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 11 de diciembre de 2025

Contenido

Lista de Figuras

Lista de Códigos

Capítulo 1

Introducción del Proyecto

1.1 Importancia de los modelos de movilidad para la evaluación de protocolos para redes móviles

La simulación de una red de comunicaciones en donde intervienen dispositivos personales de comunicación requiere de 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 de datos existente que permita modelar trayectorias de una forma eficaz.

1.2 Proceso de diseño de un modelo de movilidad

El diseño de un modelo de movilidad implica varias etapas:

- **Caracterización de datos**: Limpieza, depuración y análisis exploratorio.
- **Identificación de trayectorias**: Extracción de secuencias de movimiento significativas.
- **Validación y evaluación**: Medición de la calidad y representatividad de las trayectorias.

1.3 Objetivo del proyecto

El objetivo principal del proyecto es obtener una caracterización estadística de las trayectorias individuales a partir de un conjunto de datos de movilidad. Esto incluye la identificación de trayectorias peatonales y su análisis mediante herramientas de IA.

1.4 Logros

- Reducir el conjunto de datos de 69.98 millones de registros, 19 columnas y un peso de 22 GB a 51 millones de registros, 7 columnas y 7 GB.
- Identificar que el 68.73 % de los registros tienen precisión GPS satelital (1-20 metros).
- Desarrollar un algoritmo de evaluación de calidad de trayectorias con métrica como volumen, cobertura temporal, precisión GPS y diversidad espacial.
- Encontrar un respaldo del algoritmo de evaluación usando el algoritmo de clusterización K Medias con los puntos de recorrido.
- Graficar la distribución de las longitudes de vuelo para poder aproximar una distribución probabilística.

Capítulo 2

Marco teórico

2.1 Elementos de un modelo de movilidad

Un modelo de movilidad describe el desplazamiento de individuos o grupos en el espacio y el tiempo. Este proyecto se enfoca en trayectorias peatonales individuales, definidas por:

- **Puntos de recorrido**: Ubicaciones específicas por las que transita el individuo.
- **Tiempos de pausa**: Intervalos durante los cuales el individuo permanece detenido en un mismo punto de recorrido.
- **Longitud de vuelo**: Distancia entre dos puntos de recorrido consecutivos.

A continuación se muestran la determinación de cada uno de estos elementos.

2.1.1 Punto de recorrido

Cada punto de recorrido se determinó a partir de los registros GPS contenidos en el conjunto de datos, específicamente mediante las coordenadas geográficas *device_lat* y *device_lon*. Para garantizar la calidad de estos puntos, se aplicó un filtro de precisión GPS, considerando únicamente aquellos registros con *device_horizontal_accuracy* menor a 10 metros, correspondiente a la precisión satelital.

2.1.2 Tiempos de pausa

Para determinar estos tiempos de pausa, se analizó la secuencia temporal de cada individuo. Cuando dos registros consecutivos presentaban coordenadas idénticas (o variaciones menores a un umbral de 0.001 grados, aproximadamente 111 metros), se interpretó que el individuo permaneció en pausa entre dichos registros.
Adicionalmente, como no es posible saber en que parte del intervalo se encuentra el primer punto de recorrido registrado. Para calcular los tiempos de pausa se asume

que el primer punto de recorrido es cuando el individuo llegó a dicho punto. Esto aunque es una simplificación es de mucha utilidad para los objetivos del proyecto.

2.1.3 Longitud de vuelo

Para calcular la distancia entre dos puntos de recorrido, se utiliza la fórmula de Haversine aplicada a las coordenadas geográficas de registros sucesivos de un mismo individuo. Solo se consideraron desplazamientos significativos, definidos como aquellos donde el cambio de coordenadas superó un umbral de 0.001 grados (aproximadamente 111 metros). Este filtro permite distinguir entre movimientos reales y variaciones menores debidas al error de medición del GPS.

Capítulo 3

Objetivos y metodología

3.1 Objetivos

El objetivo principal del proyecto es obtener una caracterización estadística de las trayectorias individuales a partir de un conjunto de datos de movilidad.

Los objetivos particulares son:

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

3.2 Metodología

El proceso se divide en las siguientes etapas:

1. Caracterización inicial

- Exploración del conjunto de datos, identificación de columnas y filas irrelevantes.
- Conteo de registros y dimensiones de la base de datos.
- Inspección de valores únicos y estructura general.

2. Limpieza de datos

- Selección de campos relevantes para el análisis.
- Eliminación de columnas redundantes o sin valor analítico.
- Conservación de columnas clave: *identificador*, *timestamp*, *coordenadas*, *precisión GPS*.

- Eliminación de registros duplicados basados en *identificador*, *timestamp* y *coordenadas*.
- Filtrado de registros con baja precisión GPS mayor a 20 metros.
- Análisis de frecuencia de aparición de identificadores únicos.
- Desarrollo de algoritmo de puntuación compuesta basado en métricas
- Clasificación de trayectorias en categorías cualitativas.

3. Determinación de puntos de recorrido

- Distribución de individuos por día mediante histogramas temporales.
- Cálculo de frecuencia de aparición y persistencia temporal.
- Identificación de individuos con registros en múltiples días.
- Cálculo de velocidades entre puntos consecutivos.
- Creación de histograma de distribución de velocidades.
- Aplicación de filtro peatonal utilizando parámetros estadísticos:
  - Media de velocidad peatonal: 1.34 m/s
  - Desviación estándar: 0.37 m/s
  - Rango aceptable: 0.6-2.08 m/s (media $\pm$ desviaciones estándar)
- Segmentación de trayectorias para eliminar puntos fuera del rango peatonal
- Mapa de distribución de puntos de recorrido por ciudad.
- Visualización de trayectorias individuales completas.
- Implementación de algoritmo de *clustering* sobre puntos de velocidad peatonal.

4. Determinación de tiempos de pausa

- Cálculo de tiempos de pausa.

5. Determinación de longitudes de vuelo

- Cálculo de longitud de vuelo entre puntos consecutivos.

Capítulo 4

Desarrollo

4.1 Caracterización inicial

4.1.1 Exploración 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']`

4.1.2 Conteo de registros y dimensiones de la base de datos

Conocer las dimensiones exactas del conjunto de datos es crucial para planificar el análisis posterior, se utiliza la biblioteca *Dask DataFrame*, 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.

4.2 Limpieza de datos

4.2.1 Selección de campos relevantes para el análisis

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`

4.2.2 Eliminación de columnas redundantes o sin valor analítico

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.

