

Programación Paralela y Concurrente

Reporte de la práctica realizada en Spark

KRISTOFFER VAN STEEMBERGHE LUJÁN

2025

ÍNDICE GENERAL

1 | Capítulo 1 Introducción.

2 | Capítulo 2 Descripción del Dataset.

3 | Capítulo 3 Operaciones utilizadas.

- 3.1 Configuración del entorno. 6
- 3.2 Limpieza de datos. 6
- 3.3 Lectura del conjunto de datos limpio. 6
- 3.4 Análisis de congestión (Estaciones y líneas con mayor afluencia). 7
- 3.5 Análisis de horario (Meses y años con mayor afluencia). 7

4 | Capítulo 4 Análisis de los resultados.

- 4.1 Estaciones con mayor afluencia. 9
- 4.2 Líneas con mayor afluencia. 10
- 4.3 Meses con mayor afluencia. 11
- 4.4 Años con mayor afluencia. 12

5 | Capítulo 5 Fragmentos clave del código con comentarios

6

Capítulo 6
Uso de IA.

7 | **Capítulo 7**
Reflexión y conclusiones.

- 7.1 Reflexión. 17
- 7.2 Conclusiones. 17

INTRODUCCIÓN.

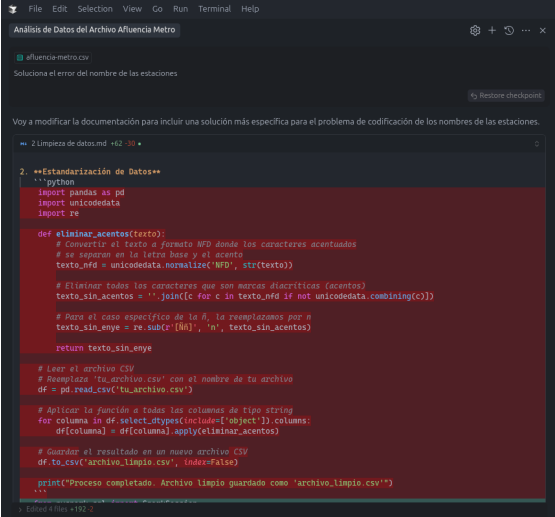
En este informe se encontrará gran parte de la documentación sobre la práctica en Spark, la cual fue realizada en un servidor local, los datos analizados fueron [afluencia-metro.csv](#) proporcionados por el Portal de Datos Abiertos de la CDMX. El objetivo de dicha práctica es responder ciertas preguntas realizando distintos análisis con operaciones como `reduceByKey`, `groupBy`, aplicación de filtros, lectura de los datos, creación de *dataframes* o *RDD*, entre otros.

DESCRIPCIÓN DEL DATASET.

Como se mencionó anteriormente, el *dataset* que utilizamos fue `afluencia-metro.csv`. El schema del *dataset* es el siguiente:

```
fecha: date
anio: integer
mes: string
linea: string
estacion: string
afluencia: integer
```

Una complicación que se me hizo fue que en el *dataset* había errores en los caracteres, debido a la codificación *UTF-8*, por lo que opté por pedirle ayuda a la IA para solucionar el error.



```
File Edit Selection View Go Run Terminal Help
Análisis de Datos del Archivo Afluencia Metro
afluencia-metro.csv
Soluciona el error del nombre de las estaciones
[?] Restore checkpoint
Voy a modificar la documentación para incluir una solución más específica para el problema de codificación de los nombres de las estaciones.
2 Limpieza de datos.md +42 -30
2. **Estandarización de Datos**
'''python
import pandas as pd
import unicodedata
import re

def eliminar_acentos(texto):
    # Convertir el texto a formato NFD donde los caracteres acentuados
    # se separan en la letra base y el acento
    texto_nfd = unicodedata.normalize('NFD', str(texto))

    # Eliminar todos los caracteres que son marcas diacríticas (acentos)
    texto_sin_acentos = ''.join(c for c in texto_nfd if not unicodedata.combining(c))

    # Para el caso específico de la Ñ, la reemplazamos por n
    texto_sin_ñye = re.sub('([Ñ])', '\n', texto_sin_acentos)

    return texto_sin_ñye

# Leer el archivo CSV
# Reemplaza 'tu_archivo.csv' con el nombre de tu archivo
df = pd.read_csv('tu_archivo.csv')

# Aplicar la función a todas las columnas de tipo string
for columna in df.select_dtypes(include=[object]).columns:
    df[columna] = df[columna].apply(eliminar_acentos)

# Guardar el resultado en un nuevo archivo CSV
df.to_csv('archivo_limpio.csv', index=False)

print("Proceso completado. Archivo limpio guardado como 'archivo_limpio.csv'")
'''
```

