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 23 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

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 Github el proyecto se encuentra dentro de la carpeta `Implementación`.
2. Ejecutar los siguientes scripts para iniciar el contenedor del proyecto:[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.

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

Capítulo 3

Caracterización de datos de trayectorias individuales

3.1 Caracterización de datos de trayectorias individuales

El primer paso en el proceso de análisis de datos es la caracterización de los datos. Este proceso implica examinar y comprender la estructura, el contenido y las características de los datos antes de realizar cualquier análisis más profundo. A continuación, se describen las tareas necesarias para caracterizar los datos de trayectorias individuales:

- Cargar los datos de trayectorias individuales desde un archivo CSV.
- Explorar las primeras filas del conjunto de datos para obtener una visión general de su estructura.
- Verificar el número total de registros y columnas en el conjunto de datos.
- Identificar y eliminar columnas innecesarias que no aportan valor al análisis.
- Identificar y eliminar las filas innecesarias que no aportan valor al análisis.

A continuación, se detallan los pasos específicos realizados en el proceso de caracterización:

3.1.1 Exploración inicial

Se exploraron las primeras 2 filas del conjunto de datos para obtener una visión general de su estructura. Esto incluye la identificación de las columnas presentes y un par de registros. Esto se logró gracias al siguiente código:

```
import dask.dataframe as dd

ruta_archivo = "Mobility_Data.csv"

ddf = dd.read_csv(
    ruta_archivo,
```

```
            encoding="utf-8",
            sep=",",
            dtype="object",    luego)
        )

        columnas = ddf.columns.tolist()

        print("Columnas␣y␣2␣ejemplos␣por␣cada␣una:\n")
        for col in columnas:
            ejemplos = ddf[col].head(2).values.tolist()
            print(f"-␣{col}:␣{ejemplos}")
```

Código 3.1: csv_glance.py, exploración inicial del conjunto de datos.

El resultado de la ejecución de este código es el siguiente:

1. `id`: Identificador numérico único para cada registro ['34284565','34284566'].
2. `identifier`: Identificador único del dispositivo ['f2640430-7e39-41b7-80bb-3fddaa44779c', 'f2640430-7e39-41b7-80bb-3fddaa44779c'].
3. `identifier_type`: Tipo de identificador del dispositivo. En este caso, 'gaid' (Google Advertising ID para Android). Otros posibles: 'idfa' (Apple), 'imei' ['gaid', 'gaid'].
4. `timestamp`: Fecha y hora del registro de movilida ['2022-11-07 02:04:21', '2022-11-08 17:29:35']
5. `device_lat/device_lon`: Coordenadas geográficas (latitud y longitud) donde se detectó el dispositivo ['21.843149', '21.843149'], ['-102.196838', '-102.196838'].
6. `country_short/province_short`: Código del país (MX = México) y región (MX.01 = Aguascalientes, según estándar ISO) ['MX', 'MX'], ['MX.01', 'MX.01'].
7. `ip_address`: Dirección IP del dispositivo (en formato IPv6)['2806:103e:16::', '2806:103e:16::'].
8. `device_horizontal_accuracy`: Precisión del GPS en metro. Menor valor = mayor precisión ['8.0', '8.0'].
9. `source_id`: Hash único que identifica la fuente de los datos. Puede ser un identificador de la aplicación o del dispositivo ['449d086de6d9c3d192345c992dfac54319b9d550a92bcd20c37f8368cb428344', '449d086de6d9c3d192345c992dfac54319b9d550a92bcd20c37f8368cb428344'].
10. `record_id`: Identificador único del registro de movilidad (diferente al id) ['77d795df-6972-4f00-ac41-d10d1812bb2d', '8f8e1281-bc4d-4d2c-b00b-eb5c52d75bc1'].
11. `home_country_code`: País de residencia del usuario ['MX', 'MX'].
12. `home_geog_point/work_geog_point`: Coordenadas geográficas del hogar y del trabajo en formato WKT (Well-Known Text) ['POINT(-102.370380092263 22.2075340951743)', 'POINT(-102.370380092263 22.2075340951743)'].
13. `home_hex_id/work_hex_id`: Identificador hexadecimal del hogar y del trabajo, representando una ubicación geográfica en un sistema de cuadrícula hexagonal ['85498853fffffff', '85498853fffffff'].
14. `data_execute`: Fecha de procesamiento del registro de movilidad, no necesariamente la fecha de recolección ['2023-05-30', '2023-05-30'].
15. `time_zone_name`: Zona horaria del dispositivo ['America$/Mexico_City', 'America/Mexico_City'].

