

Memoria Práctica BICIMAD - Programación Paralela



1. Definición del problema planteado

Una empresa del sector del transporte está actualmente ofertando un servicio de bicicletas de uso compartido. Su objetivo es maximizar sus beneficios, para ello se está planteando una reorganización de los puntos de estacionamiento de sus bicicletas, y además, una nueva campaña de marketing para atraer a antiguos usuarios del servicio de bicicletas de uso compartido, ya que las tendencias de este servicio cambiaron después de la pandemia de COVID-19. La empresa pretende que la campaña empiece a funcionar en marzo del próximo año, y dure toda la primavera, acabada la cual se espera que los niveles de popularidad de la marca ya sean más altos.

2. Idea de la solución del problema

Como la empresa que quiere realizar la campaña no se fundó hasta 2021 y necesita datos pre-pandemia, vamos a utilizar los de la empresa de BiciMAD (datos que son públicos). Resolveremos el problema usando pyspark, obteniendo cuál es el perfil de un usuario medio, y cuáles son las estaciones de BiciMAD más concurridas. Además, en caso de que la empresa quiera acceder a los datos de una sola zona, por tener esta más nivel adquisitivo de media, se podrá hacer uso de un filtro opcional de zona, además del filtro opcional de estación del año (invierno, primavera, otoño, verano), obteniendo así únicamente los datos de la estación que se indique.

3. Diseño de la solución

En primer lugar traducimos los archivos *json* mediante la función *info_line*, complementada con la *info_date* para descifrar las fechas, ya que éstas vienen en un formato poco amable, difícil de tratar. Declaramos algunas variables globales que el usuario puede cambiar si lo desea, en función de si quiere obtener los datos de una época del año o de un distrito de Madrid, y así sacar conclusiones en función de esos datos. En caso de ser utilizadas, se aplicarán los filtros *filter_distrito* y *filter_estacion* al RDD antes de proceder al análisis.

A continuación, nos enfocamos en ver las estaciones de bicicletas más concurridas según el día de

la semana, la edad de los usuarios y el tipo de usuario. Para ello utilizamos las funciones *estacion.mas salen.por.dia*, *estacion.mas llegan.por.dia*, *estacion.mas salen.por.tipo*, etc. Todas se basan en el mismo esquema, el paradigma de programación *MapReduce*, nuestra implementación consiste en aplicar la función *filter* junto con la función *map* y la función *countByKey()* de Python, para posteriormente comparar los elementos del RDD resultante para ver cual es el máximo.

Una vez hemos hecho esto, vemos cuál es el uso de bicicletas para cada rango de edad y por cada tipo de usuario. Para ello observamos el número de veces que cada usuario utiliza el servicio de bicicletas y cuál es su tiempo medio de uso. Esto es relativamente sencillo usando las funciones *countByKey()* y *groupByKey()*. Consideramos que el tiempo medio de uso puede ser relevante puesto que cuanto más tiempo se usa una bicicleta, la empresa obtiene un mayor beneficio.

En caso de que la empresa tenga interés en realizar un análisis anual, hemos implementado una función adicional que devuelve el número de viajes por mes.

4. Análisis de resultados

Para visualizar las estaciones de parada de la empresa BiciMAD tenemos los siguientes planos:

- [Plano de las estaciones de BiciMAD](#)
- [Plano detallado BiciMAD](#)

Hemos analizado los datos de 2018 y 2017 completos y los del mes de marzo de 2019. Usaremos los datos de marzo para extraer la mayoría de nuestras conclusiones, puesto que es el mes en el que nuestro cliente quiere empezar su campaña, y nos fijaremos en los de 2018 y 2017 para ratificar las tendencias que hemos observado. Además hemos añadido algunas gráficas para visualizar mejor los resultados en las que hemos tenido en cuenta diferentes grupos de edad y una clasificación de los usuarios.

