



Caracterización de datos de trayectorias individuales

Presentado por:
Jorge Rafael Martínez Buenrostro

Asesora: Dra. Elizabeth Pérez Cortés

México, CDMX, a 6 de agosto de 2025

Resumen

Una trayectoria se define como la secuencia de desplazamientos realizada por un individuo en movimiento, compuesta por tres elementos principales: **puntos de recorrido, tiempos de pausa y longitudes de vuelo**. Los puntos de recorrido corresponden a las ubicaciones o pasos específicos por los que transita el individuo. Los tiempos de pausa representan los intervalos durante los cuales el individuo permanece detenido en un mismo punto de recorrido. Por último, la longitud de vuelo se refiere a la distancia recorrida entre dos puntos de recorrido consecutivos. En el presente trabajo se describe el proceso de caraterización de datos de movilidad, es decir, el proceso de limpieza y depuración de la información, mediante el cual elimina aquellos campos y registros que no le aportan valor a la trayectoria individual. El objetivo es identificar la mayor cantidad de trayectorias individuales, para así poder crear un modelo que permita simular el movimiento de individuos.

Contenido

Lista de Códigos	IV
1. Introducción del Proyecto	1
1.1. Descripción general del proyecto	1
1.2. Objetivos y propósitos	1
1.3. Alcance del sistema	1
2. Requisitos del sistema	2
2.1. Instrucciones de instalación	2
3. Caracterización de datos de trayectorias individuales	3
3.1. Exploración inicial del conjunto de datos	4
3.2. Dimensiones del conjunto de datos	5
3.3. Depuración de columnas	5
3.4. Depuración de filas	6
4. Análisis del conjunto de datos	11
5. Observaciones y recomendaciones	14
5.1. Calidad de datos	14
5.2. Duplicación significativa de registros	14
5.3. Persistencia de individuos	14
A. Uso de Docker y Docker Compose	16
A.1. ¿Qué son Docker y Docker Compose?	16
A.2. Instalación en Linux (Ubuntu/Debian)	16
A.3. Instalación en Windows	17
A.4. Descripción del archivo <code>docker-compose.yml</code>	18
A.5. Scripts de control del contenedor	21
A.5.1. <code>start_container.sh</code>	21
A.5.2. <code>restart_container.sh</code>	21
A.5.3. <code>stop_container.sh</code>	22
A.6. Proceso de uso y desarrollo del contenedor	22
B. Scripts para la caracterización de datos	24

Lista de Figuras

3.1. Frecuencia de aparición de los valores de 'device_horizontal_accuracy'.	7
3.2. Frecuencia de aparición de los identificadores únicos.	7
3.3. Comparación de histogramas por rangos de repeticiones.	8
3.4. Frecuencia de aparición de los identificadores únicos después de eliminar duplicados.	9
3.5. Comparación de histogramas por rangos de repeticiones.	10
4.1. Comparación de distribución de individuos por día.	12
4.2. Distribución de individuos por días.	13

Lista de Códigos

2.1. Iniciar contenedor del proyecto.	2
2.2. Iniciar contenedor del proyecto.	2
2.3. Reiniciar contenedor del proyecto.	2
A.1. Actualizar el sistema.	16
A.2. Instalar Docker.	17
A.3. Verificar instalación de Docker.	17
A.4. Instalar Docker Compose.	17
A.5. Verificar instalación de Docker Compose.	17
A.6. Verificar instalación de Docker y Docker Compose.	17
A.7. Archivo docker-compose.yml	18
A.8. Script para iniciar el contenedor.	21
A.9. Script para reiniciar el contenedor.	21
A.10. Script para detener y eliminar el contenedor y sus volúmenes.	22
A.11. Dar permisos de ejecución a los scripts.	22
A.12. Iniciar contenedor y ejecutar el proyecto.	22
A.13. Reiniciar el contenedor completamente.	22
A.14. Eliminar el contenedor y limpiar el entorno.	22
B.1. csv_glance.py, exploración inicial del conjunto de datos.	24
B.2. csv_count_registers.py, conteo de registros en el conjunto de datos.	25
B.3. remove_columns.py, eliminación de campos innecesarios en el conjunto de datos.	26
B.4. unique_values.py, obtención de valores únicos de la columna 'device_horizontal_accuracy'.	27
B.5. accuracy_histogram.py, creación de un histograma de frecuencias de la columna 'device_horizontal_accuracy'.	29
B.6. identifier_histogram.py, creación de un histograma de frecuencias de la columna 'identifier'.	32
B.7. identifier_histogram_detailed.py, análisis de frecuencias de la columna 'identifier'.	35
B.8. csv_deduplicate.py, eliminación de duplicados en el conjunto de datos.	39
B.9. identifier_histogram_daily.py, análisis de frecuencias de la columna 'identifier' por día.	40
B.10. migrate_csv_to_postgres.py, migración de datos desde un CSV a una base de datos PostgreSQL.	44

Capítulo 1

Introducción del Proyecto

1.1 Descripción general del proyecto

La simulación de una red de comunicaciones con dispositivos personales requiere modelos que representen fielmente los patrones de movimiento de las personas. De lo contrario, las conclusiones derivadas de dicha simulación pueden ser poco útiles. Para avanzar hacia la definición de un modelo de trayectorias individuales, se propone caracterizar los datos de una base existente que permita modelar trayectorias de forma eficaz.

1.2 Objetivos y propósitos

El objetivo principal del proyecto es obtener una caracterización estadística de las trayectorias individuales.

Los propósitos específicos son:

- Caracterizar la base de datos para extraer las trayectorias contenidas.
- Aplicar un modelo de inteligencia artificial para identificar y analizar dichas trayectorias.

1.3 Alcance del sistema

El sistema se enfoca en la identificación de trayectorias peatonales individuales y su análisis mediante herramientas de IA. El alcance incluye:

- Caracterización de la base de datos existente.
- Identificación de trayectorias individuales.
- Generación de reportes y visualizaciones de los resultados.

No se incluye la creación de modelos de IA desde cero; se emplearán herramientas y modelos ya existentes.

Capítulo 2

Requisitos del sistema

Docker: version 28.2.2, build e6534b4

Docker Compose: version 1.29.2, build unknown

2.1 Instrucciones de instalación

1. Clonar el repositorio del proyecto desde el siguiente enlace el proyecto se encuentra dentro de la carpeta **Implementación**.
2. Para levantar y acceder al contenedor, ejecutar el siguiente script:¹

```
1 ./start_container.sh      # Linux/Mac  
2 .\start_container.bat    # Windows
```

Código 2.1: Iniciar contenedor del proyecto.

3. Para cerrar el contenedor del proyecto, ejecutar el siguiente script:

```
1 ./stop_container.sh      # Linux/Mac  
2 .\stop_container.bat    # Windows
```

Código 2.2: Iniciar contenedor del proyecto.

4. Para ver reflejados los cambios realizados en el código, ejecuta el siguiente script:

```
1 ./restart_container.sh    # Linux/Mac  
2 .\restart_container.bat  # Windows
```

Código 2.3: Reiniciar contenedor del proyecto.

¹Para más detalles sobre el uso de Docker y Docker Compose, consulte el Apéndice A.

Capítulo 3

Caracterización de datos de trayectorias individuales

El análisis de datos comienza con una etapa fundamental, la caracterización del conjunto de datos. Esta fase tiene como objetivo examinar y comprender la estructura, el contenido y las principales propiedades de los datos antes de aplicar técnicas analíticas más complejas.

En el caso de los datos de trayectorias individuales, la caracterización permite identificar posibles inconsistencias, redundancias y elementos irrelevantes que puedan afectar la calidad del análisis. Las tareas principales llevadas a cabo en esta etapa son las siguientes:

- Explorar las primeras filas del conjunto de datos para obtener una visión general de su estructura.
- Verificar la cantidad total de registros y columnas disponibles.
- Identificar y eliminar columnas que no aportan información relevante para el análisis o inconsistentes.
- Identificar y eliminar las filas que no aportan información relevante para el análisis o inconsistentes.

A continuación, se describen en detalle las acciones específicas realizadas durante el proceso de caracterización.

3.1 Exploración inicial del conjunto de datos

Como primer paso en la caracterización, se realiza una exploración preliminar del conjunto de datos con el fin de comprender su estructura general. Para ello, se inspeccionan las primeras dos filas del conjunto de datos, lo cual permite identificar las columnas presentes y observar ejemplos representativos de sus valores.

El código utilizado para realizar esta exploración se encuentra en el Apéndice B.1. A continuación, se presenta un resumen de las columnas detectadas junto con una muestra de sus respectivos valores:

1. **id**: Identificador numérico único por registro
['34284565', '34284566']
2. **identifier**: UUID del dispositivo
['f2640430-7e39-41b7-80bb-3fddaa44779c']
3. **identifier_type**: Tipo de ID (ej. 'gaid' para Android)
['gaid', 'gaid']
4. **timestamp**: Fecha-hora del registro
['2022-11-07 02:04:21']
5. **device_lat/device_lon**: Coordenadas GPS
['21.843149'], [-102.196838']
6. **country_short/province_short**: Códigos de ubicación
['MX'], ['MX.01']
7. **ip_address**: Dirección IPv6
['2806:103e:16::']
8. **device_horizontal_accuracy**: Precisión GPS en metros
['8.0']
9. **source_id**: Hash de la fuente de datos
['449d086d...344']
10. **record_id**: Hash único por registro
['77d795df...']
11. **home_country_code**: País de residencia
['MX']
12. **home_geog_point/work_geog_point**: Coordenadas en WKT
['POINT(-102.37038 22.20753)']
13. **home_hex_id/work_hex_id**: ID hexagonal (H3)
['85498853fffffff']
14. **data_execute**: Fecha de procesamiento
['2023-05-30']
15. **time_zone_name**: Zona horaria
['America/Mexico_City']

3.2 Dimensiones del conjunto de datos

Para verificar las dimensiones del conjunto de datos, se utiliza la biblioteca Dask, que permite trabajar con grandes volúmenes de datos de manera eficiente. Para hacer uso de esta biblioteca se usa el lenguaje de programación Python. El código del Apéndice B.2 nos da un ejemplo de su uso para esta etapa. El resultado de este script da como resultado que el conjunto de datos contiene un total de **69,980,000** registros y **19** campos. Esto indica que hay una cantidad significativa de datos disponibles para el análisis.