3.1.2 Número de registros y columnas

Se verificó el número total de registros y columnas en el conjunto de datos utilizando el siguiente código:

```
import dask.dataframe as dd

ruta_archivo = "Mobility_Data.csv"
columnas_usar = ["id"]

ddf = dd.read_csv(
    ruta_archivo,
    usecols=columnas_usar,
    sep=",",
    dtype={"id": "str"},
    blocksize="256MB",
)

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

print(f"␣Total␣de␣registros:␣{total_registros:,}")
```

Código 3.2: csv_count_registers.py, conteo de registros en el conjunto de datos.

El resultado de la ejecución de este código es 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 Identificar y eliminar campos innecesarios que no aportan valor al análisis

Ya que tenemmos un conjunto de datos con 19 campos, es importante identificar y eliminar aquellas que no aportan valor al análisis. Para ello, vamos a revisar los valores únicos de los siguiente campos, para determinar si son redundantes o no aportan información relevante. A continuación, se presentan los campos que se consideran innecesarios:

- `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

Para eliminar estos campos vamos a tomar solamente las columnas que sí vamos a conservar. A continuación, se muestra el código utilizado para realizar esta tarea:

```
import dask.dataframe as dd

# Columnas que si vamos a conservar
columnas_deseadas = [
    'identifier',
    'timestamp',
    'device_lat',
    'device_lon',
    'device_horizontal_accuracy',
    'record_id',
    'time_zone_name'
    ]

# Cargar solo las columnas necesarias
df = dd.read_csv('Mobility_Data.csv', usecols=
   columnas_deseadas)

# Guardar el resultado
df.to_csv('Mobility_Data_Slim.csv', index=False,
   single_file=True, encoding='utf-8-sig')
```

Código 3.3: remove_columns.py, eliminación de campos innecesarios en el conjunto de datos.

El resultado de la ejecución de este código es un nuevo archivo CSV que se guarda en la misma carpeta el cual contiene únicamente las columnas seleccionadas, eliminando así las que no aportan valor al análisis.

3.1.4 Identificar y eliminar las filas innecesarias que no aportan valor al análisis

Ahora que tenemos una base datos más ligera, es importante identificar y eliminar las filas que no aportan valor al análisis. Para ellos, vamos a realizar diagramas que nos permitan identificar la distribución de los datos y poder determinar si hay filas que no aportan valor al análisis. Las tablas seleccionadas para este análisis son las siguientes:

- `identifier`: Identificador único del dispositivo.
- `device_horizontal_accuracy`: Precisión del GPS en metro. Menor valor = mayor precisión.

La primer columna analizada fue `device_horizontal_accuracy`, que representa la precisión del GPS en metros. Estos valores dependen del sistema de medición y la fuente de los datos, generalemente se sigue la siguiente escala:

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

Con base en esta escala, el primer paso es identificar el rango de valores que contiene la columna. Para ello, se utilizó el *Código:* 3.4, que permite obtener los valores únicos de la columna `device_horizontal_accuracy` y guardarlos en un archivo de texto.

```
import dask.dataframe as dd
import pandas as pd
from tqdm import tqdm

archivo_csv = "Mobility_Data.csv"
columna_objetivo = "device_horizontal_accuracy"
archivo_salida = "valores_unicos.txt"
chunksize = 1_000_000  # Procesar 1M de registros a la
    vez

valores_unicos = set()

for chunk in tqdm(pd.read_csv(archivo_csv, usecols=[
    columna_objetivo], chunksize=chunksize)):
     valores_unicos.update(chunk[columna_objetivo].
         dropna().astype(str))


with open(archivo_salida, "w", encoding="utf-8") as f:
     f.write("\n".join(sorted(valores_unicos)))

print(f"\n␣Se␣encontraron␣{len(valores_unicos):,}␣
    valores␣unicos.")
print(f"␣Guardados␣en:␣{archivo_salida}")
```

```
print("\n␣Ejemplo␣de␣valores␣unicos:")
print("\n".join(sorted(valores_unicos)[:10]))
```

Código 3.4: unique_values.py, obtención de valores únicos de la columna 'device_horizontal_accuracy'.