OPERACIONES UTILIZADAS.

3.1

Configuración del entorno.

Decidí seguir el ejemplo visto en clase para realizar esta práctica, por lo que decidí modificar un poco el `docker-compose.yml` dejando solamente los entornos de *Spark* y *Jupyter*.

Intenté trabajar directamente en Spark pero se me hizo un poco laborioso por lo que opté por *Jupyter*, creando todo el código en [Cursor](#) para fines prácticos.

3.2

Limpieza de datos.

Como comenté en la descripción del *dataset*, el archivo tenía algunas observaciones:

- En los nombres de las estaciones que contenían acentos o ñ, se mostraban algunos caracteres *raros* o especiales, lo cual ocasionaba un problema al querer tener el conjunto de datos de una forma limpia y adecuada.
- En la columna `linea` se encontraba la cadena de texto *Linea n* por lo que al querer mostrar información acerca de cada línea iba a provocar un error.

Debido a esto consideré que el archivo necesitaba una limpieza, o un formato, por lo que acudí con la IA para poder solucionar más que nada el error de los caracteres. El notebook con el cual se hizo dicho proceso es `limpiar-datos.ipynb`.

3.3

Lectura del conjunto de datos limpio.

Una vez que limpiamos nuestro *dataset* procedemos a cargarlo en un *dataframe*, para posteriormente realizar una vista previa para verificar que dicha limpieza haya sido efectiva. Después decidí imprimir el `schema` para ver el tipo de dato de cada columna con el que iba a trabajar y finalmente mostré el número de datos (filas) por simple curiosidad.

3.4

Análisis de congestión (Estaciones y líneas con mayor afluencia).

Para esta primer y segunda sección del análisis decidí usar `groupBy`. Las operaciones que se realizan para mostrar las estaciones y líneas con mayor afluencia son:

1. Agrupar los datos por `estacion-línea` y sumar la afluencia: Primero agrupo la información por la columna `estacion` o `línea` y luego sumo los valores de la columna `afluencia` para cada columna. Esto es necesario ya que queremos obtener el total de afluencia para cada columna, es decir, el total de personas que han estado en cada una de ellas.
2. Ordenar las estaciones por la afluencia total: Después de calcular la afluencia total, se organiza de mayor a menor según su afluencia.
3. Seleccionar las 10 *estaciones-líneas* con mayor afluencia: Debido a que existen muchas estaciones y muchas líneas decido seleccionar las 10 con mayor afluencia debido a motivos estéticos de la gráfica, ya que si graficamos todas sería difícil ver los detalles de la gráfica.
4. Configurar la gráfica: En esta parte es donde se ajustan distintos parámetros de la gráfica, como el color de las columnas, el título, etiquetas, entre otros, con el fin de que sea entendible fácilmente.
5. Mostrar la gráfica: Simplemente mostramos la gráfica con los resultados obtenidos.
6. Finalmente imprimimos también los resultados en pantalla.
7. **Motivo de uso:** Se decidió tomar esta metodología debido a que la aprendí en clase y un requisito era usar `groupBy`, esto es necesario porque la información de la afluencia de personas está dispersa por varias filas, a lo cual `groupBy` agrupa estos registros según sea necesario (estación o línea) y luego la función `sum` calcula el total de afluencia para cada caso.

