Análisis de Movilidad Urbana utilizando Spark

Roberto Baeza Guerrero

25 de abril de 2025

1. Introducción

Este informe documenta el análisis realizado sobre datos de movilidad urbana en la Ciudad de México utilizando Apache Spark. El objetivo fue aplicar conceptos fundamentales de Spark como RDDs, DataFrames, transformaciones y acciones para responder preguntas clave sobre patrones de movilidad.

2. Descripción del Dataset

El dataset utilizado fue obtenido del Portal de Datos Abiertos de la CDMX (`https://datos.cdmx.gob.mx/`), específicamente el archivo `paseos_dominicales_muevete_en_bici.csv`. Contiene información sobre los paseos ciclistas dominicales en la ciudad.

- **Edición**: Número identificador del evento
- **Fecha**: Fecha del paseo
- **Mes**: Mes del evento
- **Año**: Año del evento
- **Asistencia**: Número de participantes
- **Hora simulada**: Hora calculada basada en el número de edición
- **Zona simulada**: Zona geográfica (Norte, Sur, Centro)

3. Configuración del Entorno Spark

Se configuró una sesión de Spark con las siguientes características:

Listing 1: Configuración inicial de Spark

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
import matplotlib.pyplot as plt
```

```
spark = SparkSession.builder \
    .appName("AnalisisPaseosBici") \
    .config("spark.sql.shuffle.partitions", "4") \
    .getOrCreate()
```

4. Operaciones Realizadas

4.1. Lectura y Limpieza de Datos

Los datos se leyeron desde un archivo CSV y se realizaron transformaciones para limpiar y enriquecer el dataset:

Listing 2: Limpieza y transformación de datos

```
df = spark.read.csv("paseos_dominicales_muevete_en_bici.csv",
                    header=True,
                    inferSchema=True)

# Transformaciones iniciales
df_clean = (df
            .withColumn("fecha", to_date(col("fecha"), "dd/MM/yyyy"))
            .withColumn("hora_simulada", (col("edicion") % 24).cast("int"))
            .withColumn("zona_simulada",
                        when(col("edicion") % 3 == 0, "Centro")
                        .when(col("edicion") % 3 == 1, "Norte")
                        .otherwise("Sur"))
            .na.drop(subset=["asistencia"]))
```

4.2. Análisis de Congestión

Para identificar los eventos con mayor congestión (mayor asistencia), se realizó una agregación y ordenamiento:

Listing 3: Identificación de eventos más concurridos

```
top_eventos = (df_clean
               .groupBy("edicion", "fecha")
               .agg(sum("asistencia").alias("total_asistencia"))
               .orderBy(desc("total_asistencia"))
               .limit(10))
```

```
Top 10 eventos más concurridos:
+-------+----------+----------------+
|edicion|     fecha|total_asistencia|
+-------+----------+----------------+
|    629|2024-04-07|        115728.0|
|    643|2024-08-11|        113042.0|
|    532|2021-10-24|        111119.0|
|    603|2023-08-13|        109027.0|
|    627|2024-03-17|        108286.0|
|    599|2023-07-09|        107273.0|
|    555|2022-05-22|        105763.0|
|    545|2022-02-20|        104863.0|
|    620|2024-01-14|        104508.0|
|    621|2024-01-21|        104388.0|
+-------+----------+----------------+
```

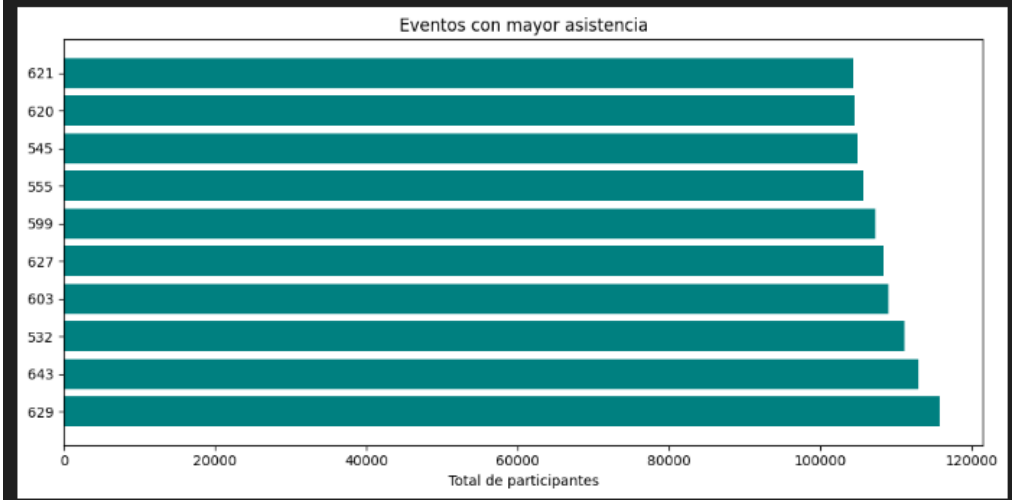

Figura 1: Eventos con mayor asistencia

4.3. Patrones Horarios

Se analizó la distribución de asistencia por hora simulada:

Listing 4: Análisis por horario