3.3 Depuración de columnas

Dado que el conjunto de datos original contiene 19 campos, es fundamental identificar y eliminar aquellas columnas que no aportan valor al análisis. Para ello, se realiza una revisión de los valores únicos presentes en cada campo, con el objetivo de detectar información redundante o irrelevante. A partir de este análisis, se identifican las siguientes columnas como innecesarias para los fines del estudio:

- `id`
- `identifier_type`
- `country_short`
- `province_short`
- `ip_address`
- `source_id`
- `home_country_code`
- `home_geog_point`
- `work_geog_point`
- `home_hex_id`
- `work_hex_id`
- `data_execute`

En lugar de eliminar columnas explícitamente, se opta por seleccionar únicamente aquellas que se desean conservar. El código utilizado para esta tarea se encuentra incluido en el Apéndice B.3. Dicho script emplea la biblioteca `dask` para cargar y guardar una nueva versión del conjunto de datos que contiene exclusivamente las siguientes columnas relevantes:

- `identifier`
- `timestamp`
- `device_lat`
- `device_lon`

- `device_horizontal_accuracy`
- `record_id`
- `time_zone_name`

Como resultado, se genera un nuevo archivo CSV que conserva únicamente la información útil para el análisis posterior, optimizando así el tamaño y la calidad del conjunto de datos.

3.4 Depuración de filas

Una vez obtenida una versión más ligera del conjunto de datos, el siguiente paso consiste en identificar y eliminar aquellas filas que no aportan valor al análisis. Para ello, se generan representaciones gráficas que permiten observar la distribución de los datos y facilitar la toma de decisión. Las columnas seleccionadas para este proceso fueron:

- `identifier`: Identificador único del dispositivo.
- `device_horizontal_accuracy`: Precisión del GPS en metros. A menor valor, mayor precisión.

La primera columna a analizar será `device_horizontal_accuracy`, que refleja la precisión del GPS en metros. Este valor depende tanto del sistema de medición como de la fuente de datos, y suele clasificarse según la siguiente escala:

- GPS puro (satelital): 1–20 metros.
- A-GPS (asistido por red): 5–50 metros.
- Triangulación por WiFi o redes móviles: 20–500 metros.
- Geolocalización por IP: 1000–5000 metros.

Con base en esta escala, primero hay que identificar el rango de valores presentes en la columna. Para ello se utiliza el código mostrado en el Apéndice B.4, el cual extrae los valores únicos de `device_horizontal_accuracy` y los guarda en un archivo de texto. El resultado indica que los valores oscilan entre 0.916 y 199.9, lo que permite construir un histograma (Apéndice B.5) para analizar la frecuencia de cada valor y así evaluar su relevancia para el análisis. El resultado se muestra en la siguiente figura:

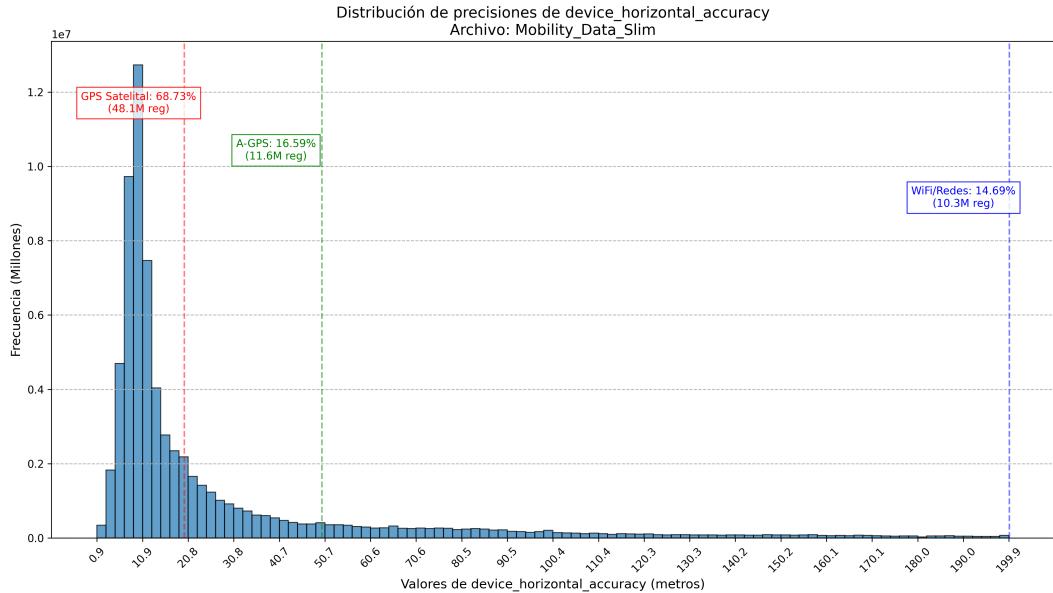


Figura 3.1: Frecuencia de aparición de los valores de 'device_horizontal_accuracy'.

Para el objetivo de este proyecto, se busca que la configuración del GPS sea lo más precisa posible, por lo que aquellos que estén dentro del rango del GPS puro (1-20 metros) son los más relevantes. Como se puede ver en la Figura 3.1, el **68.73 %** de los valores se encuentran dentro de este rango. Sin embargo, el **31.27 %** de registros con están por encima de este rango, precisión A-GPS (5-50 metros) y triangulación por WiFi/red móvil (20-500 metros).

La siguiente columna a evaluar es **identifier**, corresponde al identificador único de cada dispositivo. Para analizar la frecuencia de aparición de estos valores se emplea un script que agrupa las repeticiones por rangos y grafica la cantidad de valores únicos usando escala logarítmica (ver Apéndice B.6).

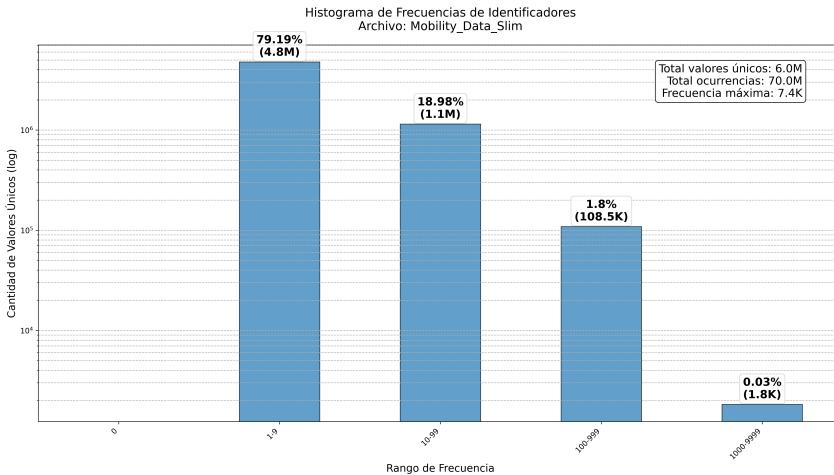


Figura 3.2: Frecuencia de aparición de los identificadores únicos.

Ejecutar este script permite saber que el total de individuos es de **6,022,772** de los cuales el **79.19 %** tienen una frecuencia de aparición de una a nueve veces, esto es **4,769,317** de individuos. Así mismo de la Figura 3.2 se observa que hay poco más de un **20 %** de individuos con más de 99 repeticiones. Por lo que se necesita hacer un análisis más detallado, para ello se ejecuta el código del Apéndice B.7, el cual segmenta los datos en tres rangos: 1-99, 100-1000 y 1001-10000 repeticiones.

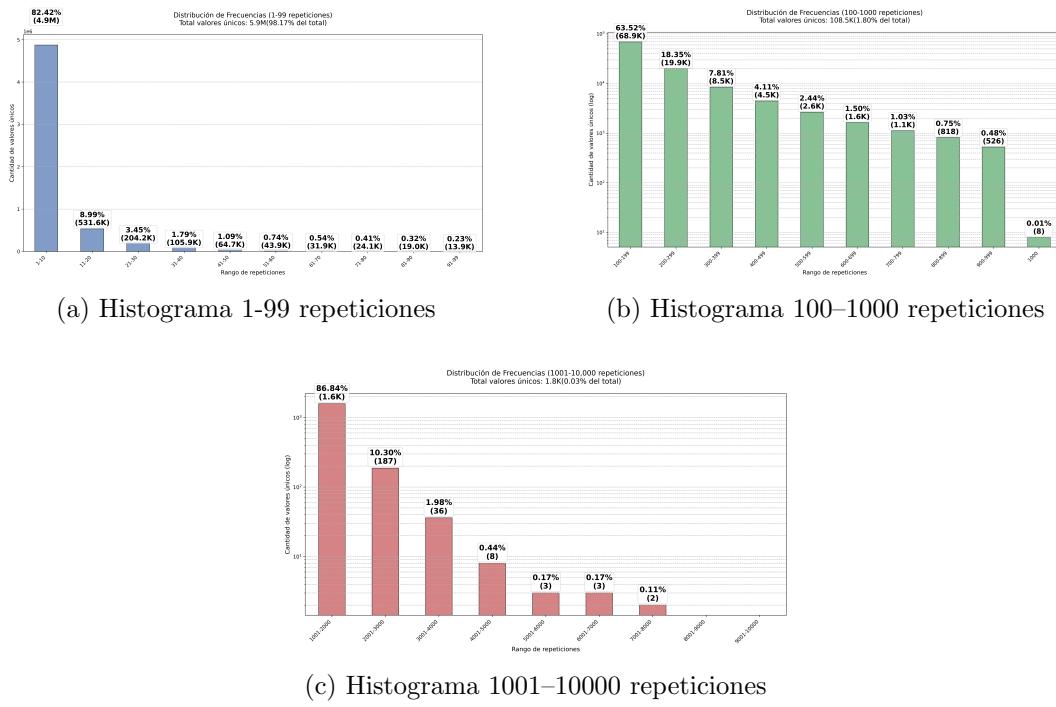


Figura 3.3: Comparación de histogramas por rangos de repeticiones.

Con la información obtenida de los histogramas de la figura anterior, se puede observar que el **98.17 %** de los identificadores únicos tienen entre 1 y 99 repeticiones, lo que equivale a **5,912,437** individuos. Por otro lado, el **1.83 %** restante tiene entre 100 y 10,000 repeticiones, lo que equivale a **110,335** individuos. Con base en esta información aún no se puede determinar que registros eliminar.

Por lo que el siguiente paso consiste en eliminar aquellos registros duplicados, es decir, aquellos que tengan el mismo valor en las columnas: `identifier`, `timestamp`, `device_lat` y `device_lon`. Para ello se utiliza el código del Apéndice B.8, que elimina los duplicados y genera un nuevo archivo CSV con los registros de individuos.