3.5

Análisis de horario (Meses y años con mayor afluencia).

Para la tercera y cuarta sección del análisis decidí usar `reduceByKey` (RDD). Las operaciones que se realizan para mostrar los meses y años con mayor afluencia son:

1. Convertir el *dataframe* en un *RDD* (Resilient Distributed Dataset): Lo hice ya que otro requisito de la práctica era usar `reduceByKey-RDD`, además de que un *RDD* es una estructura de datos más básica y flexible, además esto nos permite utilizar operaciones como `map` o `reduce`.
2. Extraer los datos necesarios `anio`, `mes` y `afluencia`: Después de extraerlos procedo a crear una *tupla* donde la clave es una combinación de año y mes y el valor es la afluencia. Además al crear la *tupla* permite agrupar los datos (algo parecido a `groupBy`).
3. Usar `reduceByKey` para sumar las afluencias por mes: Después de haber agrupado los datos, necesitamos sumar las afluencias que pertenecen al mismo mes o año.
4. Ordenar los resultados por afluencia: Ordenamos los resultados de la afluencia de forma descendente.

5. Obtener los 10 meses con mayor afluencia: Como comenté anteriormente, por razones para el manejo de la gráfica solo utilizaré los primeros 10 meses o años.
6. Graficar las afluencias por mes o año: Procedo a graficar cada mes o año con su afluencia calculada.
7. Imprimir los resultados: Por último procedo a imprimir los resultados.
8. **Motivo de uso:** En este caso decidí usar `reduceByKey` ya que también era un requisito, además, la función `reduceByKey` se usa ya que es muy eficiente y rápido para sumar valores asociados a una misma clave en un *dataset*.

ANÁLISIS DE LOS RESULTADOS.

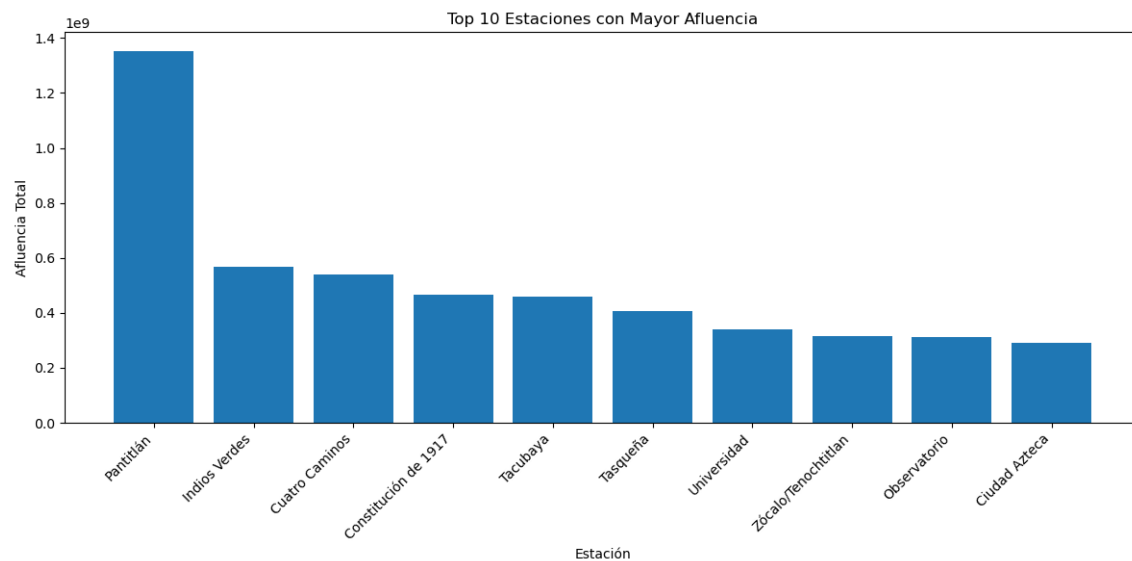
Debido a que cada operación la graficamos es más fácil identificar los resultados.