4.2.3 Filtrado de registros con baja precisión GPS

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:

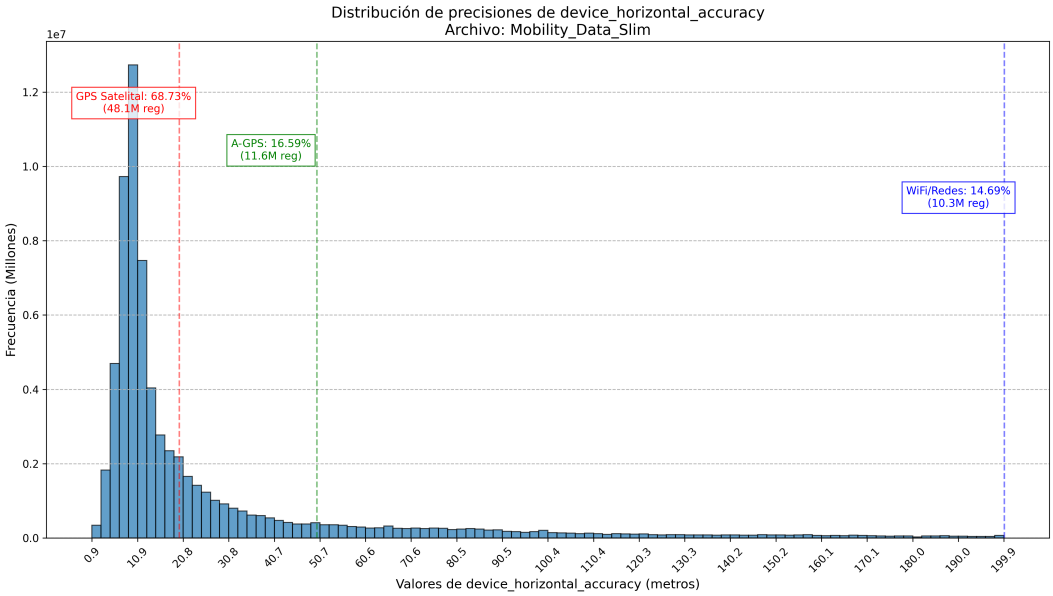


Figura 4.1: Frecuencia de aparación de los valores de 'device_horizontal_accuracy'.

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 4.1, 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).

4.2.4 Análisis de frecuencia de aparición de identificadores únicos

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).

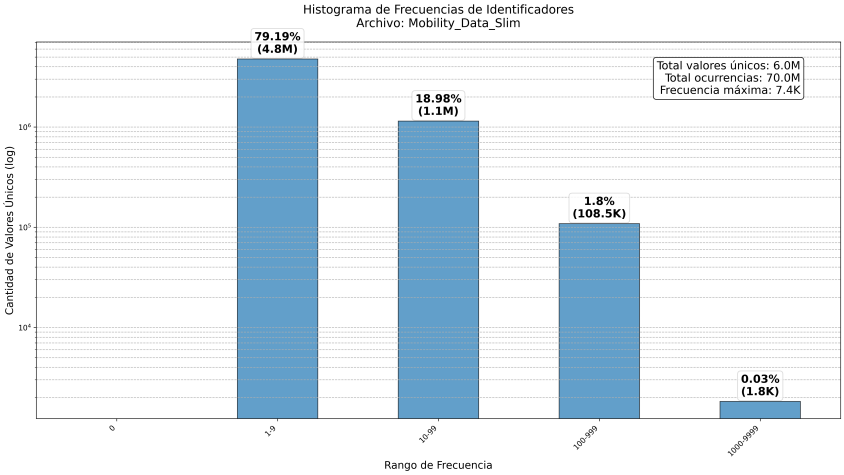
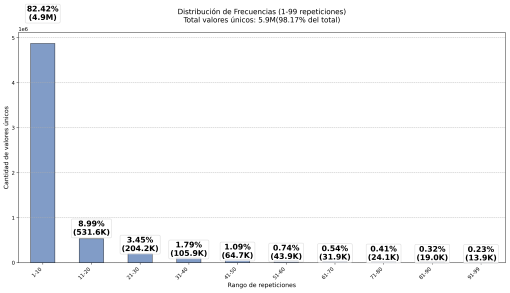
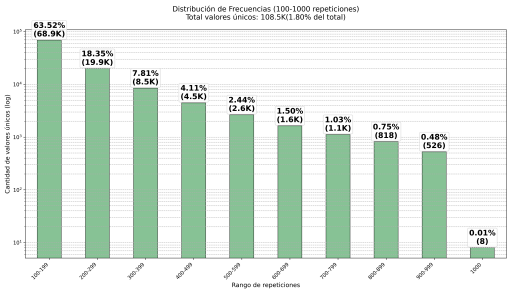


Figura 4.2: Frecuencia de aparición de los identificadores únicos.

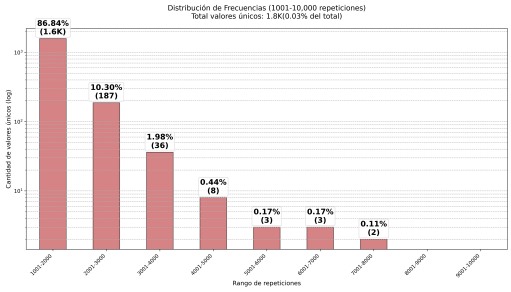
Ejecutar este código permite saber que el total de individuos es de **6,022,772** de los cuales el **79.19 %** tienen una frecuencia de aparición de una a nueve veces, esto es **4,769,317** de individuos. Así mismo de la Figura 4.2 se observa que hay poco más de un **20 %** de individuos con más de 99 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.



(a) Histograma 1-99 repeticiones



(b) Histograma 100–1000 repeticiones



(c) Histograma 1001–10000 repeticiones

Figura 4.3: 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

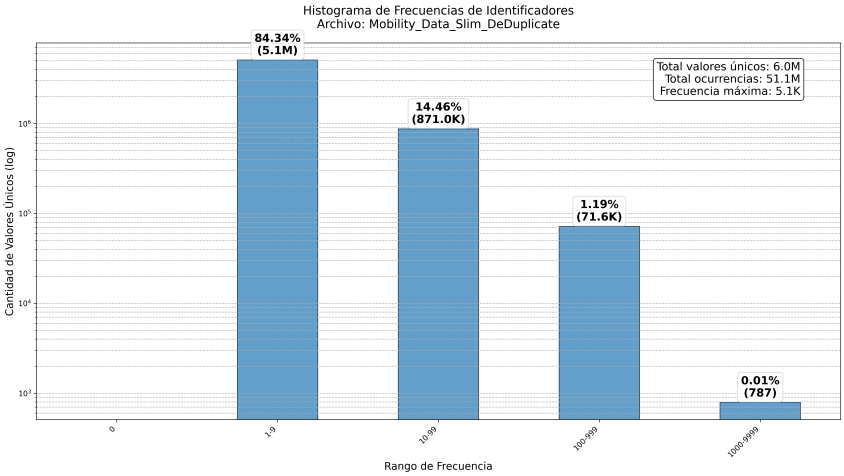


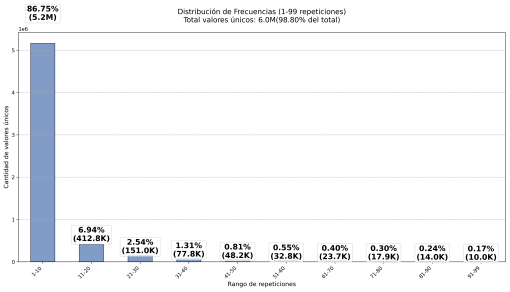
Figura 4.4: Frecuencia de aparición de los identificadores únicos después de eliminar duplicados.