Con este nuevo archivo se vuelve a realizar el análisis de frecuencia de aparición de individuos. En la siguiente figura se muestra el histograma de la frecuencia de aparición de los identificadores únicos

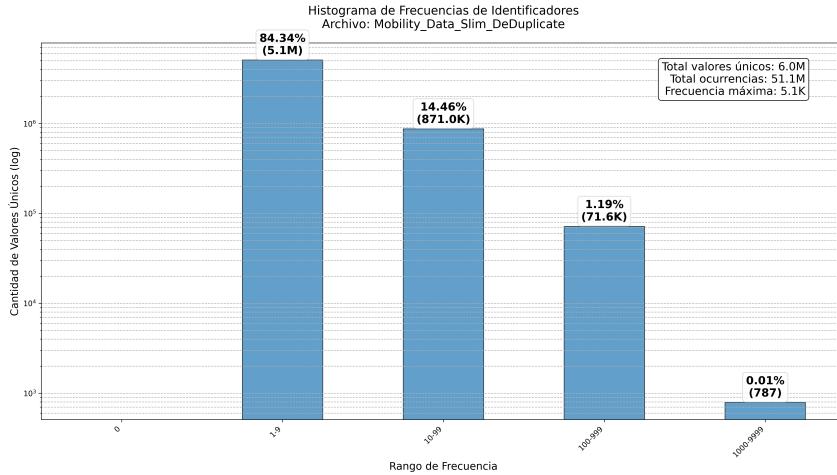


Figura 3.4: Frecuencia de aparición de los identificadores únicos después de eliminar duplicados.

Comparando los resultados de la Figura 3.2 y la Figura 3.4 podemos destacar varios hallazgos importantes:

- El número de individuos (**6,022,772**) se mantuvo sin cambios.
- La eliminación del **27 %** de registros. De **70 millones** a **51 millones** de registros.
- La reducción del **31.1 %** en la frecuencia máxima de aparición (de 7,400 a 5,100) corrige sesgos que afectaban especialmente a individuos con alta frecuencia de registros repetidos.

De la Figura 3.5 se puede observar que la distribución de los individuos se mantiene similar; sin embargo, ahora el número de individuos que tienen entre 1 y 99 repeticiones aumentó del **98.17 %** al **98.8 %**, lo que equivale a un aumento de **37,899** individuos. Por otro lado, el número de individuos con más de 100 repeticiones bajó del **1.83 %** al **1.2 %**, lo que equivale a una disminución de **37,899** individuos, lo que sugiere que la mayoría de los individuos no generan datos de manera continua o frecuente. Sin embargo, es importante destacar que al eliminar aquellos puntos de recorrido duplicados por individuo permite asumir que los puntos de recorrido restantes son más representativos de la movilidad real de los individuos.

Dado los resultados obtenidos en esta etapa de la caracterización, se concluye que no es necesario eliminar ninguna fila del conjunto de datos, ya que la depuración de columnas y la eliminación de duplicados han sido suficientes para optimizar la calidad del conjunto de datos.

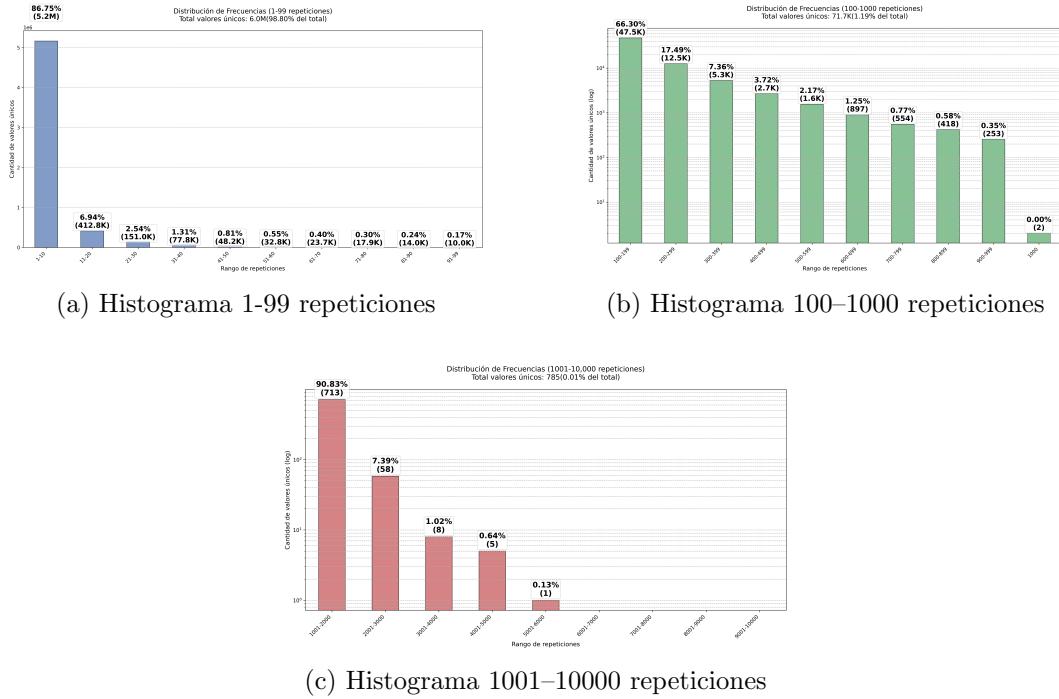
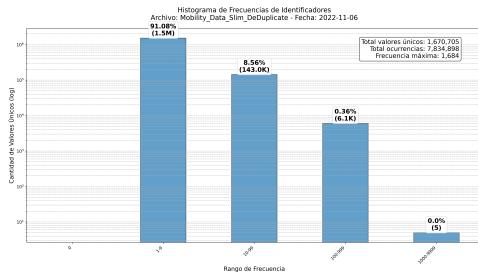


Figura 3.5: Comparación de histogramas por rangos de repeticiones.

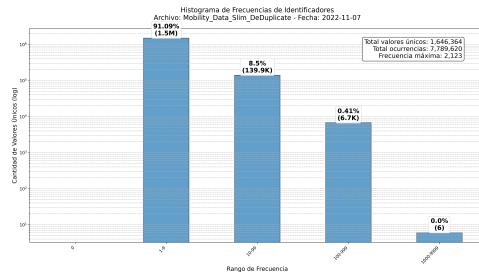
Capítulo 4

Análisis del conjunto de datos

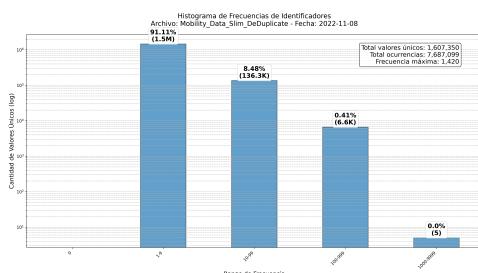
En esta sección se presenta un análisis detallado del conjunto de datos, que incluye la identificación de patrones y tendencias en las trayectorias individuales. El conjunto de datos contiene puntos de recorrido obtenidos a lo largo de diez días. Por lo que el primer acercamiento del análisis es poder ver la distribución de los puntos de recorrido en cada día registrado por medio un histograma por día. Para crear estos histogramas se ejecuta el código del Apéndice B.9. A continuación, se muestran estos histogramas.



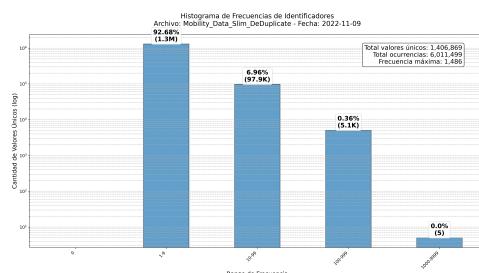
(a) Histograma del 06/Nov/2022



(b) Histograma del 07/Nov/2022



(c) Histograma del 08/Nov/2022



(d) Histograma del 09/Nov/2022

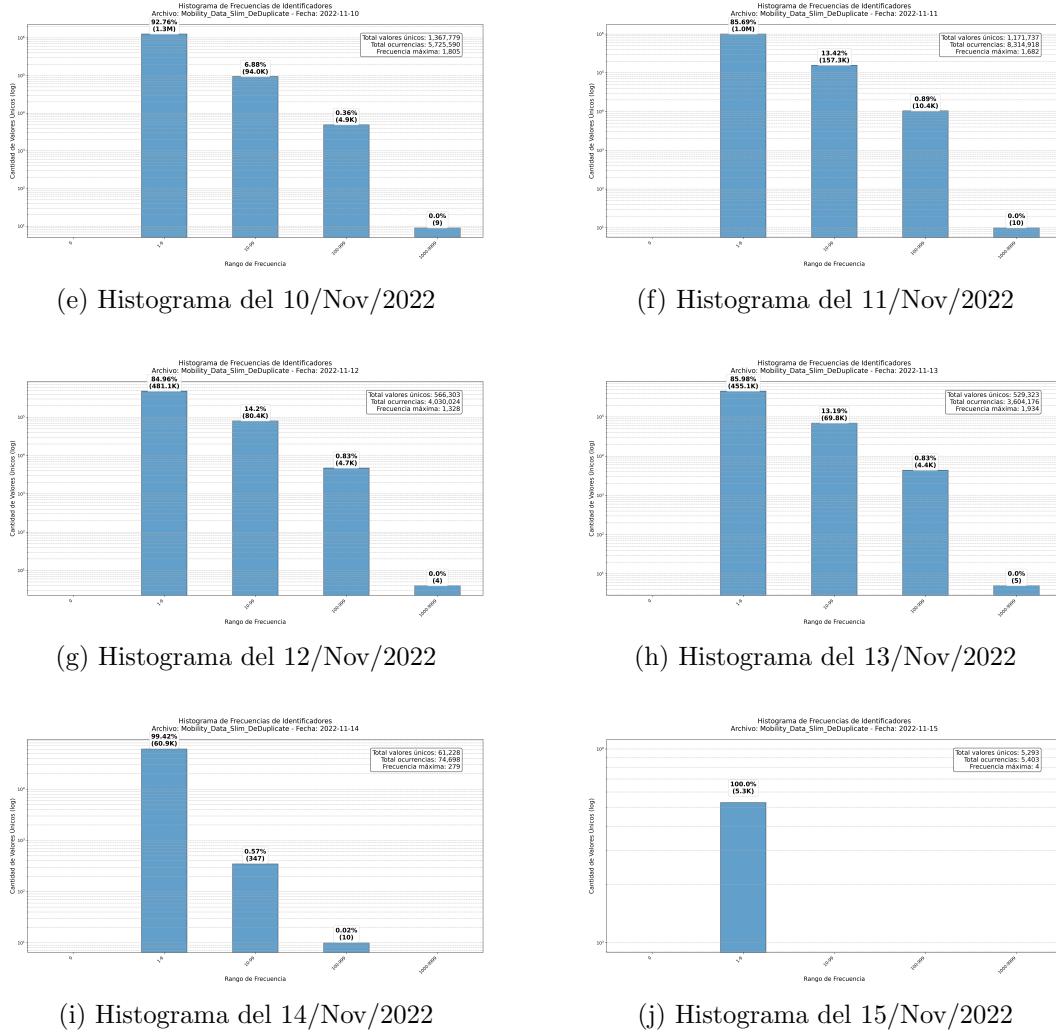


Figura 4.1: Comparación de distribución de individuos por día.