4.1

Estaciones con mayor afluencia.

Las 10 estaciones con mayor afluencia son:

1. Pantitlán
2. Indios Verdes
3. Cuatro Caminos
4. Constitución de 1917
5. Tacubaya
6. Tasqueña
7. Universidad
8. Zócalo/Tenochtitlan
9. Observatorio
10. Ciudad Azteca

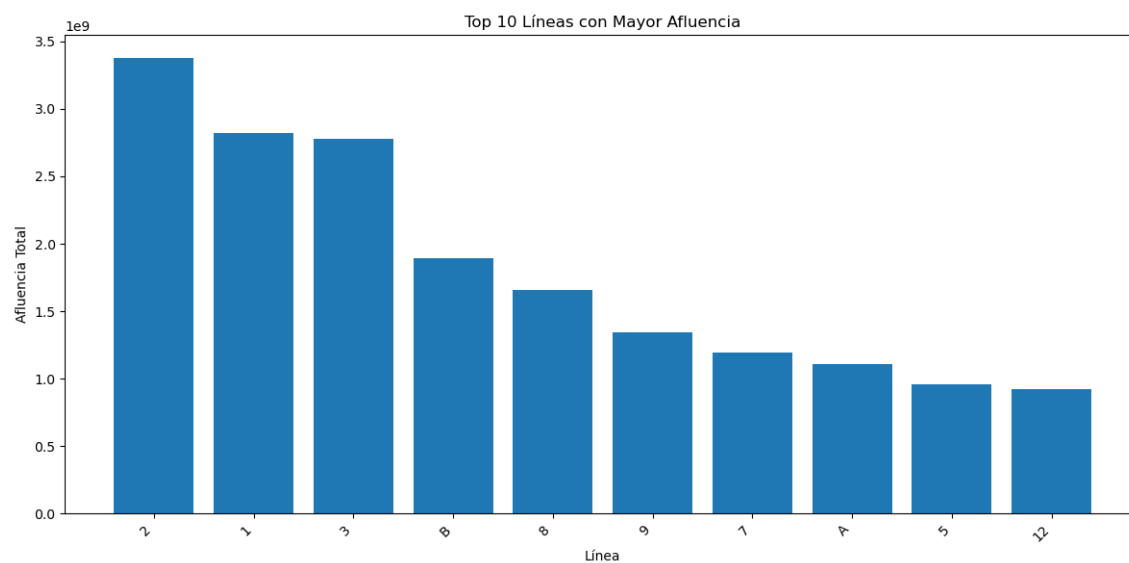


4.2

Líneas con mayor afluencia.

Las 10 líneas con mayor afluencia son:

1. Línea 2
2. Línea 1
3. Línea 3
4. Línea B
5. Línea 8
6. Línea 9
7. Línea 7
8. Línea A
9. Línea 5
10. Línea 12



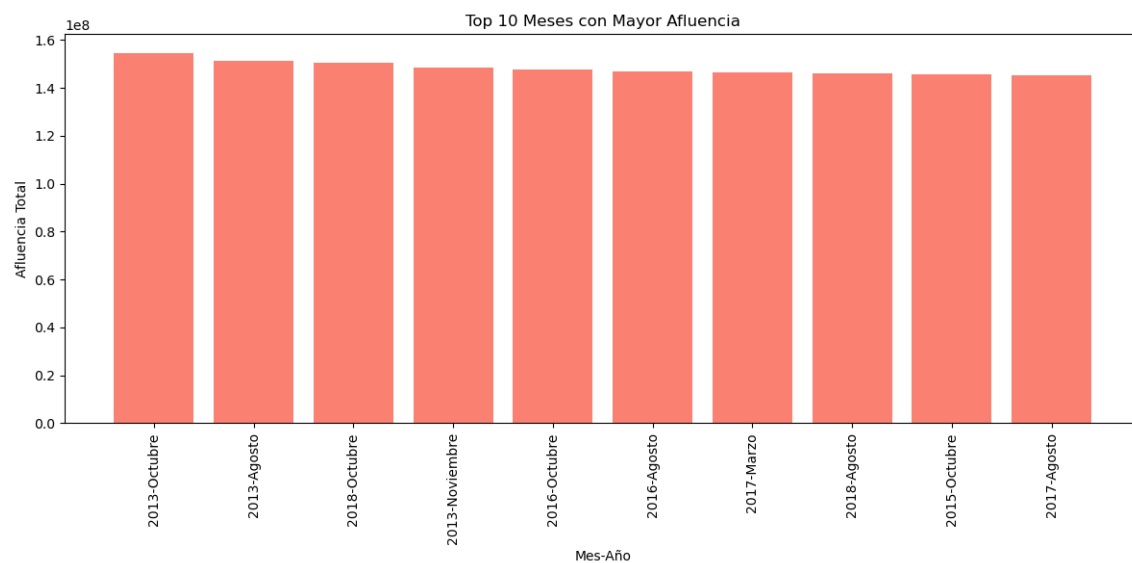
4.3

Meses con mayor afluencia.

Los 10 meses con mayor afluencia son:

1. Octubre del 2013
2. Agosto del 2013
3. Octubre del 2018
4. Noviembre del 2013
5. Octubre del 2016
6. Agosto del 2016
7. Mayo del 2017
8. Agosto del 2018
9. Octubre del 2015
10. Agosto del 2017

Por lo que observamos Octubre es el mes en el que más se usa el STC Metro.



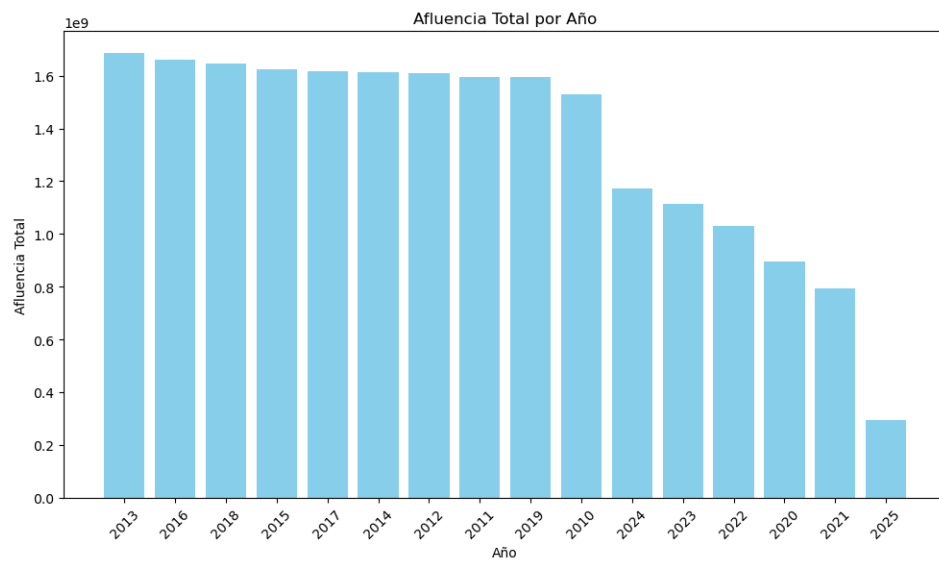
4.4

Años con mayor afluencia.