Los grupos de edad son:

- Grupo 0: Indeterminado
- Grupo 1: Niños y adolescentes (0-16 años)
- Grupo 2: Jóvenes sin la mayoría de edad (17-18 años)
- Grupo 3: Jóvenes con mayoría de edad (19-26 años)
- Grupo 4: Adultos de edad menos avanzada (27-40 años)
- Grupo 5: Adultos de edad más avanzada (41-65 años)
- Grupo 6: Jubilados (a partir de los 66 años)

Tipos de usuarios:

- Tipo I: Abonados (tienen un abono anual)
- Tipo II: Usuarios ocasionales (utilizan el servicio de bicicletas esporádicamente)
- Tipo III: Trabajadores de la empresa

Algunos datos relevantes que hemos obtenido son:

4.1. Marzo 2019 (todos los distritos)

Las estaciones más concurridas por día de la semana son Lavapiés (43), Jaime el Conquistador (175) y Paseo de la Esperanza (163), teniendo esta última predominancia en los días más dedicados al ocio. También se observa un considerable aumento del uso de las bicicletas los domingos (de unas 700 en cada una de estas estaciones a alrededor de 1000). Estos podrían ser buenos sitios para poner nuestros anuncios.

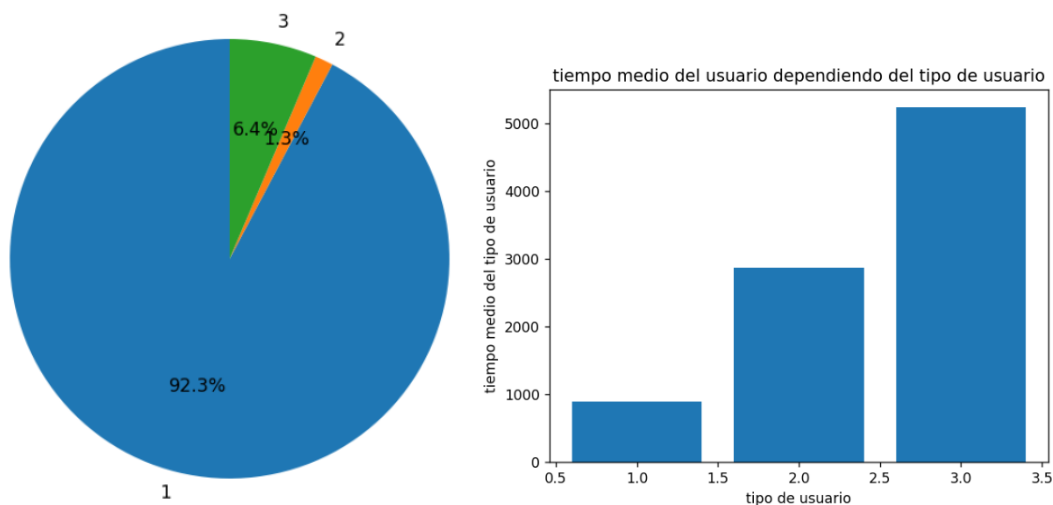


Figura 1: Porcentaje de viajes por tipo de usuario y tiempo medio del usuario dependiendo del tipo de usuario (ocasional, trabajador de la empresa ya bonado, respectivamente)

Se observa en la Figura 1 que la gran mayoría de los usuarios gozan de una suscripción anual, y que solo un pequeño porcentaje hace viajes casuales o de empresa, pero que el tiempo medio de uso de estos usuarios es el menor. Esto, sin embargo, no es de gran relevancia debido al elevado número de clientes que están suscritos (casi el 93 %). Esto ha de ser tenido en cuenta a la hora de poner el anuncio. La estación más concurrida en total por estos usuarios es Paseo de la Esperanza (163).

La fiabilidad de nuestro análisis sobre los rangos de edad de los usuarios puede ser puesta en tela de juicio, puesto que en el 43 % de los casos no se conoce la edad del usuario. De todas maneras, en la Figura 2 podemos observar que dentro del 57 % restante, los grupos que más usan las bicicletas son el 4 y el 5 con amplia diferencia, esto es, personas entre 27 y 40 años (28 %) y entre 40 y 63 (21 %). Así, nuestro anuncio probablemente deberá estar enfocado a gente bien entrada en la etapa adulta de su vida. Como es de esperar, los más jóvenes son los que, cuando cogen la bicicleta, viajes más largos hacen. La estación más concurrida por el grupo 4 vuelve a ser la 163, y las más concurridas por el grupo 5 son la 64 (Antonio Maura) y la 90 (Goya). Los jóvenes frecuentan mucho la estación 75 (Menéndez Pelayo).

