Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Unidad Iztapalapa

Caracterización de datos de trayectorias individuales

Presentado por:
Jorge Rafael Martínez Buenrostro

Asesora: Dra. Elizabeth Pérez Cortés

México, CDMX, a 30 de julio 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 detenenido 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 Figuras

Lista de Códigos

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

2.1 Requisitos

Docker: version 28.2.2, build e6534b4

Docker Compose: version 1.29.2, build unknown

2.1.1 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]

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

Código 2.1: Iniciar contenedor del proyecto.

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

```
./stop_container.sh   # Linux/Mac
.\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:

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

Código 2.3: Reiniciar contenedor del proyecto.

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

Capítulo 3

Caracterización de datos de trayectorias individuales

3.1 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.

3.1.1 Exploración inicial del conjunto de datos

Como primer paso en la caracterización, se realizó una exploración preliminar del conjunto de datos con el fin de comprender su estructura general. Para ello, se inspeccionaron las primeras dos filas, lo cual permitió 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']`

3.1.2 Dimensiones del conjunto de datos

Para verificar las dimensiones del conjunto de datos, se utilizó la bilbioteca Dask, que permite trabajar con grandes volúmenes de datos de manera eficiente. Junto con Python se usó el código en el Apéndice B.2. Como resultado ahora sabemos 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.