El orden de mayor afluencia con respecto a los años desde el 2013 hasta el 2015 es:

1. 2013
2. 2016
3. 2018
4. 2015
5. 2017
6. 2014
7. 2012
8. 2011
9. 2019
10. 2010
11. 2024
12. 2023
13. 2022
14. 2020
15. 2021
16. 2025

Podemos ver que el año con mayor afluencia es 2013 y el que tiene menor afluencia es 2025, sin embargo el año aún no acaba, por lo que puede cambiar de lugar.



FRAGMENTOS CLAVE DEL CÓDIGO CON COMENTARIOS

5

Un primer fragmento que me gustaría enseñar es el que generó la IA ya que no se me hubiera ocurrido dicha forma de solucionar el problema de los caracteres del *dataset*:

```
1 # Función para corregir la codificación de caracteres
2 def corregir_codificacion(df, column_name):
3     return df.withColumn(column_name,
4         regexp_replace(
5             regexp_replace(
6                 regexp_replace(
7                     regexp_replace(
8                         regexp_replace(
9                             regexp_replace(
10                                regexp_replace(
11                                    regexp_replace(
12                                        regexp_replace(
13                                            col(column_name),
14                                            "Ã±", "ó"),
15                                            "Ã¡", "á"),
16                                            "Ãí", "í"),
17                                            "Ã©", "é"),
18                                            "Ã±", "ñ"),
19                                            "Ã±", "ñ"),
20                                            "Ã±", "ñ"),
21                                            "Ã±", "ñ"),
22                                            "Ã±", "ñ"),
23                                            "Ã±", "ñ"),
24                                            "Ã±", "ñ")
25     )
```

Este código realiza una sustitución de los caracteres especiales por los caracteres correctos, conservando los acentos y las ñ.

Otro código relevante es el que utilicé para encontrar las estaciones-líneas con mayor afluencia, en este caso mostraré el de las líneas:

```
1 # Calcular la afluencia total por línea
2 afluencia_por_linea = df.groupby("línea").agg(sum("afluencia").alias("afluencia_total"))
3
4 # Ordenar por afluencia total y tomar las 10 estaciones con mayor afluencia
5 top_lineas = afluencia_por_linea.orderBy(desc("afluencia_total")).limit(10)
6
7 # Convertir a pandas para graficar
8 top_lineas_pd = top_lineas.toPandas()
9
10 # Configurar el estilo de la gráfica
11 plt.figure(2, figsize=(12, 6))
12 plt.bar(top_lineas_pd['línea'], top_lineas_pd['afluencia_total'])
13 plt.title('Top 10 Líneas con Mayor Afluencia')
14 plt.xlabel('Línea')
15 plt.ylabel('Afluencia Total')
16 plt.xticks(rotation=45, ha='right')
17 plt.tight_layout()
18
19 # Mostrar la gráfica
20 plt.show()
21
22 # Mostrar la tabla de datos
23 top_lineas.show()
```

Por último intenté guardar cada una de las gráficas con el siguiente código pero al momento de guardar las imágenes solo se muestran en blanco:

```
1 plt.figure(1) # Top 10 estaciones
2 plt.savefig('./work/results/graphics/top10_estaciones.png')
3
4 plt.figure(2) # Top 10 líneas
5 plt.savefig('./work/results/graphics/top10_lineas.png')
6
7 plt.figure(3) # Afluencia por meses
8 plt.savefig('./work/results/graphics/afluencia_por_meses.png')
9
10 plt.figure(4) # Afluencia por años
11 plt.savefig('./work/results/graphics/afluencia_por_anos.png')
```

USO DE IA.

El primer uso que realicé de la IA fue como encontrar o ver el token para acceder al servidor de Jupyter, ya que al momento de levantar el `compose` no me daba ningún *log* o información, acudí a [Claude](#) y la respuesta que obtuve era ejecutar en la terminal `sudo docker logs jupyter`, lo cual me devolvió todos los *logs* de Jupyter y encontré el link del servidor que también contenía el token para acceder.