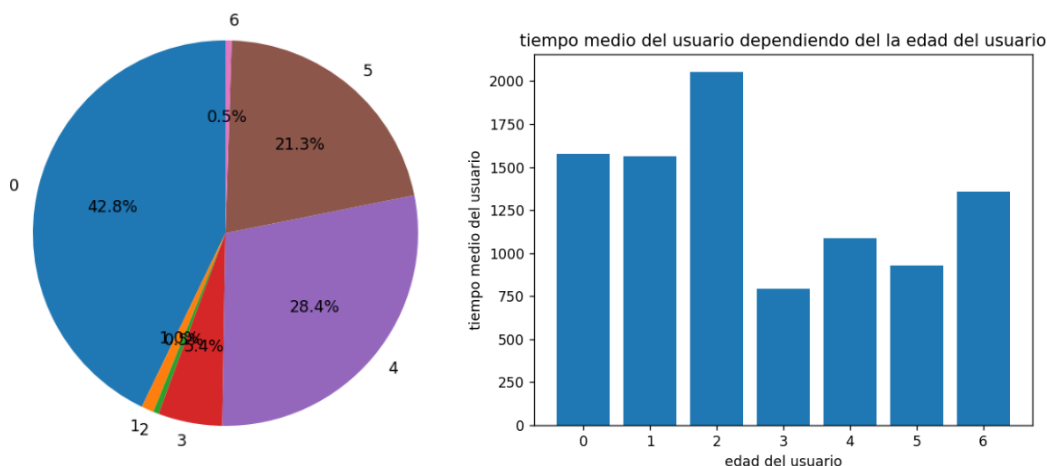


Figura 2: Porcentaje de uso de las bicicletas según el grupo de edad y tiempo medio del usuario dependiendo de la edad del usuario.

4.2. Marzo de 2019 (distrito Retiro)

Hemos decidido aplicar el filtro, en nuestro caso, a Retiro, una zona amplia y foco de turistas, pero se podría aplicar a cualquier distrito indicando en el código implementado, los números de estación de ese distrito. En este distrito las estaciones más concurridas son la 64 (Antonio Maura), la 90 (Goya) y la 83 (Pío Baroja). Cabe destacar que los viajes que se inician en este distrito, por lo que parece, no suelen, en su mayoría, salir del distrito. Los datos de edad y tipo de usuario siguen la línea de los globales de Madrid, con los grupos 4 y 5 de edad dominando, y siendo la inmensa mayoría de viajes hechos por usuarios con suscripción anual.

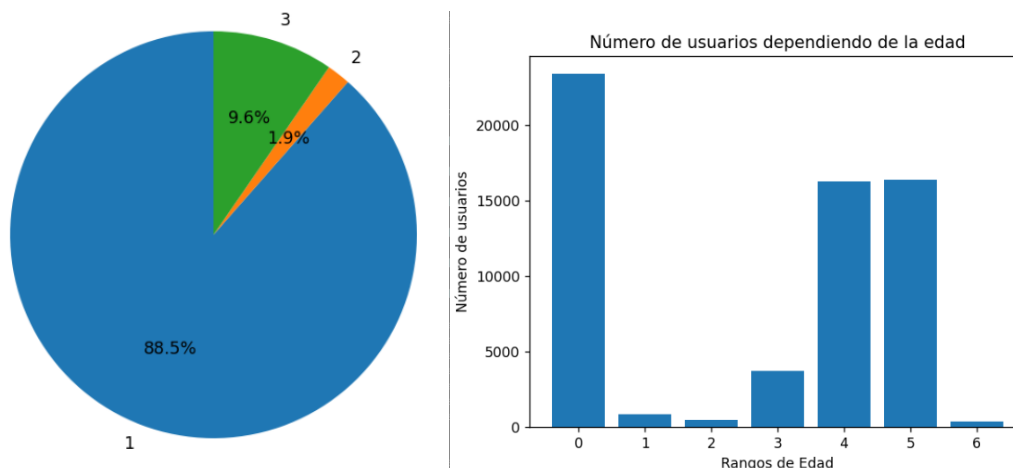


Figura 3: Porcentaje de viajes por tipo de usuario y número de usuarios dependiendo de la edad (distrito Retiro).