De la figura anterior se puede observar que la distribución de individuos por día es bastante similar a la distribución vista en la Figura 3.4, donde se observa que la mayoría de los individuos tienen una frecuencia de aparición baja. Además se puede ver que los primeros seis días hay en promedio **1,400,000** de individuos, mientras que los últimos cuatro días este valor va disminuyendo hasta llegar a **5,293** individuos el día 15 de noviembre. Esto puede ser un indicativo de que la recolección de datos no fue constante a lo largo del tiempo, lo que podría deberse a diversos factores como problemas técnicos o cambios en el comportamiento de los individuos.

Un patrón que puede dar mucha información es determinar cuantos individuos tienen puntos de recorrido en más de un día. Para esto se ejecuta el código del Apéndice

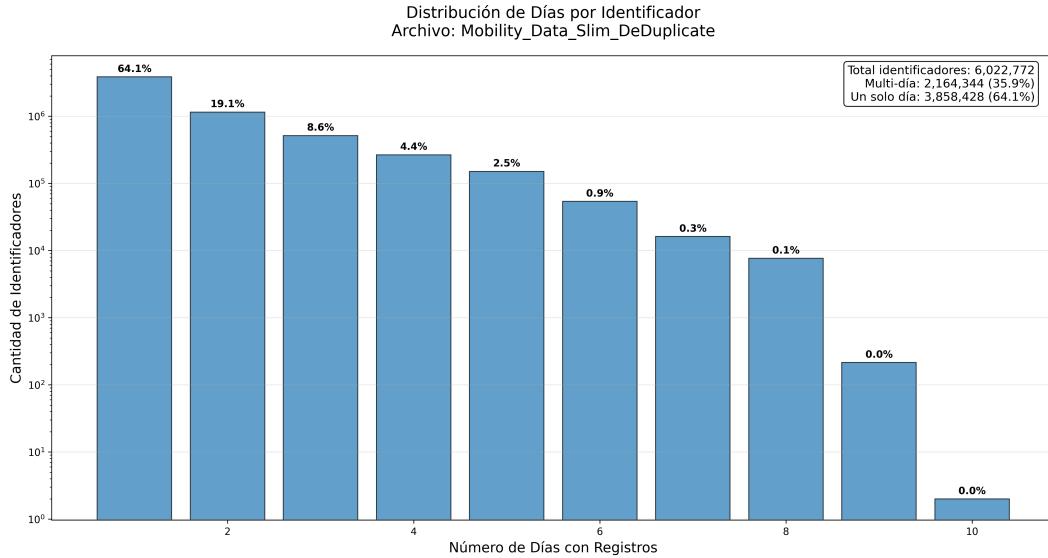


Figura 4.2: Distribución de individuos por días.

El resultado de este análisis se muestra en la figura anterior, donde se puede observar que el **64.06 %** de los individuos tienen puntos de recorrido en un día nada más, esto representa **3,858,428** individuos. Por otro lado, el porcentaje de individuos que tienen puntos de recorrido en más de seis días es del **0.4 %**, lo que representa **77,878** individuos. Este análisis es importante porque permite identificar a los individuos que tienen un comportamiento más rutinario y que podrían ser más relevantes para el estudio de patrones de movilidad.

Con esta información se puede realizar un análisis más detallado de las trayectorias de los individuos del conjunto de datos. Por ejemplo, se puede identificar a los individuos que tienen puntos de recorrido en más de un día y que además tienen una alta frecuencia de aparición en esos días. Para esto se puede utilizar un gestor de base de datos como PostgreSQL, que permite realizar consultas complejas y obtener información más detallada sobre los individuos y sus trayectorias.

Para lograr esto, se puede utilizar el servicio de PostgreSQL creado al levantar el contenedor de Docker, como se describe en el Apéndice A.7. Es necesario crear una base de datos y una tabla para almacenar los datos. Importar los datos del archivo CSV a esta tabla se hace mediante el código del Apéndice B.10. Una vez que los datos están en la base de datos, se pueden realizar consultas para obtener información más detallada sobre los individuos y sus trayectorias. A continuación, se exploraran algunos queries que permiten obtener información relevante sobre los individuos y sus trayectorias.

Capítulo 5

Observaciones y recomendaciones

Durante el proceso de caracterización de datos de trayectorias individuales del Capítulo 3 se identificaron aspectos importantes que impactan la calidad del análisis y la validez de los resultados obtenidos. Esta sección presenta las principales observaciones derivadas del proceso y las recomendaciones para mejorar futuros análisis.

5.1 Calidad de datos

La caracterización reveló que el **68.73 %** de los registros tienen una precisión GPS dentro del rango satelital (1-20 metros), mientras que el **31.27 %** presenta una menor precisión. **Recomendación:** Se sugiere establecer un filtro que conserve únicamente registros con precisión GPS dentro del rango satelital (1-20 metros), en este caso el campo responsable es `device_horizontal_accuracy`. Esto garantizará mayor confiabilidad en las coordenadas de posición.

5.2 Duplicación significativa de registros

Se detectó un **27 %** de registros duplicados en el conjunto de datos original, lo que representa aproximadamente **19 millones** de entradas redundantes. **Recomendación:** Implementar rutinas automáticas de detección de duplicados como paso inicial. Esto optimizará el almacenamiento y procesamiento, además de mejorar la calidad del análisis.

5.3 Persistencia de individuos

Los datos muestran una marcada disminución en el número de individuos registrados a lo largo del periodo de tiempo, pasando de aproximadamente **1.4 millones** de individuos en los primeros seis días a apenas **5,293** individuos para el último día. **Recomendación:** Esta tendencia sugiere posibles problemas en la recolección de

datos o en la participación de los individuos. Se recomienda investigar la causa de esta disminución, y considerar recolectar datos en períodos más cortos para mantener una muestra representativa. Incluso delimitar la recolección a una zona geográfica específica.

Apéndice A

Uso de Docker y Docker Compose

En la sección ?? se establece como requisito el uso de Docker y Docker Compose para la ejecución del proyecto. A continuación, se detallan las instrucciones necesarias para su instalación, ya que ambas herramientas son fundamentales para la implementación. Además, se describe el archivo `docker-compose.yml`, el cual permite crear un contenedor que incluye todas las dependencias requeridas para el correcto funcionamiento del sistema.

A.1 ¿Qué son Docker y Docker Compose?

Docker es una plataforma de virtualización ligera que permite desarrollar, empaquetar y ejecutar aplicaciones en contenedores aislados. Un contenedor incluye el código, las dependencias y configuraciones necesarias para que la aplicación se ejecute de manera consistente en cualquier entorno. Esto facilita la portabilidad, escalabilidad y despliegue de software.

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones multicontenedor mediante archivos de configuración YAML. A través de un solo archivo `docker-compose.yml`, es posible especificar los servicios, redes y volúmenes que componen una aplicación, simplificando así su orquestación.

Estas herramientas son fundamentales en este proyecto para garantizar que el entorno de ejecución sea replicable y controlado, independientemente del sistema operativo o configuración local del usuario.

A.2 Instalación en Linux (Ubuntu/Debian)

Para instalar Docker y Docker Compose en un sistema Linux basado en Debian o Ubuntu, siga los siguientes pasos:

1. Actualizar los paquetes del sistema:

```
1 sudo apt update  
2 sudo apt upgrade
```

Código A.1: Actualizar el sistema.

2. Instalar Docker:

```
1 sudo apt install docker.io
2 sudo systemctl enable docker
3 sudo systemctl start docker
```

Código A.2: Instalar Docker.

3. Verificar que Docker está instalado correctamente:

```
1 docker --version
```

Código A.3: Verificar instalación de Docker.

4. Instalar Docker Compose:

```
1 sudo apt install docker-compose
```

Código A.4: Instalar Docker Compose.

5. Verificar la instalación:

```
1 docker-compose --version
```

Código A.5: Verificar instalación de Docker Compose.

A.3 Instalación en Windows

Para instalar Docker y Docker Compose en Windows, se recomienda utilizar Docker Desktop, que incluye ambas herramientas de forma integrada.

1. Acceder al sitio oficial: <https://www.docker.com/products/docker-desktop/>
2. Descargar el instalador correspondiente para Windows.
3. Ejecutar el instalador y seguir el asistente de instalación.
4. Reiniciar el sistema si es necesario.
5. Verificar que Docker y Docker Compose estén correctamente instalados desde la terminal de Windows (PowerShell o CMD):

```
1 docker --version
2 docker-compose --version
```

Código A.6: Verificar instalación de Docker y Docker Compose.

Nota: Docker Desktop requiere que la virtualización esté habilitada en la BIOS del sistema. También es necesario contar con Windows 10 o superior.

A.4 Descripción del archivo docker-compose.yml

El archivo `docker-compose.yml` permite definir y configurar el entorno de ejecución del proyecto utilizando contenedores de Docker. A continuación, se presenta su contenido y una explicación de cada uno de sus elementos:

```
1  version: "3.8"
2  services:
3    data-analysis:
4      image: python:3.13-bookworm
5      container_name: data-analysis
6      tty: true
7      stdin_open: true
8      volumes:
9        - ./:/app
10     working_dir: /app
11     environment:
12       - PYTHONPATH=/app
13       - DB_HOST=postgres
14       - DB_PORT=5432
15       - DB_NAME=data_analysis
16       - DB_USER=postgres
17       - DB_PASSWORD=postgres123
18     command: >
19       sh -c "
20       pip install --no-cache-dir requirements.txt &&
21       echo 'Esperando a que PostgreSQL este listo...' &&
22       until pg_isready -h postgres -p 5432 -U postgres; do
23         echo 'PostgreSQL no esta listo - esperando...'
24         sleep 2
25       done &&
26       echo 'PostgreSQL esta listo!' &&
27       tail -f /dev/null
28     "
29     depends_on:
30       postgres:
31         condition: service_healthy
32     networks:
33       - data-network
34     postgres:
35       image: postgres:15-alpine
36       container_name: postgres-db
37       restart: always
38       environment:
39         POSTGRES_DB: data_analysis
40         POSTGRES_USER: postgres
41         POSTGRES_PASSWORD: postgres123
42       ports:
43         - "5432:5432"
44       volumes:
```

```
45      - postgres_data:/var/lib/postgresql/data
46
47  networks:
48    - data-network
49
50    healthcheck:
51      test: ["CMD-SHELL", "pg_isready -U postgres"]
52      interval: 10s
53      timeout: 5s
54      retries: 5
55      start_period: 30s
56
57    adminer:
58      image: adminer:latest
59      container_name: adminer
60      restart: always
61
62    ports:
63      - "8080:8080"
64
65    depends_on:
66      - postgres
67
68    networks:
69      - data-network
70
71    volumes:
72      postgres_data:
73        networks:
74          data-network:
75        driver: bridge
```

Código A.7: Archivo docker-compose.yml

A continuación se explica el propósito de cada sección:

- **version: "3.8"** Define la versión del esquema de Docker Compose utilizado. La versión 3.8 es compatible con la mayoría de las características modernas de Docker.
- **services** Define tres servicios que componen la aplicación: **data-analysis**, **postgres** y **adminer**.
- **data-analysis** Servicio principal que contiene la aplicación de análisis de datos:
 - **image: python:3.13-bookworm**: Utiliza una imagen oficial de Python 3.13 basada en Debian Bookworm.
 - **container_name: data-analysis**: Asigna un nombre personalizado al contenedor.
 - **tty: true** y **stdin_open: true**: Habilitan la interacción con el terminal del contenedor.
 - **volumes**: Monta el directorio actual del proyecto como **/app** dentro del contenedor.
 - **working_dir: /app**: Establece el directorio de trabajo dentro del contenedor.
 - **environment**: Define variables de entorno incluyendo **PYTHONPATH** y las credenciales de conexión a la base de datos PostgreSQL.