3.1.3 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 realizó 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 identificaron 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 optó 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.

3.1.4 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 generaron representaciones gráficas que permiten observar la distribución de los datos y facilitar la toma de decisiones. 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 utilizó 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 indicó que los valores oscilan entre 0.916 y 199.9, lo que permitió 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:

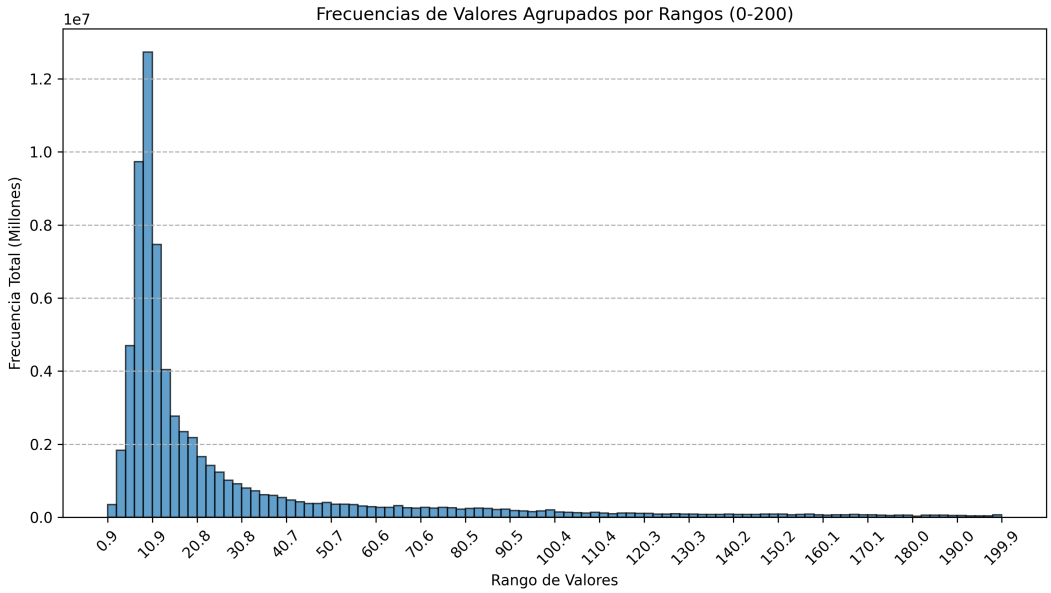


Figura 3.1: Frecuencias de valores agrupados por rangos (0–200 metros).

Como se observa en la Figura 3.4, la mayoría de los valores se concentran en el rango de 0 a 20 metros, lo cual es consistente con la precisión obtenida mediante GPS puro. Para reforzar esta observación, se realizó un conteo porcentual por tipo de tecnología de geolocalización:

- GPS puro (1–20 metros): 68.73 %.
- A-GPS (5–50 metros): 16.56 %.
- WiFi/red móvil (20–500 metros): 14.69 %.

La siguiente columna evaluada fue `identifier`, correspondiente al identificador único de cada dispositivo. Se analiza la frecuencia de aparición de estos valores, para lo cual se empleó un script que agrupa las repeticiones por rangos y grafica la cantidad de valores únicos usando escala logarítmica (ver Apéndice B.6).

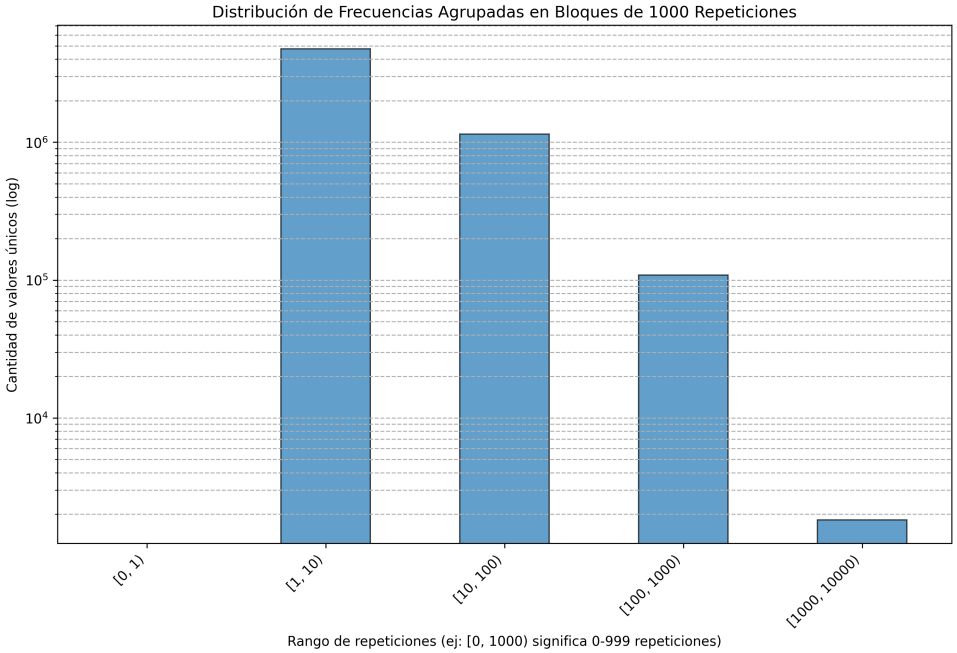


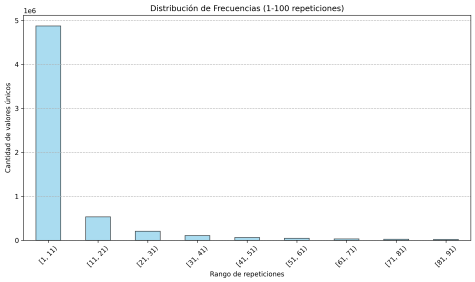
Figura 3.2: Distribución de frecuencias agrupadas en bloques de 1000 repeticiones.

La Figura 3.2 muestra que la mayoría de los identificadores tienen una baja frecuencia. Para un análisis más detallado, se usó el código del Apéndice B.7, el cual segmenta los datos en tres rangos: 1–100, 100–1000 y 1000–10000 repeticiones.

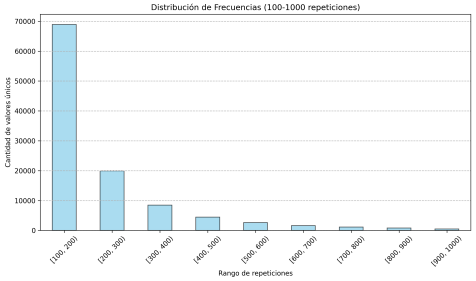
Además de los histogramas, el script genera un resumen con el número total de valores únicos por rango:

```
=== Resumen de frecuencias ===
**Rango 1 - 100 repeticiones**:
- Total de valores únicos: 5,912,437

**Rango 100 - 1000 repeticiones**:
- Total de valores únicos: 108,519

**Rango 1000 - 10000 repeticiones**:
- Total de valores únicos: 1,816
```

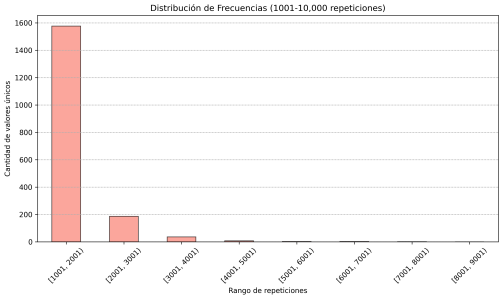
Esta información evidencia que la mayoría de los identificadores aparecen entre 1 y 100 veces, siendo predominante el subconjunto de 1 a 10 repeticiones.



(a) Histograma 1–100 repeticiones



(b) Histograma 100–1000 repeticiones



(c) Histograma 1001–10000 repeticiones

Figura 3.3: Comparación de histogramas por rangos de repeticiones.

Con base en esta información es necesario hacer un análisis más detallado de los identificadores. Para ello, se eliminan los registros que tienen duplicados, es decir, aquellos que tienen la misma información en las columnas `timestamp`, `device_lat`, `device_lon`. Después de aplicar este filtro se obtiene un total de **50,753,197**.

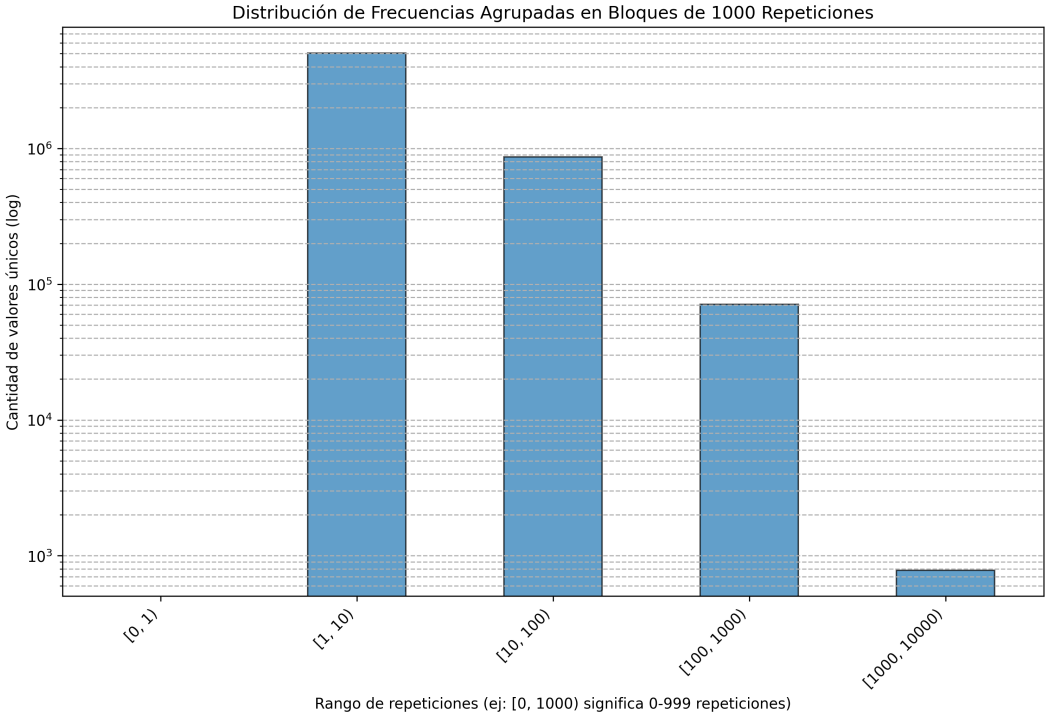


Figura 3.4: Frecuencias de valores agrupados por rangos (0–200 metros).

Apéndice A

Uso de Docker y Docker Compose

En la sección 2.1 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:

```
sudo apt install docker.io
sudo systemctl enable docker
sudo systemctl start docker
```

Código A.2: Instalar Docker.

3. Verificar que Docker está instalado correctamente:

```
docker --version
```

Código A.3: Verificar instalación de Docker.

4. Instalar Docker Compose:

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

Código A.4: Instalar Docker Compose.

5. Verificar la instalación:

```
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):

```
docker --version
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 un contenedor de Docker. A continuación, se presenta su contenido y una explicación de cada uno de sus elementos:

```
1  version: "3.8"
2
3  services:
4    data-analysis:
5      image: python:3.13-bookworm
6      container_name: data-analysis
7      runtime: nvidia
8      tty: true
9      stdin_open: true
10     volumes:
11       - ./:/app
12       - python-packages:/usr/local/lib/python3.13/site-packages
13     command: sh -c "pip␣install␣-r␣requirements.txt␣&&␣pip␣
           install␣-e␣.␣&&␣python3␣/app/src/main.py"
14     working_dir: /app
15     environment:
16       - PYTHONPATH=/app
17
18 volumes:
19   python-packages:
```

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` → `data-analysis`
  Se define un servicio llamado `data-analysis`, que representa el contenedor principal del proyecto.
- `image: python:3.13-bookworm`
  Utiliza una imagen oficial de Python 3.13 basada en Debian Bookworm como entorno base.
- `container_name: data-analysis`
  Asigna un nombre personalizado al contenedor para facilitar su identificación.
- `runtime: nvidia`
  Indica que el contenedor utilizará el runtime de NVIDIA para permitir acceso a la GPU. Requiere tener instalado `nvidia-docker`.
- `tty: true` y `stdin_open: true`
  Habilitan la interacción con el terminal del contenedor, lo que es útil para

ejecutar comandos manuales si es necesario.

- `volumes`
  - `./:/app`: Monta el directorio actual del proyecto como `/app` dentro del contenedor.
  - `python-packages:/usr/local/lib/python3.13/site-packages`: Crea un volumen persistente para las bibliotecas de Python instaladas.
- `command`
  Ejecuta una serie de comandos cuando el contenedor inicia: instala las dependencias del archivo `requirements.txt`, instala el proyecto en modo editable (`pip install -e .`) y ejecuta el archivo `main.py`.
- `working_dir: /app`
  Establece el directorio de trabajo dentro del contenedor como `/app`.
- `environment`
  Define la variable de entorno `PYTHONPATH` para que Python pueda encontrar correctamente los módulos dentro del proyecto.
- `volumes → python-packages`
  Declara un volumen persistente llamado `python-packages`, que se utiliza para almacenar los paquetes instalados sin perderlos entre reinicios del contenedor.

Este archivo permite que el entorno de desarrollo sea fácilmente replicable y ejecutable, sin necesidad de instalar manualmente dependencias o configurar rutas en el sistema anfitrión.

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.

```
#!/bin/bash

# Verifica si el contenedor esta corriendo
if ! docker ps --filter "name=^/data-analysis$" --filter "
    status=running" | grep -q data-analysis; then
    echo "Contenedor no esta corriendo. Levantando con docker-
        compose..."
    docker-compose up -d
    sleep 2
else
    echo "Contenedor ya esta corriendo. Usando instancia
        existente."
fi

# Ejecuta el script dentro del contenedor
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.

```
#!/bin/bash
docker restart data-analysis
sleep 2
# Ejecuta el script dentro del contenedor
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.