4.3. Año 2018 (distrito Retiro)

Las conclusiones que extrajimos de los datos de marzo de 2019 en cuanto al perfil de los usuarios quedan confirmadas en este análisis anual, que da resultados muy parecidos. En cuanto a las estaciones más concurridas, como es de esperar los datos también son similares, aunque aparecen algunas estaciones nuevas a tener en cuenta como la 76 (Avda Mediterráneo).

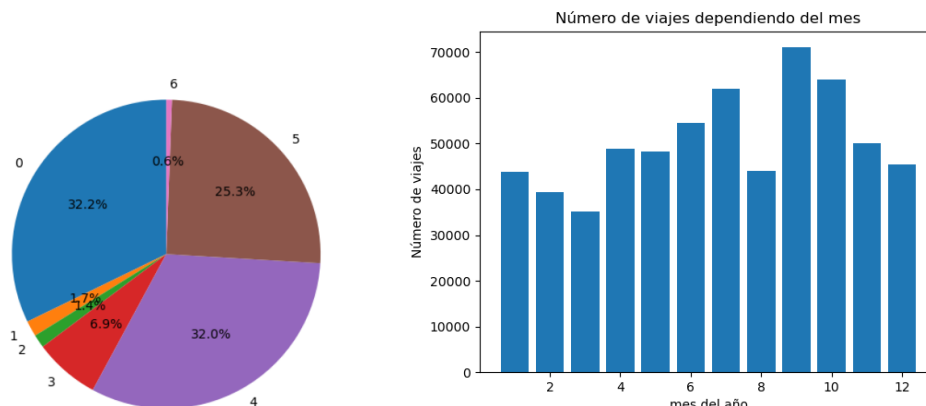


Figura 4: Porcentaje de tiempo medio de viaje según el rango de edad y Número de viajes realizados cada mes (distrito Retiro).

Los datos realmente interesantes de este año, al menos potencialmente, para nuestra empresa, son los viajes por mes, que incrementan sustancialmente en otoño, como podemos observar en la Figura 4.

4.4. Año 2017 (distrito Retiro)

Se observa que el número de viajes en este año es algo menor que en 2018, lo cual es una muy buena señal, pues significa que el uso de las bicicletas tenía una tendencia ascendente antes de la pandemia. Además, se observa un aumento en 2018 respecto de este año en la cantidad de abonados anuales, lo que refuerza nuestra construcción del perfil de usuario.

5. Conclusión

Una estrategia de marketing eficaz consistiría en colocar anuncios publicitarios cerca de las estaciones 175 y 163 y reforzar el número de bicicletas allí, ya que dichas estaciones tenían una fuerte demanda de este servicio antes de la pandemia, y así poder recuperar antiguos clientes. Esos anuncios deberían tener en cuenta que los usuarios más comunes suelen ser personas de mediana edad, siendo la suscripción anual, la opción más atractiva para este tipo de usuarios, para ello la empresa podría ofrecer algún tipo de oferta o plan de financiación para la suscripción anual.

Además, cabe destacar que la duración media de viaje de los jóvenes es significativamente más elevada que para otros grupos, especialmente con más afluencia en la zona circundante a la estación 75. Los anuncios en dicha estación podrían enfocarse en este tipo de clientes.

Finalmente debe tener en cuenta que en primavera, y especialmente en marzo, el uso de las bicicletas suele ser menor, así que se podría plantear la posibilidad de pasar la campaña a septiembre, el mes con más demanda antes de la pandemia del servicio de bicicletas, y prolongarla a lo largo del otoño.