- **command:** Instala las dependencias, espera a que PostgreSQL esté disponible usando `pg_isready`, y mantiene el contenedor activo.
 - **depends_on:** Especifica que este servicio depende de que PostgreSQL esté saludable antes de iniciarse.
 - **networks:** Conecta el contenedor a la red `data-network`.
- **postgres** Servicio de base de datos PostgreSQL:
 - **image:** `postgres:15-alpine`: Utiliza la imagen oficial de PostgreSQL 15 basada en Alpine Linux.
 - **container_name:** `postgres-db`: Nombre del contenedor de la base de datos.
 - **restart: always**: Configura el contenedor para reiniciarse automáticamente en caso de fallo.
 - **environment**: Define las variables de entorno para la configuración inicial de PostgreSQL.
 - **ports**: Expone el puerto 5432 para permitir conexiones externas a la base de datos.
 - **volumes**: Crea un volumen persistente para almacenar los datos de PostgreSQL.
 - **healthcheck**: Configura verificaciones de salud para determinar cuándo PostgreSQL está listo.
 - **adminer** Interfaz web para administración de la base de datos:
 - **image:** `adminer:latest`: Utiliza la imagen oficial más reciente de Adminer.
 - **ports**: Expone el puerto 8080 para acceder a la interfaz web.
 - **depends_on**: Especifica dependencia del servicio PostgreSQL.
 - **volumes : postgres_data** Declara un volumen persistente para almacenar los datos de PostgreSQL, garantizando que los datos persistan entre reinicios de contenedores.
 - **networks : data-network** Define una red bridge personalizada que permite la comunicación entre los contenedores del proyecto.

Esta configuración crea un entorno completo de desarrollo que incluye la aplicación de análisis de datos, una base de datos PostgreSQL y una herramienta de administración web, todos interconectados y fácilmente replicables en cualquier sistema que tenga Docker instalado.

A.5 Scripts de control del contenedor

Para facilitar el manejo del contenedor durante el desarrollo del proyecto, se han creado tres scripts auxiliares en Bash que automatizan las operaciones más comunes: iniciar, reiniciar y detener el contenedor.

A.5.1 start_container.sh

Este script verifica si el contenedor `data-analysis` ya se encuentra en ejecución. En caso de que no esté activo, lo inicia utilizando `docker-compose up -d`. Posteriormente, ejecuta el archivo `main.py` dentro del contenedor.

```

1 #!/bin/bash
2
3 if ! docker ps --filter "name=/data-analysis$" --filter "
4     status=running" | grep -q data-analysis; then
5     echo "Contenedor no está corriendo. Levantando con docker-
6         compose..."
7     docker-compose up -d
8     echo "Esperando que se instalen las dependencias..."
9     while ! docker exec data-analysis pip show colorama &> /dev
10        /null; do
11            sleep 2
12        done
13        echo "Dependencias instaladas correctamente."
14    else
15        echo "Contenedor ya está corriendo. Usando instancia_
16            existente."
17    fi
18
19 echo "Ejecutando script..."
20 docker exec -it data-analysis python3 /app/src/main.py

```

Código A.8: Script para iniciar el contenedor.

A.5.2 restart_container.sh

Este script reinicia completamente el contenedor (equivalente a detenerlo y volverlo a levantar), lo cual resulta útil cuando se han modificado archivos como `requirements.txt` o `setup.py`. Tras reiniciar, vuelve a ejecutar el archivo principal del proyecto.

```

1 #!/bin/bash
2 docker restart data-analysis
3 sleep 2
4 docker exec -it data-analysis python3 /app/src/main.py

```

Código A.9: Script para reiniciar el contenedor.

A.5.3 stop_container.sh

Este script detiene y elimina el contenedor junto con los volúmenes asociados. Debe utilizarse con precaución, ya que elimina todas las dependencias instaladas en el entorno del contenedor. Solo es necesario en casos donde se requiere limpiar completamente el entorno.

```
1 #!/bin/bash  
2 docker-compose down --volumes
```

Código A.10: Script para detener y eliminar el contenedor y sus volúmenes.

A.6 Proceso de uso y desarrollo del contenedor

A continuación se describe el flujo recomendado para desarrollar y ejecutar el sistema dentro del contenedor de Docker:

1. Verifique que Docker y Docker Compose están instalados (Apéndice A.6).

Nota para usuarios de Windows: Si se utiliza Windows como sistema operativo, se deben usar los archivos con extensión **.bat** en lugar de **.sh**, y deben ser ejecutados desde la terminal de Windows (por ejemplo, CMD o PowerShell).

2. Asigne permisos de ejecución a los scripts:

```
1 chmod +x start_container.sh restart_container.sh  
      stop_container.sh
```

Código A.11: Dar permisos de ejecución a los scripts.

3. Para iniciar el contenedor y ejecutar el proyecto con los cambios más recientes del código fuente:

```
1 ./start_container.sh
```

Código A.12: Iniciar contenedor y ejecutar el proyecto.

4. Si se realizan cambios en las dependencias o archivos de configuración del entorno (como **requirements.txt**), utilice:

```
1 ./restart_container.sh
```

Código A.13: Reiniciar el contenedor completamente.

5. Para detener el contenedor y eliminar todos los volúmenes asociados:

```
1 ./stop_container.sh
```

Código A.14: Eliminar el contenedor y limpiar el entorno.

Este conjunto de scripts permite un desarrollo ágil dentro del contenedor, ya que los cambios realizados en el código fuente local se reflejan de inmediato gracias al uso de **volumes**. Además, se reduce la necesidad de ejecutar manualmente comandos repetitivos, facilitando el trabajo del usuario final y asegurando la correcta ejecución del proyecto.

Apéndice B

Scripts para la caracterización de datos

En la sección 3 se describen los pasos del proceso de caracterización de datos de trayectorias individuales. En este anexo se presentan los scripts utilizados para llevar a cabo dicho proceso.

```
1 import dask.dataframe as dd
2 import sys
3
4 print("Exploracion\u00f3n inicial\u00d3n de\u00d3n datos\u00f3n con\u00f3n Dask\n")
5
6 if len(sys.argv) < 2:
7     print("Error:\u00d3n Debe\u00f3n especificar\u00f3n un\u00f3n archivo\u00f3n CSV")
8     sys.exit(1)
9
10 ruta_archivo = sys.argv[1]
11
12 ddf = dd.read_csv(
13     ruta_archivo,
14     encoding="utf-8",
15     sep=",",
16     dtype="object",
17 )
18
19 columnas = ddf.columns.tolist()
20
21 print("Columnas\u00f3n 2\u00f3n ejemplos\u00f3n por\u00f3n cada\u00f3n una:\n")
22 for col in columnas:
23     ejemplos = ddf[col].head(2).values.tolist()
24     print(f"-{col}: {ejemplos}")
25
26 input("Presiona\u00f3n Enter\u00f3n para\u00f3n continuar...\")
```

Código B.1: csv_glance.py, exploración inicial del conjunto de datos.

```
1 import dask.dataframe as dd
2 import sys
3 import os
4
5 def contar_registros(ruta_archivo):
6
7     columnas_usar = ["record_id"]
8     try:
9         print(f"\nCargando archivo {ruta_archivo}...")
10        ddf = dd.read_csv(
11            ruta_archivo,
12            usecols=columnas_usar,
13            sep=",",
14            dtype={"record_id": "str"},
15            blocksize="256MB",
16        )
17
18        print("Contando registros (pacienza para archivos grandes)...")
19        total_registros = ddf.shape[0].compute()
20
21        print("\nAnalisis completado:")
22        print(f"Archivo analizado: {ruta_archivo}")
23        print(f"Total de registros: {total_registros:,}")
24
25    except Exception as e:
26        print(f"\nOcurrio un error inesperado: {str(e)}")
27
28 if __name__ == "__main__":
29     print("== Contador de registros en archivos CSV grandes ==")
30
31     if len(sys.argv) < 2:
32         print("Uso: python csv_count_registers.py < nombre_del_archivo.csv >")
33         sys.exit(1)
34
35     archivo = sys.argv[1]
36     contar_registros(archivo)
```

Código B.2: csv_count_registers.py, conteo de registros en el conjunto de datos.

```
1 import dask.dataframe as dd
2
3 columnas_deseadas = [
4     'identifier',
5     'timestamp',
6     'device_lat',
7     'device_lon',
8     'device_horizontal_accuracy',
9     'record_id',
10    'time_zone_name'
11 ]
12
13 df = dd.read_csv('Mobility_Data.csv', usecols=
14     columnas_deseadas)
15 df.to_csv('Mobility_Data_Slim.csv', index=False, single_file=
16     True, encoding='utf-8-sig')
```

Código B.3: remove_columns.py, eliminación de campos innecesarios en el conjunto de datos.

```
1  import pandas as pd
2  from tqdm import tqdm
3  import os
4  import sys
5  from src.menus.menu import MainMenu
6  def main():
7      print("\n" + "*"*50)
8      print("EXTRACTOR DE VALORES UNICOS DE COLUMNAS CSV")
9      print("*"*50 + "\n")
10
11     if len(sys.argv) < 2:
12         print("Uso: python extract_unique.py <archivo.csv>")
13         sys.exit(1)
14
15     csv_file = sys.argv[1]
16
17     if not os.path.exists(csv_file):
18         print(f"Error: El archivo '{csv_file}' no existe.")
19         sys.exit(1)
20
21     chunk_size = 1_000_000
22
23     try:
24         available_columns = pd.read_csv(csv_file, nrows=0).
25                         columns.tolist()
26     except Exception as e:
27         print(f"Error leyendo el archivo: {e}")
28         sys.exit(1)
29
30     try:
31         selected_index = MainMenu.display_available_columns(
32                         available_columns)
33         target_column = available_columns[selected_index]
34     except (ValueError, IndexError):
35         print("Seleccion invalida.")
36         sys.exit(1)
37     except Exception as e:
38         print(f"Error inesperado al seleccionar columna: {e}")
39         sys.exit(1)
40
41     safe_column_name = target_column.replace(" ", "_").
42                         replace("/", "_")
43     output_file = f"valores_unicos_{safe_column_name}.txt"
44
45     unique_values = set()
46     print(f"\nProcesando columna: {target_column}\n")
47
48     try:
49         for chunk in tqdm(pd.read_csv(csv_file, usecols=[
```

```

    target_column], chunksize=chunk_size)):
        unique_values.update(chunk[target_column].dropna()
            ().astype(str))
    except Exception as e:
        print(f"Error durante el procesamiento: {e}")
        sys.exit(1)
51
52     try:
53         numeric_values = sorted([float(v) for v in
54             unique_values])
55         is_numeric = True
56     except ValueError:
57         is_numeric = False
58
59     try:
60         with open(output_file, "w", encoding="utf-8") as f:
61             if is_numeric:
62                 min_val = numeric_values[0]
63                 max_val = numeric_values[-1]
64                 f.write(f"# Rango de valores: {min_val}-{max_val}\n")
65                 f.write("\n".join(str(v) for v in
66                     numeric_values))
67             else:
68                 sorted_values = sorted(unique_values)
69                 f.write("# Rango de valores: No numerico\n")
70                 f.write("\n".join(sorted_values))
71     except Exception as e:
72         print(f"Error guardando los resultados: {e}")
73         sys.exit(1)
74
75     print(f"\nSe encontraron {len(unique_values)} valores únicos.")
76     print(f"Resultados guardados en: {output_file}")
77
78     print("\nMuestra de valores únicos (primeros 10):")
79     print("\n".join(sorted(unique_values)[:10]))
80
81     if __name__ == "__main__":
82         main()

