origen}, Frec_enganche: {frec_destino}\n")
        f.write("\n\n")
    f.close()


def tiempo_medio_viaje_por_edad(rdd_base):

    rdd_tiempo_medio_viaje_por_edad = rdd_base\
        .map(lambda x: ((x[0], x[1], x[3]), x[4]))\
        .filter(lambda x: x[0][2]>0)\
        .mapValues(lambda x: (x,1))\
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))\
        .mapValues(lambda x: x[0]/x[1])\
        .map( lambda x: ( ((x[0][0],x[0][1]), (x[0][2], x[1]))))\
        .groupByKey()\
        .mapValues(lambda x: sorted(list(x), key=lambda y: y[0]))

    f = open("tiempo_medio_viaje_por_edad.txt", "w")
    for (viaje, edad_tiempo) in rdd_tiempo_medio_viaje_por_edad\
        .collect():
        print(f'Origen {viaje[0]} - Destino {viaje[1]}')
        f.write(f'Origen {viaje[0]} - Destino {viaje[1]}\n')
        for (edad, tiempo) in edad_tiempo:
            print(f'Grupo de edad {edad} ----> \
                  Tiempo promedio: {tiempo}')
            f.write(f'Grupo de edad {edad} ---->\
                    Tiempo promedio: {tiempo}\n')
        f.write('\n\n')
        print()
    f.close()

def tiempo_medio_por_dia_semana(rdd_base):

    rdd_tiempo_medio_por_dia = rdd_base\
        .map(lambda x: (x[2], (x[4], 1)))\
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))\
        .mapValues(lambda x: x[0]/x[1])

    f = open("tiempo_medio_por_dia_semana.txt", "w")
    for (dia, tiempo) in \
    sorted(list(rdd_tiempo_medio_por_dia.collect()), key=lambda y: y[0]):
        print(f'Dia {dia} ----> Tiempo promedio: {tiempo}')
        f.write(f'Dia {dia} ----> Tiempo promedio: {tiempo}\n')
    print()
    f.close()

def tiempo_medio_por_edad(rdd_base):
```

```
    rdd_tiempo_medio_por_edad = rdd_base\
        .map(lambda x: (x[3], (x[4], 1)))\
        .filter(lambda x: x[0] > 0)\
        .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))\
        .mapValues(lambda x: x[0]/x[1])

    f = open("tiempo_medio_por_edad.txt", "w")
    for (edad, tiempo) in \
    sorted(list(rdd_tiempo_medio_por_edad.collect()), key=lambda y: y[0]):
        print(f'Grupo de edad {edad} ---->\
              Tiempo promedio: {tiempo}')
        f.write(f'Grupo de edad {edad} ---->\
                Tiempo promedio: {tiempo}\n')
    print()
    f.close()

def viajes_mismo_origen_destino_por_estacion(rdd_base):

    rdd_viajes_mismo_origen_destino = rdd_base\
        .map(lambda x: (x[0], x[1]))\
        .filter(lambda x: x[0] == x[1])\
        .map(lambda x: (x[0], 1))\
        .reduceByKey(lambda x, y: x + y)

    f = open("viajes_mismo_origen_destino_por_estacion.txt", "w")
    for (estacion, n_viajes) in \
    sorted(list(rdd_viajes_mismo_origen_destino.collect())\
           , key=lambda y: y[1], reverse=True):
        print(f'Numero de estacion {estacion} ---->\
            Numero de viajes con mismo origen y destino: {n_viajes}')
        f.write(f'Numero de estacion {estacion} ---->\
            N mero de viajes con mismo origen y destino: {n_viajes}\n')
    print()
    f.close()


if __name__=="__main__":

    if len(sys.argv)<2:
        years = [2021]
    else:
        years = list(map(int, sys.argv[1].split()))

    if len(sys.argv)<3:
        months = [3]
    else:
        months = list(map(int, sys.argv[2].split()))

    if len(sys.argv)<4:
        datadir = "."
```

```
else:
    datadir = sys.argv[3]

print(f"years: {years}")
print(f"months: {months}")
print(f"datadir: {datadir}")

main(sc, years, months, datadir)
```

B. Código Gráficas.

```
import matplotlib.pyplot as plt

def obtener_grafica(namefile, pos, title, xlabel, ylabel):
    datos = []
    f = open(namefile, 'r')
    for linea in f:
        print(linea)
        linea_split = linea.split()
        if linea_split != []:
            datos\
            .append((float(linea_split[pos]), float(linea_split[-1])))
    datos = sorted(datos, key=lambda x: x[0])
    x, y = unzip(datos)

    plt.plot(x, y)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.show

def unzip(datos):
    x = []
    y = []
    for d in datos:
        x.append(d[0])
        y.append(d[1])
    return x, y
```