6. Anexo con la implementación del programa en Python

```
1 import json
2 from pprint import pprint
3 from pyspark import SparkContext, SparkConf
4 import datetime
5 import sys
6 import os
7 import matplotlib.pyplot as plt
8
9
10 # Set station, district, and filters (also can be done with sys.argv)
11 YEAR = 2017
12 STATION = 'Spring'
13 DISTRICT = [i for i in range(64,92)] # Retiro district
14 WANT_TO_FILTER_DISTRICT = True
15 WANT_TO_FILTER_STATION = False
16 YEARLY_ANALYSIS = False
17
18 # Function to extract information from each line
19 def line_info(line):
20     data = json.loads(line)
21     ageRange1 = data['ageRange']
22     id1 = data['user_day_code']
23     start = data['idunplug_station']
24     end = data['idplug_station']
25     day = data['unplug_hourTime']['$date']
26     duration = data['travel_time']
27     user_type = data['user_type']
28     return (ageRange1, id1, start, end, date_converter(day), duration, user_type)
29
30 # Date conversion
31 def date_converter(date):
32     """Converts the date received from the BiciMad files.
33     Returns a tuple with the date in tuple format.
34     The tuple contains the weekday, the day of the month, the month, and the year."""
35
36     month = int(date[5:7])
```

```

37     day = int(date[8:10])
38     week_day = datetime.datetime(YEAR, month, day).weekday()
39     return (week_day, day, month, YEAR)
40
41 def day_converter(week_day):
42     """Given a weekday from the datetime module, i.e., a number from 0 to 6,
43     converts the number to human-readable format."""
44
45     days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', '
46     Sunday']
47     return days[week_day]
48
49 def season(date_tuple):
50     """Given a tuple returned by date_converter, it returns the
51     season for that date."""
52
53     (name, day, month, year) = date_tuple
54     if month in (4, 5) or (month == 3 and day >=20) or (month == 6 and day <=20):
55         return 'Spring'
56     elif month in (7, 8) or (month == 6 and day >=21) or (month == 9 and day <=21):
57         return 'Summer'
58     elif month in (10, 11) or (month == 9 and day >=22) or (month == 12 and day <=20):
59         return 'Fall'
60     else:
61         return 'Winter'
62
63 # Functions related to filtering the information prior to analysis
64
65 def filter_by_season():
66     return WANT_TO_FILTER_STATION
67
68 def filter_by_district():
69     return WANT_TO_FILTER_DISTRICT
70
71 def is_in_district(x):
72     return x in DISTRICT
73
74 def filter_district(rdd):
75     return rdd.filter(lambda x: is_in_district(int(x[2])))
76
77 def filter_season(rdd):
78     return rdd.filter(lambda x : season(x[4]) == STATION)
79
80 # Auxiliary filters for the main functions
81
82 # Age filters
83
84 def filter_by_age_start(rdd, age):
85     """Given an age, returns an RDD where only the trips made by users of that
86     age are included, with the start as key. Other functions in this section
87     are analogous."""
88
89     return rdd.filter(lambda x : x[0] == age).map(lambda x: (x[2], (x[1], x[3], x[4])
90     ))
91
92 def filter_by_age_end(rdd, age):
93     # Returns the end as key for records of a given age
94     rdd_age = rdd.filter(lambda x : x[0] == age).map(lambda x: (x[3], (x[1], x[2], x
95     [4])))
96     return rdd_age
97
98 # Functions to filter by user type
99
100 def filter_type_start(rdd, type):
101     rdd_type = rdd.filter(lambda x : x[6] == type).map(lambda x: (x[2], (x[1], x[3], x
102     [4])))
103     return rdd_type