```

Código B.4: unique_values.py, obtención de valores únicos de la columna 'device_horizontal_accuracy'.

```
1  import os
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import sys
6  from tqdm import tqdm
7
8  def classify_tech(valor):
9      if 1 <= valor <= 20:
10          return 'GPS\u201dSatelital'
11      elif 5 <= valor <= 50:
12          return 'A-GPS\u201d(Asistido\u201dpor\u201dred)'
13      elif 20 <= valor <= 500:
14          return 'Triangulacion\u201dWiFi/Redes\u201dMoviles'
15      else:
16          return 'Fuera\u201dde\u201drango'
17
18  def format_count(count):
19      if count >= 1_000_000:
20          return f"{count/1_000_000:.1f}M"
21      elif count >= 1_000:
22          return f"{count/1_000:.1f}K"
23      return str(count)
24
25  def main():
26      if len(sys.argv) < 2:
27          print("Error:\u201dDebe\u201despecificar\u201dun\u201darchivo\u201dCSV\u201dcomo\u201dargumento")
28          sys.exit(1)
29
30      csv_file = sys.argv[1]
31      filename = os.path.splitext(os.path.basename(csv_file))
32      column = "device_horizontal_accuracy"
33      bins = 100
34
35      print(f"\nIniciando\u201dprocesamiento\u201ddel\u201darchivo:\u201d{csv_file}\u201d")
36      print(f"Columna\u201danalizada:\u201d{column}\u201d")
37
38      os.makedirs("img", exist_ok=True)
39      print("Directorio\u201d'img'\u201dverificado/creado")
40
41      print("\nProcesando\u201ddatos\u201dy\u201dclasificando\u201dtecnologias...")
42      frequency = pd.Series(dtype=float)
43      tech_counts = {
44          'GPS\u201dSatelital': 0,
45          'A-GPS\u201d(Asistido\u201dpor\u201dred)': 0,
46          'Triangulacion\u201dWiFi/Redes\u201dMoviles': 0,
```

```
47         'Fuera_de_rango': 0
48     }
49
50     total_row = sum(1 for _ in pd.read_csv(csv_file,
51                     usecols=[column], chunksize=1_000_000))
52
53     with tqdm(total=total_row, unit='M_rows') as pbar:
54         for chunk in pd.read_csv(csv_file, usecols=[column],
55             chunksize=1_000_000):
56             chunk_clean = chunk[column].dropna()
57
58             for valor in chunk_clean:
59                 tech = classify_tech(valor)
60                 tech_counts[tech] += 1
61
62                 counts = chunk_clean.value_counts()
63                 if not frequency.empty or not counts.empty:
64                     frequency = pd.concat([frequency, counts],
65                         axis=0).groupby(level=0).sum()
66                     pbar.update(1)
67
68             total = sum(tech_counts.values())
69             percentage = {k: (v/total)*100 for k, v in tech_counts.
70                           items()}
71
72             print("\nGenerando histograma con estadísticas...")
73             counts, edges = np.histogram(frequency.index, bins=bins,
74                 weights=frequency.values)
75
76             plt.figure(figsize=(14, 8))
77             plt.bar(edges[:-1], counts, width=np.diff(edges), align
78                     ='edge', edgecolor='black', alpha=0.7)
79
80             plt.axvline(x=20, color='r', linestyle='--', alpha=0.5)
81             plt.axvline(x=50, color='g', linestyle='--', alpha=0.5)
82             plt.axvline(x=200, color='b', linestyle='--', alpha
83                     =0.5)
84
85             gps_str = f"GPS_Satelital:{percentage['GPS_Satelital
86                 ']:.2f}\n({format_count(tech_counts['GPS_Satelital
87                 '])})\n"
88             agps_str = f"A-GPS:{percentage['A-GPS_(Asistido_por_
89                 red)']:.2f}\n({format_count(tech_counts['A-GPS_(
90                 Asistido_por_red)'])})\n"
91             wifi_str = f"WiFi/Redes:{percentage['Triangulacion_
92                 WiFi/Redes_Moviles']:.2f}\n({format_count(
93                 tech_counts['Triangulacion_WIFI/Redes_Moviles'])})\n"
94
95             plt.text(10, max(counts)*0.9, gps_str, ha='center',
96                     color='r', fontsize=10,
```

```
83         bbox=dict(facecolor='white', alpha=0.8,
84             edgecolor='r'))
85     plt.text(40, max(counts)*0.8, agps_str, ha='center',
86             color='g', fontsize=10,
87             bbox=dict(facecolor='white', alpha=0.8,
88                 edgecolor='g'))
89
90     plt.title(f"Distribucion de precisiones de {column}\n"
91               f"nArchivo:{filename}", fontsize=14)
92     plt.xlabel(f"Valores de {column} (metros)", fontsize
93                =12)
94     plt.ylabel("Frecuencia (Millones)", fontsize=12)
95     plt.xticks(edges[::5], rotation=45)
96     plt.grid(axis='y', linestyle='--')
97     plt.tight_layout()
98
99     output_path = os.path.join("img", f"histograma_{column}_"
100                                f"{filename}.png")
101    plt.savefig(output_path, dpi=300, bbox_inches='tight')
102    plt.close()
103
104   print("\n==== DISTRIBUCION DE TECNOLOGIAS DE "
105         "GEOLOCALIZACION ===")
106   for tech, count in tech_counts.items():
107       print(f'{tech}:{count:,} registros ({percentage['
108           f'tech]:.2f}%)')
109
110   print(f'\nHistograma generado exitosamente')
111   print(f'Archivo guardado en: {output_path}')
112   print(f'Total registros analizados: {total:,}\n')
113
114   if __name__ == "__main__":
115       main()
```

Código B.5: accuracy_histogram.py, creación de un histograma de frecuencias de la columna 'device_horizontal_accuracy'.

```
1  import os
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import numpy as np
5  from collections import Counter
6  import sys
7  from tqdm import tqdm
8  import math
9
10 def format_count(count):
11     if count >= 1_000_000:
12         return f"{count/1_000_000:.1f}M"
13     elif count >= 1_000:
14         return f"{count/1_000:.1f}K"
15     return str(count)
16
17 def main():
18     if len(sys.argv) < 2:
19         print("Error: Debe especificar un archivo CSV como argumento")
20         sys.exit(1)
21
22     csv_file = sys.argv[1]
23     filename = os.path.splitext(os.path.basename(csv_file))[0]
24     column = "identifier"
25     chunksize = 1_000_000
26
27     print(f"\nIniciando procesamiento del archivo: {csv_file}")
28     print(f"Columna analizada: {column}")
29     os.makedirs("img", exist_ok=True)
30     print("Directorio 'img' verificado/creado")
31
32     print("\nProcesando datos y contando frecuencias...")
33     counter = Counter()
34
35     total_chunks = sum(1 for _ in pd.read_csv(csv_file,
36                         usecols=[column], chunksize=chunksize))
37
38     with tqdm(total=total_chunks, unit='chunk') as pbar:
39         for chunk in pd.read_csv(csv_file, usecols=[column],
40                               chunksize=chunksize):
41             counter.update(chunk[column].dropna().astype(str))
42             pbar.update(1)
43
44     frecuency = pd.Series(counter)
45     total_unique_values = len(frecuency)
46     max_freq = frecuency.max()
```

```
45      print(f"Datos procesados correctamente")
46      print(f"Total de valores Unicos:{total_unique_values
47          : , }")
48      print(f"Frecuencia maxima:{max_freq:, }")
49
50      bins = [0] + [10**i for i in range(0, int(np.log10(
51          max_freq)) + 2)]
52      group_freq = pd.cut(frecuency, bins=bins, right=False).
53          value_counts().sort_index()
54
55      total_occurrence = frecuency.sum()
56      percentage_per_range = (group_freq /
57          total_unique_values * 100).round(2)
58
59      print("\nGenerando histograma con estadisticas...")
60      plt.figure(figsize=(16, 9))
61      ax = group_freq.plot(kind='bar', logy=True, alpha=0.7,
62          edgecolor='black')
63
64      formated_labels = []
65      for interval in group_freq.index.categories:
66          left = int(interval.left)
67          right = int(interval.right - 1)
68          formated_labels.append(f"{left}-{right}" if left
69              != right else f"{left}")
70
71      plt.xticks(range(len(formated_labels)),
72          formated_labels, rotation=45, ha='right')
73
74      plt.title(f"Histograma de Frecuencias de
75          Identificadores\nArchivo:{filename}", fontsize=16,
76          pad=20)
77      plt.xlabel("Rango de Frecuencia", fontsize=14)
78      plt.ylabel("Cantidad de Valores Unicos(log)", fontsize
79          =14)
80      plt.grid(True, which="both", ls="--", axis='y')
81
82      stats_text = (
83          f"Total_valores_unicos:{format_count(
84              total_unique_values)}\n"
85          f"Total_ocurrencias:{format_count(total_occurrence)
86              }\n"
87          f"Frecuencia_maxima:{format_count(max_freq)}"
88      )
89      plt.annotate(stats_text,
90          xy=(0.95, 0.95),
91          xycoords='axes fraction',
92          fontsize=15,
93          ha='right',
94          va='top',
```

```

84         bbox=dict(boxstyle='round', facecolor='
85             white', alpha=0.9))
86
87     max_val = group_freq.max()
88     min_y = 0.9
89
90     for i, (count, porcent) in enumerate(zip(group_freq.
91         values, percentage_per_range.values)):
92         if count > 0:
93             y_pos = count * 1.1 if count * 1.1 > min_y else
94                 min_y * 1.2
95
96             text = f'{porcent}%\n{format_count(count)}'
97
98             ax.text(
99                 i, y_pos, text,
100                 ha='center', va='bottom',
101                 fontsize=15,
102                 fontweight='bold',
103                 bbox=dict(
104                     facecolor='white',
105                     alpha=0.85,
106                     edgecolor='lightgray',
107                     boxstyle='round', pad=0.3
108                 )
109             )
110
111     output_path = os.path.join("img", f"histograma_{column}
112         _{filename}.png")
113     plt.tight_layout()
114     plt.savefig(output_path, dpi=300, bbox_inches='tight')
115     plt.close()
116
117     print("\n==== DISTRIBUCION DE FRECUENCIAS ====")
118     for i, (intervalo, count) in enumerate(group_freq.items
119         ()):
120         print(f'Rango {formatted_labels[i]}: {count},')
121
122         print(f'\nHistograma generado exitosamente')
123         print(f'Archivo guardado en: {output_path}')
124         print(f'Total ocurrencias analizadas: {total_occurrence
125             :,}\n')
126
127     if __name__ == "__main__":
128         main()