Una vez obtenidos los valores únicos pudimos determinar que el rango de valores es de 0 a 200. Este rango es muy importante para poder crear un script que grafique los valores de la columna `device_horizontal_accuracy` y que nos permita identificar la frecuencia de los valores para poder determinar si hay filas que no aportan valor al análisis. Este script se muestra en el *Código:* 3.5.

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

archivo_csv = "Mobility_Data_Slim.csv"
columna = "device_horizontal_accuracy"
bins = 100
os.makedirs("img", exist_ok=True)

frecuencias = pd.Series(dtype=float)
for chunk in pd.read_csv(archivo_csv, usecols=[columna
    ], chunksize=1_000_000):
     frecuencias = pd.concat([frecuencias, chunk[columna
         ].value_counts()])

counts, edges = np.histogram(frecuencias.index, bins=
    bins, weights=frecuencias.values)

plt.figure(figsize=(12, 6))
plt.bar(edges[:-1], counts, width=np.diff(edges), align
    ='edge', edgecolor='black', alpha=0.7)
plt.title("Frecuencias␣de␣Valores␣Agrupados␣por␣Rangos␣
    (0-200)")
plt.xlabel("Rango␣de␣Valores")
plt.ylabel("Frecuencia␣Total␣(Millones)")
plt.xticks(edges[::5], rotation=45)
plt.grid(axis='y', linestyle='--')