Comparando los resultados de la Figura 4.2 y la Figura 4.4 podemos destacar varios hallazgos importantes:

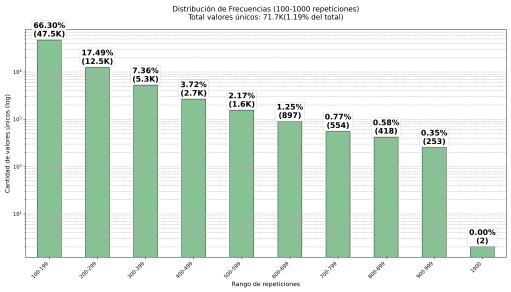
- El número de individuos (**6,022,772**) se mantuvo sin cambios.
- La eliminación del **27 %** de registros. De **70 millones** a **51 millones** de registros.
- La reducción del **31.1 %** en la frecuencia máxima de aparición (de 7,400 a 5,100) corrige sesgos que afectaban especialmente a individuos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

De la Figura 4.5 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.

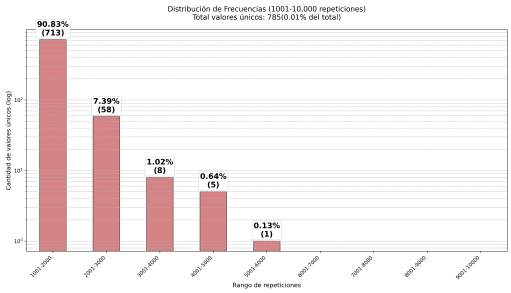
Dado los resultados obtenidos en esta etapa de la caracterización, se concluye que no es necesario eliminar ninguna fila del conjunto de datos, ya que la depuración de columnas y la eliminación de duplicados han sido suficientes para optimizar la calidad del conjunto de datos.



(a) Histograma 1-99 repeticiones



(b) Histograma 100–1000 repeticiones



(c) Histograma 1001–10000 repeticiones

Figura 4.5: Comparación de histogramas por rangos de repeticiones.

4.2.5 Desarrollo de algoritmo de puntuación compuesta basado en métricas

A partir de esta etapa se crea una base de datos llamada *trajectories* en la que se migra el conjunto de datos limpio a una tabla llamada *mobility_data*, el script B.10 analiza automáticamente la estructura del CSV, mapea los tipos de datos, y carga los datos en partes de 1,000 registros para optimizar el rendimiento. Una vez migrados los datos se analizan los datos el primer paso para encontrar las mejores trayectorias es identificar el porcentaje de puntos de recorrido que tienen una precisión de GPS mejor a 20 metros. Para esto se ejecuta el query del Apéndice B.11, cuyo resultado se muestra en la siguiente tabla.

Total Puntos de Recorrido	Total Puntos de Recorrido con Precisión GPS	Porcentaje de Puntos de Recorrido con Precisión GPS
51,077,925	34,364,037	67.28 %

Figura 4.6: Porcentaje de puntos de recorrido con precisión GPS.

Debido a que una trayectoria debe tener más un punto de recorrido, para este análisis se consideran únicamente los individuos que tienen más de tres puntos de recorrido. Para así obtener el porcentaje de individuos que cumplen con esta condición. Se ejecuta el query del Apéndice B.12, cuyo resultado se muestra en la siguiente tabla.

Total de Individuos	Individuos con más de tres puntos	Porcentaje de Puntos de Individuos con más de tres puntos
6,022,772	2,119,560	35.19 %

Figura 4.7: Porcentaje de individuos con más de tres puntos de recorrido.

Ahora hay que identificar los individuos que tienen más de tres puntos de recorrido y que además tienen una precisión de GPS mejor a 20 metros. Para esto se ejecuta el query del Apéndice B.13, cuyo resultado se muestra en la siguiente tabla.

Total de Individuos	Individuos con más de tres puntos y precisión GPS	Porcentaje de Puntos de Individuos con más de tres puntos y precisión GPS
6,022,772	1,534,172	25.47 %

Figura 4.8: Porcentaje de individuos con más de tres puntos de recorrido y precisión GPS.

Estos queries permiten identificar la calidad de los datos. A partir de estos se plantea un algoritmo de evaluación de la calidad de las trayectorias, el cual se encuentra en el Apéndice B.14.

Este algoritmo inicia con la detección de movimientos significativos. Un desplazamiento se considera significativo cuando el cambio en coordenadas entre registros consecutivos supera un umbral de 0.001 grados (equivalente a aproximadamente 111 metros). Posteriormente, se calculan para cada individuo un conjunto de métricas fundamentales de calidad de datos:

- **Volumen de datos** (records_counts): Número total de registros GPS.
- **Cobertura temporal** (time_span_days): Duración total (en días) entre el primer y el último registro.
- **Consistencia temporal** (active_days_count): Número de días únicos con al menos un registro.
- **Calidad técnica** (avg_accuracy_meters): Precisión promedio del GPS (en metros), reportada por el dispositivo.
- **Riqueza del movimiento** (movement_points): Cantidad de movimientos significativos detectados.
- **Diversidad espacial** (spatial_range): Rango geográfico total (en grados) cubierto por la trayectoria, medido como la diferencia máxima en latitud y longitud.

Cada métrica se interpreta según los siguientes criterios:

Volumen de datos : Un mayor número de registros proporciona una base más sólida para el análisis. Se considera que trayectorias con más de 500 registros son ideales, mientras que el mínimo funcional se establece en 50 registros.

Cobertura y consistencia temporal : Periodos de observación extensos (*time_span_days*) permiten capturar patrones a largo plazo. La relación de actividad (*activity_ratio = active_days_count / time_span_days*) mide la regularidad en la recolección de datos.

Precisión GPS : Es determinante para la confiabilidad de los análisis. Valores menores a 10 metros se consideran excelentes, entre 10-30 metros buenos, y mayores a 50 metros requieren precaución interpretativa.

Riqueza de movimiento : Refleja la dinámica de la trayectoria. Valores altos sugieren patrones de movilidad complejos, mientras que valores bajos indican comportamientos mayormente estáticos.

Diversidad espacial : Resulta fundamental para estudios que requieren comprender la amplitud de los patrones de movilidad urbana.