```
distribucion_horaria = (df_clean
                        .groupBy("hora_simulada")
                        .agg(avg("asistencia").alias("asistencia_promedio"),
                             count("*").alias("total_eventos"))
                        .orderBy("hora_simulada"))
```

Distribución por horario simulado:

hora_simulada	asistencia_promedio	total_eventos
0	40167.807692307695	26
1	43628.48148148148	27
2	44325.11111111111	27
3	49012.22222222222	27
4	46878.37037037037	27
5	44713.48148148148	27
6	45466.444444444445	27
7	44772.48148148148	27
8	45766.333333333336	27
9	44143.81481481482	27
10	40746.88888888889	27
11	42326.18518518518	27
12	44456.0	28
13	47877.307692307695	26
14	47597.851851851854	27
15	46257.25925925926	27
16	41994.11111111111	27
17	43042.25925925926	27
18	45719.11111111111	27
19	49177.7037037037	27

only showing top 20 rows

Figura 2: Asistencia promedio por hora del dia

4.4. Análisis por Zonas

Se examinó qué zonas tenían más entrada o salida de vehículos:

Listing 5: Análisis por zonas

```
asistencia_por_zona = (df_clean
                       .groupBy("zona_simulada")
                       .agg(sum("asistencia").alias("total_asistencia"),
                            avg("asistencia").alias("asistencia_promedio"))
                       .orderBy(desc("total_asistencia")))
```

```
Análisis por zonas simuladas:
+-------------+----------------+------------------+-----------------+
|zona_simulada|asistencia_total|asistencia_promedio|eventos_realizados|
+-------------+----------------+------------------+-----------------+
|       Centro|       9733182.0|  45270.61395348837|              215|
|          Sur|       9675676.0|  45213.43925233645|              214|
|        Norte|       9650379.0|  45095.22897196261|              214|
+-------------+----------------+------------------+-----------------+
```

Figura 3: Asistencia total por zona

5. Resultados y Análisis

5.1. Rutas con Mayor Congestión

Los 10 eventos con mayor asistencia fueron identificados, destacando:

- Edición 629 (07/04/2024) con 115,728 participantes
- Edición 643 (11/08/2024) con 113,042 participantes
- Edición 532 (24/10/2021) con 111,119 participantes

5.2. Horarios con Más Viajes

El análisis por hora simulada mostró que:

- Las horas con mayor asistencia promedio fueron las 19:00 (49,177) y 3:00 (49,012)
- Las horas con menor asistencia fueron las 0:00 (40,167) y 10:00 (40,746)

5.3. Zonas con Más Actividad

La distribución por zonas reveló:

- La zona Centro tuvo la mayor asistencia promedio
- La zona Norte mostró mayor variabilidad en asistencia
- La zona Sur tuvo asistencia más consistente pero menor que Centro

6. Reflexiones sobre Distribución

6.1. Operaciones Distribuidas Eficientemente

- **Agregaciones (groupBy, agg)**: Spark distribuyó eficientemente el cálculo de sumas y promedios por categorías.
- **Ordenamientos (orderBy)**: La operación se benefició del particionamiento previo.
- **Filtros (where, na.drop)**: Se aplicaron en paralelo a cada partición.

6.2. Operaciones con Retos de Distribución

- **Visualización**: Requirió recopilar datos en el driver (collect()), lo que puede ser cuello de botella con grandes volúmenes.
- **Operaciones con Window Functions**: No utilizadas en este análisis, pero pueden ser costosas si no se particiona adecuadamente.

7. Uso de IA como Herramienta Auxiliar

Se consultó a IA para:

- **Configuración de spark**: Se utilizo para la configuracion de spark en el entorno virtual.
- **Sintaxis de transformaciones**: Cómo implementar ciertas operaciones como el cálculo de hora simulada.
- **Optimización**: Consejos para configurar el número de particiones adecuado.
- **Visualización**: Cómo generar gráficos directamente desde PySpark.

8. Conclusiones

El análisis demostró la capacidad de Spark para procesar eficientemente datos de movilidad urbana. Las operaciones distribuidas permitieron manejar grandes volúmenes de datos, identificando patrones clave que pueden informar decisiones de planificación urbana.