El segundo uso fue para solucionar los "errores" que tenía mi *dataset*, directamente desde el chat de IA de [Cursor](#) le añadí como contexto el archivo `afluencia-metro.csv` y le pedí que creará un notebook que solucionara los errores en el nombre de las estaciones y eliminara la sentencia "Linea *n*" de la columna `linea`, el cual me generó el código `limpieza-datos.ipynb` el cual lee mi archivo `csv`, lo convierte a un *dataframe*, crea una función llamada `corregir_codificacion` a la cual se le pasa el *dataframe* y el nombre de la columna, después hace funciones anidadas de `regexp_replace` la cual sustituye los caracteres erróneos por los correctos. Posteriormente elimina la sentencia de la columna `linea` dejando solo el nombre de cada línea y guarda todos los cambios en un nuevo archivo llamado `afluencia-metro-corregido.csv`.

La última consulta que le realicé fue justamente como guardar las gráficas de `matplotlib` y me generó el código mostrado en el anterior capítulo pero como comenté tiene un error ya que solo guarda una imagen en blanco.

Otro uso de la IA aparte de las consultas es el autocompletado que ofrecen los editores de texto, este simplemente lo uso para codificar más rápido o corregir errores de sintaxis.

REFLEXIÓN Y CONCLUSIONES.

7.1 Reflexión.

El análisis realizado sobre los datos de afluencia del STC Metro en la CDMX ha proporcionado una visión clara de los patrones de movilidad en el sistema de transporte. A través de herramientas como Spark, Jupyter y las bibliotecas de Python (pandas, matplotlib), se ha logrado procesar y analizar grandes volúmenes de datos para identificar las estaciones, líneas, meses y años con mayor concentración de personas.

Este tipo de análisis no solo facilita la comprensión de cómo los usuarios interactúan con el metro, sino que también puede ser crucial para la toma de decisiones operativas. Por ejemplo, al identificar las líneas con mayor afluencia, se pueden hacer ajustes en la programación de trenes o en la asignación de recursos, mejorando así la eficiencia del sistema y la experiencia del usuario. Además, el análisis de afluencia por mes y año ayuda a identificar tendencias estacionales y a anticipar las necesidades de infraestructura y servicios en el futuro.

El uso de `reduceByKey` y otras funciones de agregación en Spark ha demostrado ser esencial para manejar los grandes volúmenes de datos, permitiendo realizar cálculos de manera eficiente. Además, la visualización de los resultados a través de gráficas ha sido fundamental para hacer comprensibles los patrones encontrados y para comunicar los resultados de manera clara.

7.2 Conclusiones.

1. **Identificación de Líneas de Alta Afluencia:** El análisis ha permitido identificar las líneas del metro con mayor afluencia, lo que es crucial para mejorar la planificación de los recursos y optimizar la distribución de los trenes en esas rutas más congestionadas.
2. **Análisis de Tendencias Temporales:** Aunque el conjunto de datos no proporciona información detallada sobre los horarios, al analizar la afluencia por mes y año, se ha logrado identificar ciertos picos en la demanda, lo que sugiere que existen meses específicos con una mayor demanda. Este tipo de análisis permite predecir periodos de alta demanda, como temporadas

festivas o eventos especiales en la ciudad.

3. **Optimización del Sistema de Transporte:** Con estos resultados, el STC Metro puede tomar decisiones informadas sobre cómo mejorar su servicio, como ajustar la frecuencia de los trenes durante los meses más concurridos o aumentar la capacidad de las líneas con mayor número de usuarios.

En resumen, el análisis de datos de afluencia del metro en la Ciudad de México no solo ha sido útil para comprender mejor los patrones de movilidad, sino que también ha mostrado el potencial de las herramientas modernas de análisis de datos para mejorar los sistemas de transporte urbano. Este enfoque de datos puede ser clave para enfrentar los retos de la urbanización creciente y la optimización de los servicios públicos en grandes ciudades.