Para obtener una evaluación integral, se implementa un sistema de puntuación ponderada que normaliza y combina las seis métricas en una puntuación global (*quality_score*) entre 0 y 100. La fórmula para cada componente es la siguiente:

- Volumen (25 %): min(100, records_count / 5.0)
- Duración (20 %): min(100, time_span_days / 0.3)
- Regularidad (20 %): min(100, (active_days_count / time_span_days) * 100)
- Precisión (15 %): max(0, 100 - avg_accuracy_meters)
- Movilidad (10 %): min(100, movement_points / 1.0)
- Diversidad espacial (10 %): min(100, spatial_range / 0.01 * 100)

Esta puntuación final se traduce a una categoría cualitativa según los siguiente umbrales:

1. **EXCELENTE:** ($\geq$80)
2. **MUY BUENA:** ($\geq$65)
3. **BUENA:** ($\geq$50)
4. **REGULAR:** ($\geq$35)
5. **BAJA:** ($<$35)

4.2.6 Clasificación de trayectorias en categorías cualitativas

La clasificación de trayectorias en categorías cualitativas se implementa mediante el script B.15, que constituye el núcleo del **pipeline de procesamiento** del proyecto. Este script ejecuta un sistema integral de filtrado, clasificación de calidad y segmentación de trayectorias, diseñado para extraer del conjunto de datos anteriormente depurado un conjunto de **trayectorias peatonales validadas**. Para el

filtrado y segmentación de movimiento peatonal para cada individuo seleccionado, el *script* aplica un proceso de refinamiento específico:

- **Cálculo de Velocidad:** Para cada par de puntos consecutivos, calcula la distancia real en metros mediante la **fórmula de *Haversine*** y la divide por la diferencia de tiempo, obteniendo la velocidad instantánea en m/s.
- **Aplicación del Filtro Peatonal:** Se descartan los puntos cuya velocidad no se encuentre dentro del **rango estadístico típico de caminata humana**. Los parámetros se basan en literatura científica: media de 1.34 m/s, desviación estándar de 0.37 m/s. El **umbral de aceptación se define como media $\pm 2\sigma$ (0.6 − 2.08 m/s o 2.16 − 7.49 km/h)**.
- **Segmentación Temporal:** La trayectoria se **divide en segmentos continuos** de movimiento puramente peatonal. Si se detecta un punto con velocidad fuera del rango (ej., un viaje en auto), la trayectoria se segmenta, aislando así los períodos válidos de caminata.
- **Agrupación Diaria:** El procesamiento se realiza **por fecha independiente**, preservando la coherencia temporal y facilitando análisis posteriores por día de actividad.

El *script* produce tres archivos clave en el directorio `pedestrian_analysis/`:

1. `trajectory_classification.csv`: Catálogo completo con las métricas de calidad, puntuación y categoría para cada `identifier`.
2. `pedestrian_trajectories_all.csv`: **Dataset principal del proyecto**. Contiene únicamente los puntos GPS validados como movimiento peatonal. Incluye las columnas críticas:
    - `segment_id`: Identificador único para cada tramo continuo de caminata.
    - `speed_ms`: Velocidad calculada en metros/segundo.
3. `statistics_per_person.csv`: Estadísticas agregadas por individuo (número de segmentos, puntos totales, velocidad promedio, categoría de calidad).

Este procesamiento es **fundamental** para garantizar la validez del modelo final. Los datos brutos de movilidad son heterogéneos: contienen desplazamientos en diversos modos de transporte (vehículo, bicicleta, transporte público) y períodos de inactividad. Para modelar **comportamiento peatonal específicamente**, es imperativo:

- **Asegurar la calidad de los datos fuente**, descartando trayectorias escasas, irregulares o imprecisas.
- **Aislar el modo de transporte çaminata"**, filtrando puntos cuya velocidad sea incoherente con el movimiento a pie.
- **Manejar la multimodalidad**, segmentando las trayectorias para separar los períodos de caminata de otros modos de transporte dentro de un mismo día de un individuo.

Se anticipa una **reducción significativa pero enfocada** del volumen de datos:

- De ∼6 millones de individuos originales, se espera retener **decenas de miles** con calidad suficiente (≥35 puntos).
- De ∼51 millones de registros GPS, se proyecta extraer **entre 5 y 10 millones** de puntos correspondientes a movimiento peatonal validado.

El resultado final será un **dataset compacto, de alta calidad y consistente**, compuesto exclusivamente por trayectorias con cobertura suficiente y patrones de velocidad inherentes al desplazamiento peatonal, organizado en segmentos que representan fielmente períodos de caminata. Esta es la base de datos esencial para entrenar modelos de movilidad peatonal realistas.

4.3 Determinación de puntos de recorrido

4.3.1 Distribución temporal y frecuencia

El análisis de la distribución temporal y frecuencia de aparición de los individuos en el dataset es fundamental para identificar patrones de rutina y segmentar a los usuarios según su persistencia temporal. Este análisis se realiza mediante el *script* B.16, complementado con los resultados previos de B.9.

El *script* implementa un análisis exhaustivo multi-día que procesa el conjunto de datos en **chunks** para optimizar el uso de memoria. El proceso sigue estos pasos:

1. **Extracción de fechas**: Para cada registro, se extrae la fecha del campo `timestamp`.
2. **Agrupación por individuo y día**: Los registros se agrupan por `identifier` y fecha, manteniendo un conjunto (`set`) de fechas únicas para cada persona.
3. **Cálculo de estadísticas**: Para cada individuo se calcula:
    - Número de días diferentes con registros
    - Primera y última fecha de registro
    - Intervalo temporal (diferencia entre última y primera fecha)
4. **Agregación global**: Se generan estadísticas agregadas:
    - Número total de identificadores únicos
    - Cantidad de individuos que aparecen en un solo día vs. múltiples días
    - Número máximo de días registrados por un individuo
    - Promedio de días para individuos multi-día
    - Distribución completa: cuántos individuos aparecen en 1 día, 2 días, 3 días, etc.

El análisis produce dos tipos principales de resultados:

- **Histograma de distribución**: Se genera un histograma con escala logarítmica en el eje Y para visualizar la distribución del número de días por individuo.

Esta escala es necesaria dada la naturaleza típicamente sesgada de los datos, donde hay muchos individuos con 1-2 días y muy pocos con 7+ días.

- **Archivo CSV detallado**: Se guarda el archivo `multi_day_summary.csv` con los siguientes campos por individuo:
  - `identifier`: Identificador único
  - `days_count`: Número de días con registros
  - `first_date`: Primera fecha de registro
  - `last_date`: Última fecha de registro
  - `time_span_days`: Diferencia en días entre primera y última fecha

Los resultados de `routine_individuals.py` se complementan con el análisis previo realizado por `identifier_histograms_daily.py`, el cual proporciona una perspectiva inversa:

Análisis	Perspectiva proporcionada
routine_individuals.py	**Vista por individuo:** Cuántos días tiene registros cada persona, permitiendo identificar su persistencia temporal.
identifier_histograms_daily.py	**Vista por día:** Cuántos individuos aparecen en cada día específico, revelando patrones de actividad diaria en la población,

La persistencia temporal es un factor crítico para identificar rutinas y segmentar a los usuarios según sus patrones de movilidad:

- **Usuarios ocasionales (1-2 días)**: Probablemente corresponden a turistas, visitantes ocasionales o registros aislados sin valor para modelar patrones habituales de movilidad.
- **Usuarios regulares (3-6 días)**: Indican cierta consistencia en los registros, posiblemente correspondiendo a personas que realizan desplazamientos recurrentes pero no diarios.
- **Usuarios rutinarios (7+ días)**: Constituyen el grupo más valioso para el análisis, ya que su presencia continua sugiere rutinas establecidas (ej., desplazamientos casa-trabajo-casa) que son fundamentales para modelar movilidad típica.