output_path = "img/histograma_frecuencias_rangos.png"
plt.savefig(output_path, dpi=300, bbox_inches='tight')
plt.close()
```

Código 3.5: accuracy_histogram.py, creación de un histograma de frecuencias de la columna 'device_horizontal_accuracy'.

El resultado de la ejecución del script anterior es un histograma que muestra la frecuencia de los valores agrupados por rangos. En la siguiente figure se puede observar

que la mayoría de los valores se encuentran en el rango de 0 a 20 metros, lo cual es consistente con la precisión del GPS puro. Con base en esto podemos acotar los valores que serán considerados válidos para el análisis.

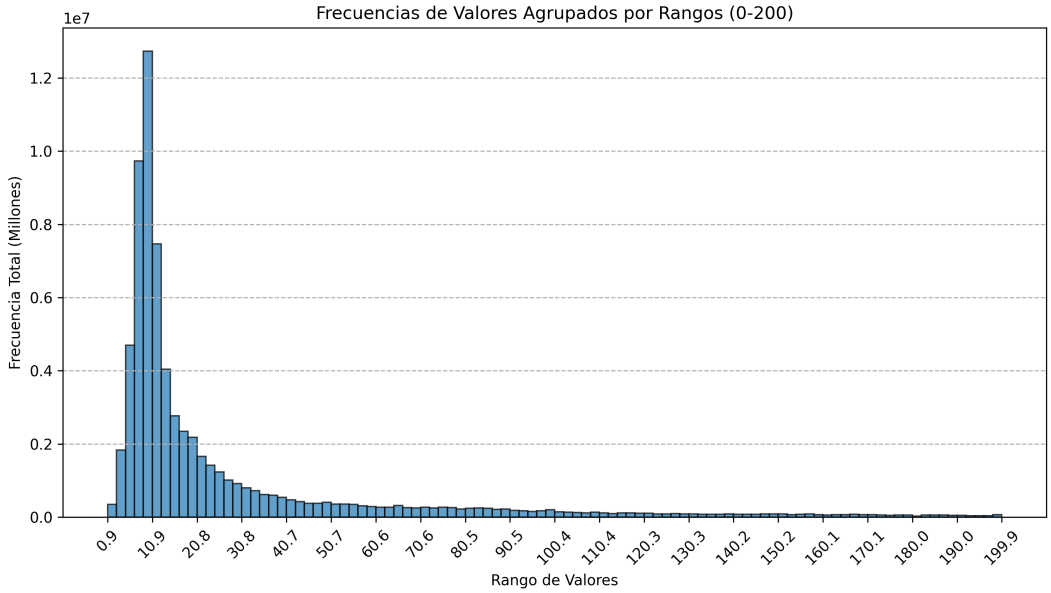


Figura 3.1: Frecuencias de valores agrupados por rangos (0-200).

Para poder tomar una mejor decisión sobre los valores que serán considerados válidos para el análisis, se realizó un conteo de los valores respecto a los rangos de precisión del GPS.

- GPS puro, satelital (1-20 metros): 68.73 %.
- A-GPS, asistido por red (5-50 metros): 16.56 %.
- Triangulación por WiFi/redes móviles (20-500 metros): 14.69 %.

La siguiente columna analizada fue `identifier`, la cual representa el identificador único del dispositivo. El primer paso de este análisis es identificar la distribución de los valores únicos. Para ello, se utilizó un código que crea un gráfico que separa por intervalos el número de repeticiones, y pone la cantidad de valores único de forma logarítimica.

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter

archivo_csv = "Mobility_Data_Slim.csv"
columna = "identifier"
chunksize = 1_000_000
os.makedirs("img", exist_ok=True)

counter = Counter()
for chunk in pd.read_csv(archivo_csv, usecols=[columna],
    chunksize=chunksize):
     counter.update(chunk[columna].dropna().astype(str))

frecuencias = pd.Series(counter)

max_freq = frecuencias.max()
bins = [0] + [10**i for i in range(0, int(np.log10(max_freq
    )) + 2)]  # Ej: [0, 1, 10, 100, 1000, ...]
frecuencias_agrupadas = pd.cut(frecuencias, bins=bins,
    right=False).value_counts().sort_index()

plt.figure(figsize=(12, 7))
frecuencias_agrupadas.plot(kind='bar', logy=True, alpha
    =0.7, edgecolor='black')

plt.xticks(rotation=45, ha='right')  # Rotar etiquetas para
     mejor legibilidad
plt.title("Distribucion de Frecuencias Agrupadas en Bloques
    de 1000 Repeticiones")
plt.xlabel("Rango de repeticiones (ej: [0, 1000) significa
    0-999 repeticiones)")
plt.ylabel("Cantidad de valores unicos (log)")
plt.grid(True, which="both", ls="--", axis='y')

output_path = os.path.join("img", "
    histograma_frecuencias_agrupadas_1000.png")
plt.savefig(output_path, dpi=300, bbox_inches='tight')
plt.close()
```

Código 3.6: identifier_histogram.py, creación de un histograma de frecuencias de la columna 'identifier'.