```

```

103
104 # Functions to filter by day of the week
105 def filter_day_start(rdd, day):
106     rdd_day = rdd.filter(lambda x : x[4][0] == day).map(lambda x: (x[2], (x[1], x[3],
107         x[4])))
108     return rdd_day
109
110 def filter_day_end(rdd, day):
111     rdd_day = rdd.filter(lambda x : x[4][0] == day).map(lambda x: (x[3], (x[1], x[2],
112         x[4])))
113     return rdd_day
114
115 # Function to return the station with the most departures for each day of the week
116 def spot_more_starts_per_day(rdd):
117     rdd_day = [0] * 7
118     for i in range(7):
119         rdd_day[i] = filter_day_end(rdd, i).countByKey()
120         max_station = max(rdd_day[i], key = rdd_day[i].get)
121         print(f'The station with the most departures on {day_converter(i)} is {
122             max_station} with {rdd_day[i][max_station]} departures.')
123
124 # Function to return the station with the most arrivals for each day of the week
125 def spot_more_ends_per_day(rdd):
126     rdd_day = [0] * 7
127     for i in range(7):
128         rdd_day[i] = filter_day_start(rdd, i).countByKey()
129         max_station = max(rdd_day[i], key = rdd_day[i].get)
130         print(f'The station with the most arrivals on {day_converter(i)} is {
131             max_station} with {rdd_day[i][max_station]} arrivals.')
132
133 # Function to return the station with the most departures for each user type
134 def spot_more_starts_per_type(rdd):
135     rdd_type = [0] * 4
136     for i in range(4):
137         rdd_type[i] = filter_type_start(rdd, i).countByKey()
138         if len(list(rdd_type[i])) > 1:
139             max_station = max(rdd_type[i], key = rdd_type[i].get)
140             print(f'The station with the most departures for user type {i} is {
141                 max_station} with {rdd_type[i][max_station]} departures.')
142         else:
143             print(f'No departures for user type {i}.')
144
145 # Function to return the station with the most arrivals for each user type
146 def spot_more_ends_per_type(rdd):
147     rdd_type = [0] * 4
148     for i in range(4):
149         rdd_type[i] = filter_type_end(rdd, i).countByKey()
150         if len(list(rdd_type[i])) > 1:
151             max_est = max(rdd_type[i], key = rdd_type[i].get)
152             print(f'La estaci n en la que llegan m s bicicletas para el tipo de
153                 usuario {i} es {max_est} y han salido {rdd_type[i][max_est]} bicis')
154         else:
155             print(f'para el tipo de usuario {i} no llegan bicicletas')
156
157 # Similar to the previous but for each age range
158 def spot_more_starts_per_age(rdd):
159     rdd_age = [0] * 7
160     for i in range(7):
161         rdd_age[i] = filter_by_age_start(rdd, i).countByKey()
162         if len(list(rdd_age[i])) > 1:
163             max_est = max(rdd_age[i], key = rdd_age[i].get)
164             print(f'The station from which most bikes leave for the age group {i} is {
165                 max_est} and {rdd_age[i][max_est]} bikes have left')
166         else:
167             print(f'for the age range {i}, no bikes leave')
168
169 def spot_more_ends_per_age(rdd):
170     rdd_age = [0] * 7
171     for i in range(7):
172         rdd_age[i] = filter_by_age_end(rdd, i).countByKey()