```
#!/bin/bash
docker-compose down -v
```

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:

```
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:

```
./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:

```
./restart_container.sh
```

Código A.13: Reiniciar el contenedor completamente.

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

```
./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 3.1 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␣inicial␣de␣datos␣con␣Dask\n")
5
6  if len(sys.argv) < 2:
7      print("Error:␣Debe␣especificar␣un␣archivo␣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␣y␣2␣ejemplos␣por␣cada␣una:\n")
22 for col in columnas:
23     ejemplos = ddf[col].head(2).values.tolist()
24     print(f"-␣{col}:␣{ejemplos}")
25
26 input("Presiona␣Enter␣para␣continuar...")
```

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

```
import dask.dataframe as dd
import sys
import os

def contar_registros(ruta_archivo):

    columnas_usar = ["record_id"]
    try:
        print(f"\nCargando archivo {ruta_archivo}...")
        ddf = dd.read_csv(
            ruta_archivo,
            usecols=columnas_usar,
            sep=",",
            dtype={"record_id": "str"},
            blocksize="256MB",
        )

        print("Contando registros (paciencia para archivos grandes)...")
        total_registros = ddf.shape[0].compute()

        print(f"\nAnalisis completado:")
        print(f"Archivo analizado: {ruta_archivo}")
        print(f"Total de registros: {total_registros:,}")

    except Exception as e:
        print(f"\nOcurrio un error inesperado: {str(e)}")

if __name__ == "__main__":
    print("=== Contador de registros en archivos CSV grandes ===")

    if len(sys.argv) < 2:
        print("Uso: python csv_count_registers.py <nombre_del_archivo.csv>")
        sys.exit(1)

    archivo = sys.argv[1]
    contar_registros(archivo)
```

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

```
import dask.dataframe as dd

columnas_deseadas = [
  'identifier',
  'timestamp',
  'device_lat',
  'device_lon',
  'device_horizontal_accuracy',
  'record_id',
  'time_zone_name'
]

df = dd.read_csv('Mobility_Data.csv', usecols=
    columnas_deseadas)

df.to_csv('Mobility_Data_Slim.csv', index=False, single_file=
    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).
               columns.tolist()
25     except Exception as e:
26         print(f"Error␣leyendo␣el␣archivo:␣{e}")
27         sys.exit(1)
28
29     try:
30         selected_index = MainMenu.display_available_columns(
               available_columns)
31         target_column = available_columns[selected_index]
32     except (ValueError, IndexError):
33         print("Seleccion␣invalida.")
34         sys.exit(1)
35     except Exception as e:
36         print(f"Error␣inesperado␣al␣seleccionar␣columna:␣{e}"
               )
37         sys.exit(1)
38
39     safe_column_name = target_column.replace("␣", "_").
          replace("/", "_")
40     output_file = f"valores_unicos_{safe_column_name}.txt"
41
42     unique_values = set()
43     print(f"\nProcesando␣columna:␣{target_column}\n")
44
45     try:
46         for chunk in tqdm(pd.read_csv(csv_file, usecols=[
```