Basado en la naturaleza de los datos de movilidad, se anticipan los siguientes patrones:

1. **Distribución sesgada**: Se espera que la gran mayoría de individuos ($> 80\,\%$) aparezca en solo 1-2 días, siguiendo una distribución de ley de potencias típica en datos de movilidad.
2. **Segmentación natural**:

- $\sim 80 - 90\,\%$: Usuarios ocasionales (1-2 días)
- $\sim 10 - 15\,\%$: Usuarios regulares (3-6 días)
- $\sim 1 - 5\,\%$: Usuarios rutinarios (7+ días)

3. **Patrones temporales**: Los días de mayor actividad probablemente serán días laborables (lunes a viernes), con menor actividad durante fines de semana y días festivos.
4. **Usuarios ideales para análisis**: Los individuos con mayor número de días de cobertura (usuarios rutinarios) constituyen los candidatos óptimos para el análisis de patrones recurrentes de movilidad y la identificación de rutinas establecidas.

Este análisis temporal permite:

- **Filtrar individuos** para análisis posteriores, seleccionando aquellos con suficiente persistencia temporal para modelar rutinas.
- **Entender la representatividad** de los datos: determinar si el dataset contiene suficientes usuarios rutinarios para un análisis significativo de patrones de movilidad.
- **Segmentar la población** según sus patrones de uso, lo que puede informar diferentes estrategias de modelado para cada grupo.
- **Identificar sesgos temporales** en la recolección de datos, como días con cobertura incompleta o períodos con baja actividad.

La combinación de estos análisis proporciona una comprensión completa de la dimensión temporal del dataset, identificando tanto la frecuencia de aparición de individuos como la distribución de actividad a lo largo del tiempo, ambos elementos esenciales para la construcción de modelos de movilidad robustos y representativos.

4.3.2 Cálculo de velocidades

El cálculo de velocidades entre puntos consecutivos de las trayectorias es una etapa crítica en el análisis de movilidad, ya que permite validar la coherencia de los datos, identificar el modo de transporte predominante y caracterizar los patrones de desplazamiento de los individuos. Esta sección utiliza dos *scripts* complementarios que operan sobre la base de datos PostgreSQL previamente creada. La velocidad instantánea se calcula entre cada par de puntos GPS consecutivos para un mismo individuo, siguiendo un procedimiento riguroso:

1. **Ordenamiento cronológico**: Los registros de cada individuo se ordenan por `timestamp`.
2. **Cálculo de distancia**: Se emplea la **fórmula de Haversine** para calcular la distancia real en metros entre puntos consecutivos, considerando la curvatura terrestre:

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{4.1}$$

donde $R = 6,371,000$ m es el radio terrestre, ϕ es la latitud en radianes, y λ es la longitud en radianes.

3. **Cálculo del intervalo temporal**: Se calcula la diferencia de tiempo en segundos entre registros consecutivos.
4. **Cálculo de velocidad**: La velocidad instantánea en m/s se obtiene como:

$$v = \frac{d}{\Delta t} \tag{4.2}$$

5. **Filtrado de valores extremos**: Se aplican filtros de calidad:
   - Tiempo mínimo: $\Delta t > 1$ segundo (para evitar divisiones por cero y mediciones irreales)
   - Rango de velocidad: 0,1 km/h $< v <$ 200 km/h (para descartar valores erróneos)

Speed_histogram.py: Análisis poblacional Este *script* realiza un análisis agregado de velocidades para los individuos de mayor calidad:

- **Selección de individuos**: Conecta a PostgreSQL y consulta la tabla `human_trajectories` para obtener los **top 50 individuos por ranking de calidad**.
- **Extracción de datos**: Para estos individuos, extrae todos sus registros GPS del período **6-15 de noviembre de 2022** con precisión GPS < 100 metros.
- **Cálculo agregado**: Calcula velocidades para todos los pares de puntos consecutivos y las agrega en un conjunto de datos unificado.
- **Clasificación por modo de transporte**: Clasifica las velocidades en rangos estándar:

Modo de transporte	Rango (km/h)	Rango (m/s)
Peatonal	1-10	0.28-2.78
Bicicleta	11-20	3.06-5.56
Automóvil	21-150	5.83-41.67
Avión	>151	>41.94

Cuadro 4.1: Clasificación de velocidades por modo de transporte

- **Visualización**: Genera un histograma de distribución de velocidades con:
  - Bins óptimos calculados como $\sqrt{n}$ (método de Scott)
  - Líneas verticales marcando los rangos de modos de transporte

  - Anotaciones con estadísticas: media, mediana, desviación estándar
  - Escala logarítmica en el eje Y para mejor visualización

- **Salida de datos**: Guarda tres archivos:

  1. Histograma en formato PNG
  2. Conjunto de datos de velocidades en `datos_velocidades_noviembre.csv`
  3. Conjunto de datos de movilidad original en `datos_movilidad_noviembre.csv`

Speed_vector.py: Análisis individual detallado Este *script* complementario permite análisis individualizados:

- **Selección individual**: Permite al usuario seleccionar un individuo específico de los top 50.
- **Extracción de datos**: Consulta la tabla `analysis_results.individual_speeds` (calculada previamente o en tiempo real).
- **Vector de velocidades**: Genera un **vector de velocidades** (array de todas las velocidades calculadas para ese individuo).
- **Estadísticas detalladas**: Calcula:

  - Media y mediana de velocidades
  - Desviación estándar
  - Cuartiles 25 % y 75 %
  - Rango intercuartílico
  - Máximos y mínimos

- **Visualizaciones duales**: Crea dos gráficos complementarios:

  1. **Histograma individual**: Distribución de velocidades con líneas para media y mediana
  2. **Serie temporal**: Evolución de la velocidad a lo largo del tiempo

- **Salida de datos**: Guarda:

  - Vector de velocidades en `vector_velocidades_{id}.csv`
  - Estadísticas en `estadisticas_{id}.csv`
  - Gráficos en formato PNG

El cálculo de velocidades es esencial por múltiples razones técnicas y analíticas:

1. **Validación de calidad de datos**: Las velocidades imposibles o extremas (ej., > 500 km/h en un segundo) indican errores de medición GPS que deben identificarse y descartarse para mantener la integridad del conjunto de datos.
2. **Identificación del modo de transporte**: La velocidad es el principal discriminante entre modos de transporte. Mientras que las coordenadas GPS por sí solas no pueden distinguir entre caminata y movimiento vehicular, los perfiles de velocidad son característicos para cada modo:

- Caminata: $1 - 6$ km/h, variaciones suaves
- Bicicleta: $10 - 25$ km/h, más consistente
- Automóvil: $20 - 120$ km/h, con variaciones abruptas

3. **Caracterización de patrones individuales**: Las estadísticas de velocidad por individuo permiten:
   - Identificar usuarios predominantemente peatonales (velocidades consistentemente < 5 km/h)
   - Detectar usuarios con patrones mixtos (rangos variables de $5 - 50$ km/h)
   - Caracterizar usuarios predominantemente motorizados (velocidades frecuentemente > 30 km/h)

Basado en principios de física de movilidad urbana y estadísticas de transporte, se anticipan los siguientes patrones:

1. **Distribución multimodal**: El histograma general de velocidades debería mostrar múltiples picos correspondientes a los principales modos de transporte:
   - **Pico peatonal pronunciado**: $3 - 6$ km/h ($0{,}8 - 1{,}7$ m/s), correspondiente a velocidad típica de caminata humana.
   - **Pico de bicicleta/tráfico lento**: $15 - 25$ km/h ($4{,}2 - 6{,}9$ m/s), incluyendo ciclistas y vehículos en congestión.
   - **Pico de automóvil**: $40 - 80$ km/h ($11{,}1 - 22{,}2$ m/s), correspondiente a vehículos en vías rápidas.
2. **Frecuencia relativa**: Se espera que las velocidades peatonales sean las más frecuentes (mayor área bajo la curva en ese rango), seguida por velocidades vehiculares, con velocidades extremas (> 150 km/h) siendo muy raras.
3. **Estadísticas poblacionales**:
   - Media general: $10 - 20$ km/h ($2{,}8 - 5{,}6$ m/s), reflejando la mezcla de modos de transporte.
   - Mediana: Posiblemente menor que la media ($6 - 12$ km/h), debido a la asimetría de la distribución.
   - Desviación estándar: Alta ($8 - 15$ km/h), indicando gran variabilidad entre individuos y modos.
4. **Heterogeneidad individual**: Los análisis por individuo deberían revelar:
   - Individuos **exclusivamente peatonales**: Distribución unimodal centrada en $3 - 6$ km/h.
   - Individuos con **uso mixto**: Distribuciones bimodales o multimodales.
   - Individuos **predominantemente motorizados**: Distribuciones sesgadas hacia altas velocidades.

Los resultados del análisis de velocidades tienen importantes implicaciones para el desarrollo de modelos de movilidad:

1. **Segmentación de usuarios**: Permite clasificar automáticamente a los individuos según su modo de transporte predominante, lo que habilita modelos específicos para cada grupo.
2. **Validación de trayectorias**: Proporciona un mecanismo para identificar y filtrar trayectorias erróneas o no representativas.
3. **Estimación de tiempos de viaje**: Las velocidades calculadas permiten estimar tiempos realistas de desplazamiento entre puntos.
4. **Calibración de parámetros**: Los estadísticos de velocidad sirven como parámetros de entrada para modelos de simulación de movilidad.
5. **Identificación de anomalías**: Permite detectar comportamientos atípicos (ej., velocidades inconsistentes con el entorno urbano).

La combinación del análisis poblacional (speed_histogram.py) y el análisis individual detallado (speed_vector.py) proporciona una comprensión completa de las dinámicas de velocidad en el dataset, identificando tanto patrones agregados como variabilidad individual, ambos esenciales para construir modelos de movilidad robustos y representativos.

4.3.3 Aplicación de filtro peatonal

La identificación y aislamiento de movimiento puramente peatonal es fundamental para el objetivo específico de modelar movilidad a pie. Esta funcionalidad está implementada dentro del *script* `pedestrian_trajectories.py` (descrito en la sección 4.2.6), específicamente en los métodos `calculate_speeds_for_trajectory()` y `segment_trajectory_by_speed()`.

Parámetros biomecánicos del filtro

El filtro peatonal se basa en parámetros estadísticos derivados de estudios de biomecánica humana y análisis de velocidad peatonal:

Párametro	Valor en m/s	Valor en km/h
Velocidad media peatonal	1.34	4.82
Desviación estándar	0.37	1.33
Límite inferior (media - 2σ)	0.6	2.16
Límite superior (media + 2σ)	2.08	7.49
Rango aceptable completo	0.60-2.08	2.16-7.49

Cuadro 4.2: Parámetros estadísticos del filtro peatonal

Estos parámetros fueron seleccionados para cubrir el espectro completo de velocidades de caminata humana:

- **Límite inferior (0.6 m/s)**: Incluye caminata muy lenta, característica de niños pequeños, adultos mayores, personas con movilidad reducida, o individuos caminando en multitudes densas.
- **Velocidad media (1.34 m/s)**: Corresponde a la velocidad típica de marcha normal para adultos saludables en condiciones urbanas estándar.
- **Límite superior (2.08 m/s)**: Captura caminata rápida, como personas apuradas, deportistas, o individuos caminando sin obstáculos.

El uso de ± 2 desviaciones estándar alrededor de la media asegura que el filtro capture aproximadamente el 95 % de la variabilidad natural de la velocidad peatonal en condiciones normales, siguiendo el principio estadístico de la distribución normal. El proceso de filtrado peatonal sigue una lógica secuencial rigurosa:

1. **Cálculo de velocidades**: Para cada trayectoria individual, se calculan las velocidades instantáneas entre todos los puntos consecutivos usando la fórmula de Haversine (descrita en la sección anterior).
2. **Clasificación binaria**: Cada punto (excepto el primero) se clasifica como:
   - **Válido**: Si la velocidad calculada al llegar a ese punto está dentro del rango peatonal ($0{,}6 \leq v \leq 2{,}08$ m/s).
   - **Inválido**: Si la velocidad está fuera del rango peatonal.
3. **Detección de rupturas**: Se identifican transiciones entre modos de transporte cuando:
   - Un punto válido es seguido por un punto inválido (inicio de segmento motorizado).
   - Un punto inválido es seguido por un punto válido (inicio de segmento peatonal).
4. **Segmentación**: La trayectoria completa se divide en múltiples segmentos basados en estas rupturas, creando fragmentos continuos donde todos los puntos son válidos (segmentos peatonales) o inválidos (segmentos no peatonales).
5. **Filtrado final**: Se aplican criterios de calidad:
   - Solo se conservan segmentos peatonales.
   - Se descartan segmentos con menos de 2 puntos (un solo punto no puede formar una trayectoria).
   - Se eliminan completamente segmentos donde todas las velocidades están fuera del rango peatonal.