```

```

167         if len(list(rdd_age[i]))>1:
168             max_est = max(rdd_age[i], key = rdd_age[i].get)
169             print(f'The station where most bikes arrive for the age group {i} is {
max_est} and {rdd_age[i][max_est]} bikes have arrived')
170         else:
171             print(f'for the age range {i}, no bikes arrive')
172
173 # Function that prints how many trips are made by age
174 def trips_per_age(rdd):
175     rdd_counted_ages = rdd.countByKey()
176     print('The age groups and the bicycles they use are:', '\n')
177     for i in rdd_counted_ages:
178         print(f'The age group {i} used {rdd_counted_ages[i]} bicycles')
179
180     # BAR CHART
181     axes = [[0,1,2,3,4,5,6],[rdd_counted_ages[i] for i in range(7)]]
182     plt.bar(axes[0],axes[1])
183     plt.ylabel('Number of users')
184     plt.xlabel('Age Ranges')
185     plt.title('Number of users depending on age')
186     plt.savefig('tripsagebar',format='png')
187
188     # PIE CHART
189     labels = axes[0]
190     sizes = axes[1]
191     fig1, ax1 = plt.subplots()
192     ax1.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
193     ax1.axis('equal')
194     plt.savefig('tripsagepie',format='png')
195
196 # Function that prints the average time of each trip by age.
197 def time_per_age(rdd_base):
198     rdd_duration= rdd_base.map(lambda x:(x[0],x[5])).groupByKey().map(lambda x : (sum(
list(x[1]))/len(list(x[1])))).collect()
199     print('The average time that bicycles are used according to the age range is:')
200     for i in range(len(list(rdd_duration))):
201         print(f'The age group {i} used bicycles for an average of {rdd_duration[i]/60}
minutes')
202
203     # BAR CHART
204     axes = [[0,1,2,3,4,5,6],list(rdd_duration)]
205     plt.bar(axes[0],axes[1])
206     plt.ylabel('average user time')
207     plt.xlabel('user age')
208     plt.title('average user time depending on the user age')
209     plt.savefig('timeage',format='png')
210
211 # Function that prints how many trips are made by user type
212 def trips_per_type(rdd):
213     rdd_counted_type = rdd.map(lambda x: (x[6],(x[0:5]))).countByKey()
214     print('The types of users and the bicycles they use are:', '\n')
215     for i in rdd_counted_type:
216         print(f'the user type {i} used {rdd_counted_type[i]} bicycles')
217
218     # BAR CHART
219     axes = [[1,2,3],[rdd_counted_type[i] for i in range(1,4)]]
220     plt.bar(axes[0],axes[1])
221     plt.ylabel('Number of users')
222     plt.xlabel('types of users')
223     plt.title('Number of users depending on user type')
224     plt.savefig('tripstypesbar',format='png')
225
226     # PIE CHART
227     labels = axes[0]
228     sizes = axes[1]
229     fig1, ax1 = plt.subplots()
230     ax1.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
231     ax1.axis('equal')
232     plt.savefig('tripstypepie',format='png')
233
234 # Function that prints the average time of each trip by user type

```



```

235 def time_per_type(rdd_base):
236     rdd_duration= rdd_base.map(lambda x:(x[6],x[5])).groupByKey().map(lambda x : (sum(
237         list(x[1]))/len(list(x[1])))).collect()
238     print('The average time that bicycles are used according to the user type is:')
239     for i in range(len(list(rdd_duration))):
240         print(f'The user type {i+1} used bicycles for an average of {rdd_duration[i]
241             /60} minutes')
242
243     # BAR CHART
244     axes = [[1,2,3],[rdd_duration[i] for i in range(3)]]
245     plt.bar(axes[0],axes[1])
246     plt.ylabel('average time of the user type')
247     plt.xlabel('user type')
248     plt.title('average user time depending on the user type')
249     plt.savefig('durationtype',format='png')
250
251 # Function that prints the number of trips made per month, in case of annual analysis
252 def trips_per_month(rdd):
253     rdd_trips=rdd.map(lambda x: (x[4][2], (x[1], x[3], x[4]))).countByKey()
254     for i in range(1,13):
255         print(f'in the month {i} of the year, {rdd_trips[i]} bicycles were used')
256     axes = [[i for i in range(1, 13)],[rdd_trips[i] for i in range(1, 13)]]
257     plt.bar(axes[0],axes[1])
258     plt.ylabel('Number of trips')
259     plt.xlabel('Month of the year')
260     plt.title('Number of trips depending on the month')
261     plt.savefig('tripsyear',format='png')
262
263 def main():
264     # Initialize Spark and load the JSON files
265     conf = SparkConf().setAppName("Routes")
266
267     # Create SparkContext with our configuration
268     with SparkContext(conf = conf) as sc:
269         sc.setLogLevel("ERROR") # Set log level to 'ERROR' to avoid cluttering output
270         with log messages
271
272         directory = os.path.abspath('bicimad_data') # Get the absolute path of the
273         data directory
274         rdd_base = sc.emptyRDD() # Initialize an empty RDD
275         i=0 # Counter for files processed
276         # Iterate over each file in the directory
277         for filename in os.listdir(directory):
278             i+=1 # Increment the file counter
279             # Process only JSON files
280             if filename.endswith(".json"):
281                 print(f"Processing file: {filename}") # Inform about the file being
282                 processed
283                 file_rdd = sc.textFile(os.path.join(directory, filename)) # Load the
284                 JSON file into an RDD
285                 rdd_base = rdd_base.union(file_rdd) # Merge the loaded RDD with the
286                 base RDD
287             else:
288                 pass # If the file is not a JSON file, ignore it
289
290         # If we processed 12 files, enable yearly analysis
291         if i==12:
292             YEARLY_ANALYSIS=True
293
294         # Transform the loaded data
295         rdd = rdd_base.map(line_info)
296
297         # Apply the district filter, if requested
298         if filter_by_district():
299             rdd = filter_district(rdd) # Apply the district filter
300             print('\n', 'Analyzing district data', '\n')
301         else:
302             print('\n', 'Analyzing Madrid data', '\n') # Inform about the full data
303             analysis