```

Código B.6: identifier_histogram.py, creación de un histograma de frecuencias de la columna 'identifier'.

```
1 import os
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import numpy as np
5 from collections import Counter
6 import sys
7 from tqdm import tqdm
8
9 def format_count(count):
10     if count >= 1_000_000:
11         return f"{count/1_000_000:.1f}M"
12     elif count >= 1_000:
13         return f"{count/1_000:.1f}K"
14     return str(count)
15
16 def create_histogram(data, bins, title, filename, color='skyblue', log_scale=False):
17     grouped = pd.cut(data, bins=bins, right=False).
18         value_counts().sort_index()
19     total_values = len(data)
20     max_count = grouped.max()
21
22     plt.figure(figsize=(14, 8))
23     ax = grouped.plot(kind='bar', color=color, edgecolor='black', alpha=0.7, logy=log_scale)
24
25     bin_labels = []
26     for interval in grouped.index.categories:
27         left = int(interval.left)
28         right = int(interval.right)
29         bin_labels.append(f'{left}-{right-1}' if right-left
30                           > 1 else str(left))
31
32         plt.xticks(range(len(bin_labels)), bin_labels, rotation
33                     =45, ha='right')
34         plt.title(f'{title}\nTotal_valores_unicos:{format_count(total_values)}', fontsize=14, pad=20)
35         plt.xlabel("Rango_de_repeticiones", fontsize=12)
36         plt.ylabel("Cantidad_de_valores_unicos" + ("_(log)" if
37             log_scale else ""), fontsize=12)
38         plt.grid(True, which="both", ls="--", axis='y')
39
40         min_y = 0.9
41         for i, (count, interval) in enumerate(zip(grouped.
42             values, grouped.index)):
43             if count > 0:
44                 percentage = (count / total_values) * 100
45                 y_pos = count * 1.1 if count * 1.1 > min_y else
46                         min_y * 1.2
47                 text = f'{percentage:.2f}%\n{format_count(
```

```
        count)})"

42
43     ax.text(i, y_pos, text,
44             ha='center', va='bottom',
45             fontsize=15, fontweight='bold',
46             bbox=dict(facecolor='white', alpha=0.8,
47                       edgecolor='lightgray', boxstyle='round',
48                       pad=0.2))

49
50     output_path = os.path.join("img", filename)
51     plt.tight_layout()
52     plt.savefig(output_path, dpi=300, bbox_inches='tight')
53     plt.close()
54     return output_path

55 def main():
56     if len(sys.argv) < 2:
57         print("Error: Debe especificar un archivo CSV")
58         sys.exit(1)

59     csv_file = sys.argv[1]
60     filename_base = os.path.splitext(os.path.basename(
61         csv_file))[0]
62     column = "identifier"
63     chunksize = 1_000_000
64     os.makedirs("img", exist_ok=True)

65     print(f"\nIniciando análisis de: {csv_file}")
66     print(f"Columna analizada: {column}")

67     print("\nContando frecuencias...")
68     counter = Counter()

69
70     with tqdm(desc="Contando filas totales", unit='filas'
71               ) as pbar:
72         total_rows = 0
73         for chunk in pd.read_csv(csv_file, usecols=[column
74             ], chunksize=chunksize):
75             total_rows += len(chunk)
76             pbar.update(len(chunk))

77     with tqdm(total=total_rows, desc="Procesando datos",
78               unit='filas') as pbar:
79         for chunk in pd.read_csv(csv_file, usecols=[column
80             ], chunksize=chunksize):
81             counter.update(chunk[column].dropna().astype(
82                             str))
83             pbar.update(len(chunk))

84     frequencies = pd.Series(counter)
85     total_unique = len(frequencies)
```

```

84     print(f"\nDatos procesados - Total valores unicos: {format_count(total_unique)}")
85
86     print("\nClasificando frecuencias...")
87     with tqdm(total=4, desc="Progreso") as pbar:
88         low_freq = frequencies[(frequencies >= 1) & (frequencies <= 99)]
89         pbar.update(1)
90         mid_freq = frequencies[(frequencies >= 100) & (frequencies <= 1000)]
91         pbar.update(1)
92         high_freq = frequencies[(frequencies >= 1001) & (frequencies <= 10000)]
93         pbar.update(1)
94
95     low_bin = list(range(1, 100, 10)) + [100]
96     mid_bin = list(range(100, 1001, 100)) + [1001]
97     high_bin = list(range(1001, 10001, 1000)) + [10001]
98
99     print("\n====Resumen de frecuencias====")
100    print(f"\nRango 1-99 repeticiones:")
101    print(f"Valores unicos: {format_count(len(low_freq))} ({len(low_freq)}/total_unique:.1%})")
102
103   print(f"\nRango 100-1000 repeticiones:")
104   print(f"Valores unicos: {format_count(len(mid_freq))} ({len(mid_freq)}/total_unique:.1%})")
105
106   print(f"\nRango 1001-10000 repeticiones:")
107   print(f"Valores unicos: {format_count(len(high_freq))} ({len(high_freq)}/total_unique:.1%})")
108
109
110  print("\nGenerando graficos...")
111  with tqdm(total=3, desc="Progreso") as pbar:
112      low_path = create_histogram(
113          low_freq,
114          bins=low_bin,
115          title="Distribucion de Frecuencias (1-99 repeticiones)",
116          filename=f"histograma_1-99_{column}_{filename_base}.png",
117          color='#4C72B0',
118      )
119      pbar.update(1)
120
121      mid_path = create_histogram(
122          mid_freq,
123          bins=mid_bin,
124          title="Distribucion de Frecuencias (100-1000 repeticiones)",
```

```
125         filename=f"histograma_100-1k_{column}_{"  
126             filename_base}.png",  
127             color='#55A868',  
128             log_scale=True  
129     )  
130     pbar.update(1)  
131  
131     high_path = create_histogram(  
132         high_freq,  
133         bins=high_bin,  
134         title="Distribucion de Frecuencias (1001-10,000  
135             _repeticiones)",  
136         filename=f"histograma_1k-10k_{column}_{"  
137             filename_base}.png",  
138             color='#C44E52',  
139             log_scale=True  
140     )  
141     pbar.update(1)  
142  
142     print("\nGraficos generados exitosamente:")  
143     print(f"{low_path}")  
143     print(f"{mid_path}")  
144     print(f"{high_path}")  
145  
146     if __name__ == "__main__":  
147         main()
```

Código B.7: identifier.histogram_detailed.py, análisis de frecuencias de la columna 'identifier'.

```
1      import dask.dataframe as dd
2      import sys
3      import os
4
5      def delete_duplicates(input_file, output_file):
6
7          ddf = dd.read_csv(input_file)
8
9          print(f"\nProcesando archivo:{input_file}")
10         print(f"Numero inicial de registros:{len(ddf):,}")
11
12         ddf_deduplicate = ddf.drop_duplicates(
13             subset=['identifier', 'timestamp', 'device_lon',
14                     'device_lat'],
15             keep='first'
16         )
17
18         print(f"Numero de registros despues de eliminar
19               duplicados:{len(ddf_deduplicate):,}")
20
21         ddf_deduplicate.to_csv(
22             output_file,
23             index=False,
24             single_file=True
25         )
26
27         print(f"\nArchivo sin duplicados guardado en:{output_file}")
28
29     if __name__ == "__main__":
30         if len(sys.argv) < 2:
31             print("Error: Debe especificar un archivo CSV como
32                   argumento")
33             sys.exit(1)
34
35         input_csv = sys.argv[1]
36         base_name = os.path.splitext(input_csv)[0]
37         output_csv = f"{base_name}_DeDuplicate.csv"
38
39         delete_duplicates(input_csv, output_csv)
```