La implementación de este filtro específico se justifica por múltiples razones técnicas y analíticas:

1. **Especificidad del modelo**: El objetivo del proyecto es modelar específicamente **movilidad peatonal**, no desplazamientos multimodales generales. In-

cluir segmentos de otros modos de transporte introduciría patrones cinemáticos incompatibles que degradarían la calidad del modelo.

2. **Robustez estadística**: El filtro por velocidad es más robusto que alternativas como:
   - Filtrado por distancia fija: No considera diferencias temporales.
   - Filtrado por tiempo fijo: No considera diferencias espaciales.
   - Filtrado por aceleración: Más complejo y propenso a ruido en datos GPS.
3. **Habilidad para manejar multimodalidad**: Muchos individuos utilizan múltiples modos de transporte en un mismo día (ej., caminar a la estación, tomar metro, caminar a la oficina). El algoritmo identifica y aísla automáticamente solo las fases peatonales.

Basado en estudios de movilidad urbana y patrones de desplazamiento, se anticipan los siguientes resultados:

1. **Reducción de volumen de datos**: Se espera que el filtro elimine entre el **60-80 %** de los puntos GPS originales de cada individuo, reflejando que:
   - La mayoría de las personas usan transporte motorizado para trayectos largos.
   - Los períodos de inactividad (en casa, en la oficina) generan puntos estáticos no peatonales.
   - Muchos desplazamientos cortos también pueden realizarse en vehículos.
2. **Calidad de segmentos resultantes**: Los segmentos peatonales identificados deberían exhibir:
   - **Coherencia espacial**: Rutas continuas sin saltos geográficos bruscos.
   - **Consistencia de velocidad**: Variaciones suaves dentro del rango peatonal, sin cambios abruptos.
   - **Sentido físico**: Representar viajes a pie plausibles (ej., çasa → estación de metro", .ºficina → restaurante", "parque de estacionamiento → centro comercial").
3. **Distribución por individuo**: El número de segmentos peatonales por persona debería correlacionar positivamente con:
   - Número de días con datos (mayor cobertura temporal → más segmentos).
   - Nivel de actividad general del individuo.
   - Hábitos de movilidad (personas que caminan más vs. personas que conducen más).
4. **Patrones temporales**: Se espera que los segmentos peatonales se concentren en:
   - Horas pico de desplazamientos (mañana y tarde).

- Trayectos de conexión entre modos de transporte.
- Actividades recreativas (caminatas en parques, centros comerciales).

Para asegurar la calidad del filtrado, se implementan múltiples mecanismos de validación:

1. **Verificación estadística**: Análisis posterior de las velocidades en segmentos etiquetados como peatonales para confirmar que se mantienen dentro del rango esperado.
2. **Consistencia temporal**: Verificación de que los segmentos tienen duraciones plausibles para caminata (generalmente 1-30 minutos para desplazamientos urbanos típicos).
3. **Coherencia espacial**: Análisis de la densidad de puntos en segmentos peatonales para detectar posibles errores de segmentación.
4. **Comparación con patrones conocidos**: Contraste de los segmentos identificados con patrones típicos de viajes peatonales documentados en literatura de movilidad urbana.

El filtrado peatonal tiene importantes implicaciones para las etapas posteriores del proyecto:

1. **Calidad del conjunto de datos**: Crea un conjunto de datos especializado de alta calidad para entrenamiento de modelos específicos de movilidad peatonal.
2. **Reducción de complejidad**: Elimina la heterogeneidad de modos de transporte, simplificando el análisis de patrones.
3. **Enfoque analítico**: Permite concentrar el análisis en las características específicas de la caminata (velocidad, patrones de pausa, preferencias de ruta).
4. **Validez de conclusiones**: Asegura que las conclusiones sobre patrones de movilidad se refieran específicamente a comportamiento peatonal, no a mezclas indeterminadas de modos.
5. **Comparabilidad**: Facilita la comparación con estudios previos específicos de movilidad peatonal.

La implementación de este filtro basado en parámetros biomecánicos robustos asegura que el análisis posterior se centre exclusivamente en movilidad peatonal válida, proporcionando una base sólida para el desarrollo de modelos precisos y representativos de comportamiento a pie en entornos urbanos.

4.3.4 Visualización geográfica

La visualización geográfica es una etapa crítica en el análisis de datos de movilidad, ya que permite transformar coordenadas numéricas abstractas en representaciones espaciales intuitivas que facilitan la validación, exploración y comprensión de patrones de movilidad. Esta sección emplea tres *scripts* complementarios que generan visualizaciones interactivas y estáticas de diferentes niveles de agregación.

Show_coordinates.py: Visualización individual interactiva

Este script genera mapas interactivos individuales utilizando la biblioteca `Folium`, permitiendo una exploración detallada de las trayectorias de usuarios específicos:

1. **Selección de individuo**: Conecta a la base de datos PostgreSQL y permite al usuario seleccionar un `identifier` específico para análisis detallado.
2. **Extracción temporal**: Recupera todas las coordenadas GPS del individuo para el período del **6 al 15 de noviembre de 2022**, un intervalo temporal significativo que captura patrones semanales.
3. **Codificación temporal por color**: Asigna un color distintivo a cada día según el esquema:
4. **Construcción del mapa**:
    - Crea un mapa centrado automáticamente en el área de actividad del individuo.
    - Dibuja una **polilínea** conectando todos los puntos en orden cronológico, visualizando la trayectoria completa.
    - Agrega marcadores en cada punto GPS con *popups* interactivos que muestran:
        - Coordenadas exactas (latitud, longitud)
        - *Timestamp* completo
        - Día de la semana y fecha
        - Velocidad calculada (si está disponible)
5. **Capas adicionales**: Para trayectorias con más de 10 puntos, agrega una capa de **mapa de calor (heatmap)** superpuesta que visualiza la densidad espacial de la actividad.
6. **Leyenda temporal**: Incluye una leyenda interactiva en la esquina superior derecha que muestra el código de colores por día, facilitando la interpretación de patrones temporales.
7. **Salida**: Guarda el mapa como archivo HTML interactivo en la carpeta `maps/`, permitiendo su visualización en cualquier navegador web.

City_distribution_map.py: Visualización agregada multi-usuario

Este script crea una visualización agregada que muestra múltiples usuarios simultáneamente, revelando patrones poblacionales:

1. **Carga de datos**: Carga el dataset `pedestrian_trajectories_all.csv` que contiene todos los segmentos peatonales filtrados y validados.
2. **Asignación de colores**: Identifica todos los usuarios únicos y asigna un color distintivo a cada uno (hasta 20 usuarios para evitar saturación visual).