```

```

298
299     # Apply the season filter, if requested
300     if filter_by_season():
301         print('The data will be filtered by season')
302         rdd = filter_season(rdd) # Apply the season filter
303     else:
304         print('\n', 'The data will not be filtered by season') # Inform about not
        applying season filter
305
306     # Start the analysis
307     print('\n', 'Statistics of bike usage by day', '\n')
308     spot_more_starts_per_day(rdd) # Determine the spots with the most starts per
day
309     print('\n')
310     spot_more_ends_per_day(rdd) # Determine the spots with the most ends per day
311
312     print('\n', 'Statistics of bike usage by user type', '\n')
313     print('The types of users are:', '\n')
314     print('0: User type could not be determined', '\n')
315     print('1: Annual user (possessor of an annual pass)', '\n')
316     print('2: Occasional user', '\n')
317     print('3: Company worker', '\n')
318
319     spot_more_starts_per_type(rdd) # Determine the spots with the most starts per
user type
320     print('\n')
321     spot_more_ends_per_type(rdd) # Determine the spots with the most ends per
user type
322
323     print('\n', 'Statistics of bike usage by age groups', '\n')
324     print('The age groups are:', '\n')
325     print('0: User age group could not be determined', '\n')
326     print('1: User is between 0 and 16 years old', '\n')
327     print('2: User is between 17 and 18 years old', '\n')
328     print('3: User is between 19 and 26 years old', '\n')
329     print('4: User is between 27 and 40 years old', '\n')
330     print('5: User is between 41 and 65 years old', '\n')
331     print('6: User is 66 years old or more', '\n')
332
333     spot_more_starts_per_age(rdd) # Determine the spots with the most starts per
age group
334     print('\n')
335     spot_more_ends_per_age(rdd) # Determine the spots with the most ends per age
group
336
337     print('\n', 'Usage according to age', '\n')
338     trips_per_age(rdd) # Determine the number of trips per age group
339     time_per_age(rdd) # Determine the duration of trips per age group
340
341     print('\n', 'Statistics of bike usage by user type', '\n')
342     trips_per_type(rdd) # Determine the number of trips per user type
343     time_per_type(rdd) # Determine the duration of trips per user type
344
345     # If yearly analysis is enabled, perform it
346     if YEARLY_ANALYSIS:
347         print('Yearly user evolution')
348         trips_per_month(rdd) # Determine the number of trips per month
349     sc.stop() # Stop the SparkContext to free resources
350
351 if __name__ == "__main__":
352
353     # Interact with the user for changing initial filters, if arguments provided.
354     if len(sys.argv) > 2:
355         if isinstance(sys.argv[1], bool):
356             if sys.argv[1]:
357                 WANT_TO_FILTER_STATION=True
358             else:
359                 WANT_TO_FILTER_STATION=False
360
361         if isinstance(sys.argv[2], str):
362             STATION=sys.argv[2]

```

```
363
364     if isinstance(sys.argv[3], bool):
365         if sys.argv[3]:
366             WANT_TO_FILTER_DISTRICT=True
367         else:
368             WANT_TO_FILTER_DISTRICT=False
369
370     if isinstance(sys.argv[4], list):
371         DISTRICT=sys.argv[4]
372
373 main() # Call the main function
```