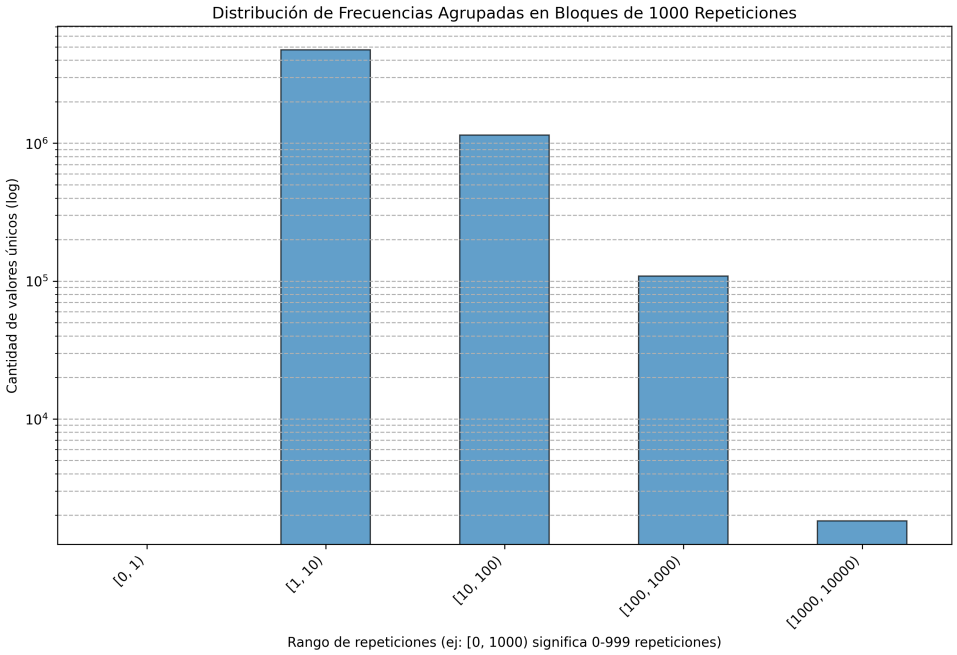


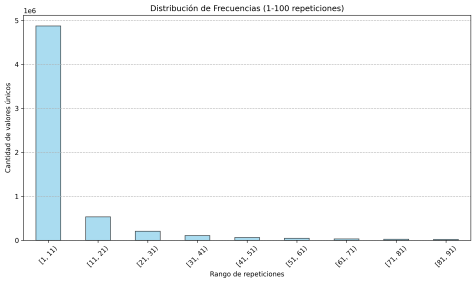
Figura 3.2: Distribución de frecuencias agrupadas en bloques de 1000 repeticiones.

Gracias a la *Figura:* 3.2 podemos observar la frecuencia de los valores únicos. Para poder obtener información más detallada, usaremos el *Código:* 3.7. Este código nos permite obtener un análisis más detallados, separando los valores en tres rangos: 1-100, 100-1000 y 1000-10000.
Además de los histogramas de la figura anterior el código también genera un resumen de las frecuencias de los valores únicos en cada rango. Este resumen se muestra a continuación:

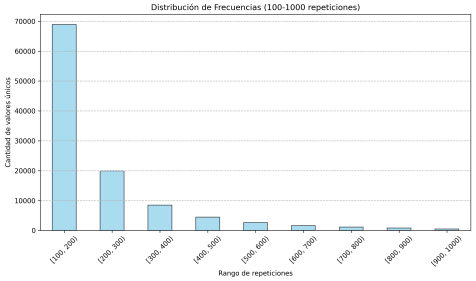
```
=== Resumen de frecuencias ===
**Rango 1-100 repeticiones**:
- Total de valores unicos: 5,912,437

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

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

Gracias a esta información podemos observar que la mayoría de los identificadores únicos se encuentran en el rango de 1 a 100 repeticiones, dentro del cual la gran mayoría de ellos tienen entre una a diez repeticiones. Con base en podemos acotar los valores que serán considerados válidos para el análisis.

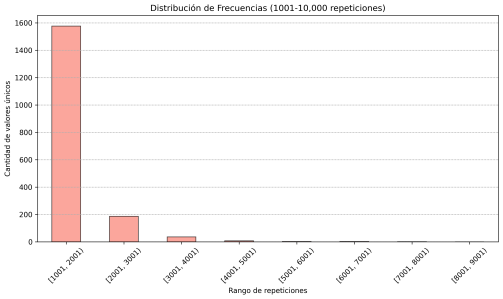
```
import os
import pandas as pd
import matplotlib.pyplot as plt
```



(a) Histograma 1-100



(b) Histograma 100-1000



(c) Histograma 1001-10000

Figura 3.3: Comparación de histogramas