```
            target_column], chunksize=chunk_size)):
47              unique_values.update(chunk[target_column].dropna
                    ().astype(str))
48      except Exception as e:
49          print(f"Error durante el procesamiento: {e}")
50          sys.exit(1)
51
52      try:
53          numeric_values = sorted([float(v) for v in
                unique_values])
54          is_numeric = True
55      except ValueError:
56          is_numeric = False
57
58      try:
59          with open(output_file, "w", encoding="utf-8") as f:
60              if is_numeric:
61                  min_val = numeric_values[0]
62                  max_val = numeric_values[-1]
63                  f.write(f"# Rango de valores: {min_val} - {
                        max_val}\n")
64                  f.write("\n".join(str(v) for v in
                        numeric_values))
65              else:
66                  sorted_values = sorted(unique_values)
67                  f.write("# Rango de valores: No numerico\n")
68                  f.write("\n".join(sorted_values))
69      except Exception as e:
70          print(f"Error guardando los resultados: {e}")
71          sys.exit(1)
72
73      print(f"\nSe encontraron {len(unique_values):,} valores
            unicos.")
74      print(f"Resultados guardados en: {output_file}")
75
76      print("\nMuestra de valores unicos (primeros 10):")
77      print("\n".join(sorted(unique_values)[:10]))
78
79  if __name__ == "__main__":
80      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
6  archivo_csv = "Mobility_Data_Slim.csv"
7  columna = "device_horizontal_accuracy"
8  bins = 100
9  os.makedirs("img", exist_ok=True)
10
11 frecuencias = pd.Series(dtype=float)
12 for chunk in pd.read_csv(archivo_csv, usecols=[columna],
       chunksize=1_000_000):
13   frecuencias = pd.concat([frecuencias, chunk[columna].
       value_counts()])
14
15 counts, edges = np.histogram(frecuencias.index, bins=bins,
       weights=frecuencias.values)
16
17 plt.figure(figsize=(12, 6))
18 plt.bar(edges[:-1], counts, width=np.diff(edges), align='edge
       ', edgecolor='black', alpha=0.7)
19 plt.title("Frecuencias de Valores Agrupados por Rangos
       (0-200)")
20 plt.xlabel("Rango de Valores")
21 plt.ylabel("Frecuencia Total (Millones)")
22 plt.xticks(edges[::5], rotation=45)
23 plt.grid(axis='y', linestyle='--')
24
25 output_path = "img/histograma_frecuencias_rangos.png"
26 plt.savefig(output_path, dpi=300, bbox_inches='tight')
27 plt.close()
```

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
7   archivo_csv = "Mobility_Data_Slim.csv"
8   columna = "identifier"
9   chunksize = 1_000_000
10  os.makedirs("img", exist_ok=True)
11
12  counter = Counter()
13  for chunk in pd.read_csv(archivo_csv, usecols=[columna],
        chunksize=chunksize):
14      counter.update(chunk[columna].dropna().astype(str))
15
16  frecuencias = pd.Series(counter)
17
18  max_freq = frecuencias.max()
19  bins = [0] + [10**i for i in range(0, int(np.log10(max_freq
        )) + 2)]  # Ej: [0, 1, 10, 100, 1000, ...]
20  frecuencias_agrupadas = pd.cut(frecuencias, bins=bins,
        right=False).value_counts().sort_index()
21
22  plt.figure(figsize=(12, 7))
23  frecuencias_agrupadas.plot(kind='bar', logy=True, alpha
        =0.7, edgecolor='black')
24
25  plt.xticks(rotation=45, ha='right')  # Rotar etiquetas para
        mejor legibilidad
26  plt.title("Distribucion␣de␣Frecuencias␣Agrupadas␣en␣Bloques
        ␣de␣1000␣Repeticiones")
27  plt.xlabel("Rango␣de␣repeticiones␣(ej:␣[0,␣1000)␣significa␣
        0-999␣repeticiones)")
28  plt.ylabel("Cantidad␣de␣valores␣unicos␣(log)")
29  plt.grid(True, which="both", ls="--", axis='y')
30
31  output_path = os.path.join("img", "
        histograma_frecuencias_agrupadas_1000.png")
32  plt.savefig(output_path, dpi=300, bbox_inches='tight')
33  plt.close()
```