Código B.8: csv_deduplicate.py, eliminación de duplicados en el conjunto de datos.

```
1      import os
2      import pandas as pd
3      import matplotlib.pyplot as plt
4      import numpy as np
5      from collections import Counter
6      import sys
7      from tqdm import tqdm
8      import math
9      from datetime import datetime
10
11     def format_count(count):
12         if count >= 1_000_000:
13             return f"{count/1_000_000:.1f}M"
14         elif count >= 1_000:
15             return f"{count/1_000:.1f}K"
16         return str(count)
17
18     def main():
19         if len(sys.argv) < 2:
20             print("Error: Debe especificar un archivo CSV como argumento")
21             sys.exit(1)
22
23         csv_file = sys.argv[1]
24         filename = os.path.splitext(os.path.basename(csv_file))[0]
25         identifier_col = "identifier"
26         timestamp_col = "timestamp"
27         chunksize = 1_000_000
28
29         print(f"\nIniciando procesamiento del archivo: {csv_file}")
30         print(f"Columnas analizadas: {identifier_col} y {timestamp_col}")
31
32         os.makedirs("img/daily_histograms", exist_ok=True)
33         print("Directorios 'img/daily_histograms' verificados/creados")
34
35         print("\nProcesando datos y agrupando por dia...")
36
37         date_freqs = {}
38         max_freq = 0
39
40         total_chunks = sum(1 for _ in pd.read_csv(csv_file,
41                           usecols=[timestamp_col, identifier_col], chunksize=
42                           chunksize))
43
44         with tqdm(total=total_chunks, unit='chunk') as pbar:
```

```
44         for chunk in pd.read_csv(csv_file, usecols=[  
45             timestamp_col, identifier_col], chunksize=  
46             chunksize):  
47             try:  
48                 chunk['date'] = pd.to_datetime(  
49                     chunk[timestamp_col],  
50                     format='mixed',  
51                     errors='coerce'  
52                 ).dt.date  
53  
54                 chunk = chunk.dropna(subset=['date'])  
55  
56                 for date, group in chunk.groupby('date'):  
57                     if date not in date_freqs:  
58                         date_freqs[date] = Counter()  
59  
60                     date_freqs[date].update(group[  
61                         identifier_col].dropna().astype(str)  
62                     )  
63  
64                     current_max = date_freqs[date].  
65                     most_common(1)[0][1] if date_freqs[  
66                         date] else 0  
67                     if current_max > max_freq:  
68                         max_freq = current_max  
69                 except Exception as e:  
70                     print(f"\nError al procesando chunk: {str(e)}")  
71                     continue  
72                 finally:  
73                     pbar.update(1)  
74  
75             if not date_freqs:  
76                 print("\nError: No se encontraron datos validos  
77                     para procesar")  
78                 sys.exit(1)  
79  
80             bins = [0] + [10**i for i in range(0, int(np.log10(  
81                 max_freq)) + 2)] if max_freq > 0 else [0, 1]  
82  
83             print("\nGenerando histogramas por dia...")  
84  
85             for date, counter in tqdm(date_freqs.items(), total=len(  
86                 date_freqs), unit='dia'):  
87                 frecuency = pd.Series(counter)  
88                 total_unique_values = len(frecuency)  
89                 total_ocurrence = frecuency.sum()  
90  
91                 group_freq = pd.cut(frecuency, bins=bins, right=  
92                     False).value_counts().sort_index()  
93                 percentage_per_range = (group_freq /
```

```
84         total_unique_values * 100).round(2)
85
86     plt.figure(figsize=(16, 9))
87     ax = group_freq.plot(kind='bar', logy=True, alpha
88                           =0.7, edgecolor='black')
89
90     formatted_labels = []
91     for interval in group_freq.index.categories:
92         left = int(interval.left)
93         right = int(interval.right - 1)
94         formatted_labels.append(f'{left}-{right}' if
95                               left != right else f'{left}')
96
97     plt.xticks(range(len(formatted_labels)),
98                formatted_labels, rotation=45, ha='right')
99
100    date_str = date.strftime('%Y-%m-%d')
101    plt.title(f"Histograma de Frecuencias de
102              Identificadores\nArchivo:{filename} Fecha:{date_str}",
103              fontsize=16, pad=20)
104    plt.xlabel("Rango de Frecuencia", fontsize=14)
105    plt.ylabel("Cantidad de Valores Unicos(log)",
106               fontsize=14)
107    plt.grid(True, which="both", ls="--", axis='y')
108
109    stats_text = (
110        f"Total_valores_unicos:{(total_unique_values)
111         : ,}\n"
112        f"Total_ocurrencias:{(total_occurrence): ,}\n"
113        f"Frequencia_maxima:{(frequency.max()): ,}"
114    )
115    plt.annotate(stats_text,
116                 xy=(0.95, 0.95),
117                 xycoords='axes fraction',
118                 fontsize=15,
119                 ha='right',
120                 va='top',
121                 bbox=dict(boxstyle='round', facecolor=
122                           'white', alpha=0.9))
123
124    max_val = group_freq.max()
125    min_y = 0.9
126
127    for i, (count, porcent) in enumerate(zip(group_freq
128                                              .values, percentage_per_range.values)):
129        if count > 0:
130            y_pos = count * 1.1 if count * 1.1 > min_y
131            else min_y * 1.2
132            text = f"{porcent}\n{format_count(count)
133                  }"
134            ax.text(
```

```
123             i, y_pos, text,
124             ha='center', va='bottom',
125             fontsize=15,
126             fontweight='bold',
127             bbox=dict(
128                 facecolor='white',
129                 alpha=0.85,
130                 edgecolor='lightgray',
131                 boxstyle='round, pad=0.3',
132             )
133         )
134
135     output_path = os.path.join("img", "daily_histograms",
136                               f"histograma_{identifier_col}_{filename}_{date_str}.png")
137     plt.tight_layout()
138     plt.savefig(output_path, dpi=300, bbox_inches='tight')
139     plt.close()
140
141     print("\nHistogramas generados exitosamente")
142     print(f"Archivos guardados en: img/daily_histograms/")
143     print(f"Total días procesados: {len(date_freqs)}")
144     print(f"Frecuencia máxima global encontrada: {format_count(max_freq)}\n")
145
146 if __name__ == "__main__":
147     main()
```

Código B.9: identifier_histogram_daily.py, análisis de frecuencias de la columna 'identifier' por día.

```
1  import pandas as pd
2  import psycopg2
3  from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
4  import os
5  from sqlalchemy import create_engine, text
6  import logging
7
8  logging.basicConfig(level=logging.INFO, format='%(asctime)s
9      - %(levelname)s - %(message)s')
10 logger = logging.getLogger(__name__)
11
12 class MobilityDataLoader:
13     def __init__(self):
14         self.db_config = {
15             'host': os.getenv('DB_HOST', 'localhost'),
16             'port': os.getenv('DB_PORT', '5432'),
17             'user': os.getenv('DB_USER', 'postgres'),
18             'password': os.getenv('DB_PASSWORD', '
19                 postgres123'),
20             'default_db': os.getenv('DB_NAME', 'postgres')
21                 # DB por defecto para crear la nueva
22         }
23         self.target_db = 'trajectories'
24         self.csv_file = 'Mobility_Data_Slim_DeDuplicate.csv
25
26     def create_database(self):
27         try:
28             conn = psycopg2.connect(
29                 host=self.db_config['host'],
30                 port=self.db_config['port'],
31                 user=self.db_config['user'],
32                 password=self.db_config['password'],
33                 database=self.db_config['default_db']
34             )
35             conn.set_isolation_level(
36                 ISOLATION_LEVEL_AUTOCOMMIT)
37             cur = conn.cursor()
38             cur.execute("SELECT 1 FROM pg_catalog.
39                 pg_database WHERE datname=%s", (self.
40                 target_db,))
41             exists = cur.fetchone()

42             if not exists:
43                 cur.execute(f'CREATE DATABASE {self.
44                     target_db}')
45                 logger.info(f"Base de datos '{self.
46                     target_db}' creada exitosamente")
47             else:
48                 logger.info(f"Base de datos '{self.
```

```
                                target_db} , ya existe")
42
43         cur.close()
44         conn.close()
45
46     except Exception as e:
47         logger.error(f"Error al crear la base de datos:
48                         {e}")
49         raise
50
51     def analyze_csv_structure(self):
52         try:
53             if not os.path.exists(self.csv_file):
54                 raise FileNotFoundError(f"Archivo {self.
55                                         csv_file} no encontrado")
56
57             df_sample = pd.read_csv(self.csv_file, nrows=5)
58             logger.info(f"Estructura del CSV:")
59             logger.info(f"Columnas: {list(df_sample.columns
60                                         )}")
61             logger.info(f"Tipos de datos:")
62             for col, dtype in df_sample.dtypes.items():
63                 logger.info(f"{col}: {dtype}")
64
65             return df_sample
66
67
68     except Exception as e:
69         logger.error(f"Error al analizar CSV: {e}")
70         raise
71
72     def create_table_from_csv(self, df_sample):
73         try:
74             engine = create_engine(
75                 f"postgresql://{self.db_config['user']}:{self.
76                                         db_config['password']}@{self.
77                                         db_config['host']}:{self.db_config
78                                         ['port']}/{self.target_db}"
79             )
80             type_mapping = {
81                 'object': 'TEXT',
82                 'int64': 'BIGINT',
83                 'int32': 'INTEGER',
84                 'float64': 'DOUBLE PRECISION',
85                 'float32': 'REAL',
86                 'bool': 'BOOLEAN',
87                 'datetime64[ns]': 'TIMESTAMP'
88             }
89             columns_ddl = []
90             for col, dtype in df_sample.dtypes.items():
91                 pg_type = type_mapping.get(str(dtype), 'TEXT')
```

```
86         clean_col = col.lower().replace(' ', '_').
87             replace('-', '_').replace('.', '_')
88         columns_ddl.append(f"{{clean_col}}{pg_type}"')
89
90         create_table_sql = f"""
91             CREATE TABLE IF NOT EXISTS mobility_data(
92                 id SERIAL PRIMARY KEY,
93                 {', '.join(columns_ddl)},
94                 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
95             )"""
96
97         with engine.connect() as conn:
98             conn.execute(text("DROP TABLE IF EXISTS"
99                           " mobility_data"))
100            conn.execute(text(create_table_sql))
101            conn.commit()
102
103            logger.info("Tabla 'mobility_data' creada"
104                         " exitosamente")
105
106        return engine
107
108    except Exception as e:
109        logger.error(f"Error al crear la tabla: {e}")
110        raise
111
112    def load_csv_to_table(self, engine):
113        try:
114            logger.info(f"Leyendo archivo CSV: {self."
115                        "csv_file}")
116            df = pd.read_csv(self.csv_file)
117
118            # Limpieza de nombres de columnas
119            df.columns = [col.lower().replace(' ', '_').
120                          replace('-', '_').replace('.', '_')
121                          for col in df.columns]
122
123            logger.info(f"Cargando {len(df)} registros a la"
124                         " tabla...")
125
126            chunk_size = 1000
127            total_chunks = len(df) // chunk_size + (1 if
128                            len(df) % chunk_size else 0)
129
130            for i, chunk in enumerate(pd.read_csv(self."
131                            csv_file, chunksize=chunk_size)):
132                # Limpieza de nombres de columnas del chunk
133                chunk.columns = [col.lower().replace(' ', '_')
```

```
127         _').replace('-', '_').replace('.', '_')
128             for col in chunk.columns]
129
130             chunk.to_sql('mobility_data', engine,
131                         if_exists='append', index=False, method=
132                         'multi')
133             logger.info(f"Procesado chunk {i+1}/{total_chunks}")
134
135             logger.info("Datos cargados exitosamente")
136
137             with engine.connect() as conn:
138                 result = conn.execute(text("SELECT COUNT(*)
139                                 FROM mobility_data"))
140                 count = result.fetchone()[0]
141                 logger.info(f"Total de registros en la
142                             tabla: {count}")
143
144             except Exception as e:
145                 logger.error(f"Error al cargar datos: {e}")
146                 raise
147
148         def run(self):
149             try:
150                 logger.info("== Iniciando proceso de carga de
151                             datos de movilidad ==")
152
153                 self.create_database()
154                 df_sample = self.analyze_csv_structure()
155                 engine = self.create_table_from_csv(df_sample)
156                 self.load_csv_to_table(engine)
157
158                 logger.info("== Proceso completado
159                             exitosamente ==")
160
161             except Exception as e:
162                 logger.error(f"Error en el proceso: {e}")
163                 raise
164
165         if __name__ == "__main__":
166             loader = MobilityDataLoader()
167             loader.run()
```

Código B.10: migrate_csv_to_postgres.py, migración de datos desde un CSV a una base de datos PostgreSQL.

Referencias

- [1] Autor referencia 1
- [2] Autor referencia 2
- [3] Autor referencia 3