```
import numpy as np
from collections import Counter

archivo_csv = "Mobility_Data_Slim.csv"
columna = "identifier"
chunksize = 1_000_000
os.makedirs("img", exist_ok=True)

counter = Counter()
for chunk in pd.read_csv(archivo_csv, usecols=[columna],
    chunksize=chunksize):
     counter.update(chunk[columna].dropna().astype(str))
frecuencias = pd.Series(counter)

frecuencias_bajas = frecuencias[(frecuencias >= 1) & (
    frecuencias <= 99)]
frecuencias_medias = frecuencias[(frecuencias >= 100) & (
    frecuencias <= 1000)]
frecuencias_altas = frecuencias[(frecuencias >= 1001) & (
    frecuencias <= 10000)]

bins_bajas = list(range(1, 100, 10))  # 1-99 en pasos de 10
bins_medias = list(range(100, 1001, 100))  # 100-1000 en
    pasos de 100
bins_altas = list(range(1001, 10001, 1000))  # 1001-10000
```

```
    en pasos de 1000

freciencias_bajas_agrupadas = pd.cut(frecuencias_bajas,
    bins=bins_bajas, right=False).value_counts().sort_index
    ()
frecuencias_medias_agrupadas = pd.cut(frecuencias_medias,
    bins=bins_medias, right=False).value_counts().sort_index
    ()
frecuencias_altas_agrupadas = pd.cut(frecuencias_altas,
    bins=bins_altas, right=False).value_counts().sort_index
    ()

print("\n=== Resumen de frecuencias ===")
print(f"\n**Rango 1-100 repeticiones**:")
print(f" - Total de valores unicos: {len(frecuencias_bajas)
    }")
print(f"\n**Rango 100-1000 repeticiones**:")
print(f" - Total de valores unicos: {len(frecuencias_medias
    )}")
print(f"\n**Rango 1000-10000 repeticiones**:")
print(f" - Total de valores unicos: {len(frecuencias_altas)
    }")


plt.figure(figsize=(10, 6))
freciencias_bajas_agrupadas.plot(kind='bar', color='skyblue
    ', edgecolor='black', alpha=0.7)
plt.title("Distribucion de Frecuencias (1-100 repeticiones)
    ")
plt.xlabel("Rango de repeticiones")
plt.ylabel("Cantidad de valores unicos")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--')
plt.tight_layout()
plt.savefig(os.path.join("img", "histograma_1_100.png"),
    dpi=300)
plt.close()

plt.figure(figsize=(10, 6))
frecuencias_medias_agrupadas.plot(kind='bar', color='
    skyblue', edgecolor='black', alpha=0.7)
plt.title("Distribucion de Frecuencias (100-1000
    repeticiones)")
plt.xlabel("Rango de repeticiones")
plt.ylabel("Cantidad de valores unicos")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--')
plt.tight_layout()
plt.savefig(os.path.join("img", "histograma_100_1000.png"),
     dpi=300)
plt.close()
```

```
    plt.figure(figsize=(10, 6))
    frecuencias_al   plt.title("Distribucion␣de␣Frecuencias␣
       (1001-10,000␣repeticiones)")
    plt.xlabel("Rango␣de␣repeticiones")
    plt.ylabel("Cantidad␣de␣valores␣unicos")
    plt.xticks(rotation=45)
    plt.grid(axis='y', linestyle='--')
    plt.tight_layout()
    plt.savefig(os.path.join("img", "histograma_1001_10000.png"
       ), dpi=300)
    plt.close()

    print("Graficos␣guardados␣en␣/img/")
```

Código 3.7: identifier_histogram_detailed.py, análisis de frecuencias de la columna 'identifier'.

3.1.5 Identificar y manejar valores faltantes o nulos en el conjunto de datos

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:

```
sudo apt update
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: Iniciar contenedor del proyecto.

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:

```
version: "3.8"

services:
  data-analysis:
    image: python:3.13-bookworm
    container_name: data-analysis
    runtime: nvidia
    tty: true
    stdin_open: true
    volumes:
      - ./:/app
      - python-packages:/usr/local/lib/python3.13/site-packages
    command: sh -c "pip␣install␣-r␣requirements.txt␣&&␣pip␣
       install␣-e␣.␣&&␣python3␣/app/src/main.py"
    working_dir: /app
    environment:
      - PYTHONPATH=/app

volumes:
  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.

Apéndice B

Scripts para la caracterización de datos

Referencias

[1] Autor referencia 1

[2] Autor referencia 2

[3] Autor referencia 3