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
7  archivo_csv = "Mobility_Data_Slim.csv"
8  columna = "identifier"
9  chunksize = 1_000_000
10 os.makedirs("img", exist_ok=True)
11
12 counter = Counter()
13 for chunk in pd.read_csv(archivo_csv, usecols=[columna],
       chunksize=chunksize):
14     counter.update(chunk[columna].dropna().astype(str))
15 frecuencias = pd.Series(counter)
16
17 frecuencias_bajas = frecuencias[(frecuencias >= 1) & (
       frecuencias <= 99)]
18 frecuencias_medias = frecuencias[(frecuencias >= 100) & (
       frecuencias <= 1000)]
19 frecuencias_altas = frecuencias[(frecuencias >= 1001) & (
       frecuencias <= 10000)]
20
21 bins_bajas = list(range(1, 100, 10))  # 1-99 en pasos de 10
22 bins_medias = list(range(100, 1001, 100))  # 100-1000 en
       pasos de 100
23 bins_altas = list(range(1001, 10001, 1000))  # 1001-10000
       en pasos de 1000
24
25 freciencias_bajas_agrupadas = pd.cut(frecuencias_bajas,
       bins=bins_bajas, right=False).value_counts().sort_index
       ()
26 frecuencias_medias_agrupadas = pd.cut(frecuencias_medias,
       bins=bins_medias, right=False).value_counts().sort_index
       ()
27 frecuencias_altas_agrupadas = pd.cut(frecuencias_altas,
       bins=bins_altas, right=False).value_counts().sort_index
       ()
28
29 print("\n=== Resumen de frecuencias ===")
30 print(f"\n**Rango 1-100 repeticiones**:")
31 print(f" - Total de valores unicos: {len(frecuencias_bajas)
       }")
32 print(f"\n**Rango 100-1000 repeticiones**:")
33 print(f" - Total de valores unicos: {len(frecuencias_medias
       )}")
34 print(f"\n**Rango 1000-10000 repeticiones**:")
35 print(f" - Total de valores unicos: {len(frecuencias_altas)
       }")
```

```
36
37
38      plt.figure(figsize=(10, 6))
39      freciencias_bajas_agrupadas.plot(kind='bar', color='skyblue
            ', edgecolor='black', alpha=0.7)
40      plt.title("Distribucion␣de␣Frecuencias␣(1-100␣repeticiones)
            ")
41      plt.xlabel("Rango␣de␣repeticiones")
42      plt.ylabel("Cantidad␣de␣valores␣unicos")
43      plt.xticks(rotation=45)
44      plt.grid(axis='y', linestyle='--')
45      plt.tight_layout()
46      plt.savefig(os.path.join("img", "histograma_1_100.png"),
            dpi=300)
47      plt.close()
48
49      plt.figure(figsize=(10, 6))
50      frecuencias_medias_agrupadas.plot(kind='bar', color='
            skyblue', edgecolor='black', alpha=0.7)
51      plt.title("Distribucion␣de␣Frecuencias␣(100-1000␣
            repeticiones)")
52      plt.xlabel("Rango␣de␣repeticiones")
53      plt.ylabel("Cantidad␣de␣valores␣unicos")
54      plt.xticks(rotation=45)
55      plt.grid(axis='y', linestyle='--')
56      plt.tight_layout()
57      plt.savefig(os.path.join("img", "histograma_100_1000.png"),
             dpi=300)
58      plt.close()
59
60      plt.figure(figsize=(10, 6))
61      frecuencias_altas_agrupadas.plot(kind='bar', color='salmon'
            , edgecolor='black', alpha=0.7)
62      plt.title("Distribucion␣de␣Frecuencias␣(1001-10,000␣
            repeticiones)")
63      plt.xlabel("Rango␣de␣repeticiones")
64      plt.ylabel("Cantidad␣de␣valores␣unicos")
65      plt.xticks(rotation=45)
66      plt.grid(axis='y', linestyle='--')
67      plt.tight_layout()
68      plt.savefig(os.path.join("img", "histograma_1001_10000.png"
            ), dpi=300)
69      plt.close()
70
71      print("Graficos␣guardados␣en␣/img/")
```

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

Referencias

[1] Autor referencia 1

[2] Autor referencia 2

[3] Autor referencia 3