3. **Construcción del mapa base**: Crea un mapa centrado en el centroide geográfico de todos los puntos, asegurando que toda el área de actividad sea visible.
4. **Visualización por usuario**:
   - Para cada usuario, dibuja todos sus puntos como **círculos pequeños** con el color asignado.
   - Conecta los puntos de cada segmento peatonal con **polilíneas** del mismo color.
   - Esto mantiene la identidad individual mientras muestra trayectorias completas.
5. **Optimización de rendimiento**: Implementa un **MarkerCluster** que agrupa automáticamente puntos cercanos en áreas densas, mejorando el rendimiento de visualización sin perder información.
6. **Control de capas**: Crea capas (`FeatureGroups`) separadas para cada usuario y agrega un control de capas interactivo que permite mostrar u ocultar usuarios individualmente, facilitando el análisis comparativo.
7. **Salida**: Guarda el resultado como `mapa_ciudades_trayectorias.html`, un mapa interactivo completo para exploración poblacional.

Graph.py: Visualizaciones estáticas multi-propósito

Este script genera un conjunto completo de visualizaciones estáticas utilizando `Matplotlib` y `Seaborn`, enfocándose en diferentes dimensiones analíticas:

Visualización	Tipo de gráfico	Insights proporcionados
Trayectorias individuales	Scatter plot con líneas	Patrones de rutas individuales, consistencia de desplazamientos, identificación de puntos fijos (hogar/trabajo)
Densidad temporal	Scatter plot con color-map	Patrones horarios de actividad, concentración de movimientos por hora del día
Actividad por hora	Histograma de barras	Picos de actividad horaria, identificación de horas pico de movilidad
Actividad por día	Histograma comparativo	Diferencias de actividad entre días laborales y fines de semana, patrones semanales
Distribución de velocidad	Histograma de densidad	Validación del filtro peatonal, distribución típica de velocidades de caminata
Usuarios más activos	Gráfico de barras horizontales	Identificación de individuos con mayor cobertura para análisis detallado

Cuadro 4.3: Tipos de visualizaciones estáticas generadas

La inversión en desarrollo de capacidades de visualización se justifica por múltiples razones críticas:

1. **Validación de calidad de datos**: Los mapas permiten identificar rápidamente anomalías que serían difíciles de detectar en análisis tabular:
   - Puntos GPS erróneos (ej., coordenadas en medio del océano o fuera del área de estudio).
   - Trayectorias físicamente imposibles (saltos espaciales bruscos que indican errores de medición).
   - Problemas de segmentación (rupturas incorrectas en trayectorias continuas).
2. **Detección de patrones espaciales**: Las visualizaciones revelan insights sobre:
   - Áreas de alta concentración (nodos de actividad).
   - Rutas comunes y preferencias de trayectoria.
   - Relaciones espaciales entre puntos de origen y destino.
   - Distribución geográfica de la población estudiada.
3. **Exploración interactiva**: Los mapas HTML permiten a los investigadores:
   - Hacer zoom en áreas de interés específico.
   - Examinar puntos individuales con información detallada.
   - Comparar visualmente múltiples usuarios o períodos temporales.
   - Identificar correlaciones espaciales con características urbanas.
4. **Comunicación de resultados**: Las visualizaciones estáticas proporcionan un medio efectivo para comunicar hallazgos en presentaciones, publicaciones y reportes técnicos.

Basado en principios de movilidad urbana y análisis espacial, se anticipan los siguientes patrones en las visualizaciones:

1. **Mapas individuales**:
   - **Clusters espaciales**: Concentraciones de puntos en ubicaciones fijas que probablemente corresponden a hogar, lugar de trabajo, u otros puntos de interés frecuentados.
   - **Trayectorias lineales**: Rutas claras de desplazamiento entre clusters, siguiendo generalmente la red vial urbana.
   - **Consistencia temporal**: Uso repetido de las mismas rutas en diferentes días (indicado por superposición de colores).
   - **Variabilidad**: Diferencias entre días laborables y fines de semana en términos de destinos y horarios.
2. **Mapa agregado poblacional**:
   - **Cobertura geográfica**: Distribución concentrada en áreas urbanas con densidad poblacional.

- **Corredores de movilidad**: Líneas visibles siguiendo avenidas principales y rutas de transporte.
- **Nodos de actividad**: Puntos de alta densidad correspondientes a centros comerciales, estaciones de transporte, zonas de empleo.

3. **Visualizaciones temporales**:

   - **Patrones horarios**: Picos de actividad en horas típicas de desplazamiento (7-9 AM para viajes al trabajo, 5-7 PM para retorno).
   - **Variación semanal**: Mayor actividad en días laborables (lunes a viernes) comparado con fines de semana.
   - **Distribución de velocidades**: Concentración en el rango peatonal (1-6 km/h) con distribución aproximadamente normal, confirmando la efectividad del filtrado.

4. **Gráficos de actividad**:

   - **Distribución de usuarios**: Pocos usuarios con mucha actividad (cola larga en la distribución).
   - **Correlaciones**: Relación positiva entre número de días con datos y número de segmentos peatonales identificados.

Las capacidades de visualización desarrolladas tienen importantes implicaciones para el análisis posterior:

1. **Validación iterativa**: Permiten verificar resultados de cada etapa del pipeline de procesamiento, facilitando correcciones tempranas.
2. **Hipótesis generación**: Los patrones visuales observados pueden sugerir nuevas hipótesis sobre comportamiento de movilidad.
3. **Contextualización espacial**: Proporcionan contexto geográfico para interpretar resultados estadísticos.
4. **Selección de casos de estudio**: Identifican individuos representativos o interesantes para análisis en profundidad.
5. **Evaluación de cobertura**: Permiten evaluar la representatividad geográfica del dataset y posibles sesgos espaciales.

La combinación de visualizaciones interactivas (para exploración detallada) y estáticas (para análisis comparativo y comunicación) proporciona un conjunto completo de herramientas para comprender y analizar los patrones de movilidad peatonal desde múltiples perspectivas espaciales y temporales.

4.3.5 Implementación de algoritmo de *clustering*

El análisis de *clustering* mediante el algoritmo K-Means se implementa para descubrir patrones y tipologías naturales en las trayectorias peatonales sin la necesidad de etiquetas predefinidas. El *script* `KMeans.py` implementa un *pipeline* completo de análisis no supervisado sobre las trayectorias peatonales filtradas.

Preparación de datos

El proceso inicia con la preparación meticulosa de los datos de entrada:

1. **Carga de datasets**: Se cargan dos archivos complementarios:
   - `pedestrian_trajectories_all.csv`: Contiene las trayectorias peatonales filtradas.
   - `trajectory_classification.csv`: Proporciona las métricas de calidad calculadas previamente.
2. **Selección de características**: Se seleccionan 14 características numéricas relevantes para el análisis de clustering:

Característica	Descripción
records_count	Volumen total de registros GPS del individuo
avg_accuracy_meters	Precisión promedio del GPS en metros
movement_points	Cantidad de puntos con desplazamiento significativo
time_span_days	Duración temporal total de la trayectoria
active_days_count	Número de días con actividad registrada
spatial_range	Amplitud geográfica total del área recorrida
score_volume	Componente del score de calidad: volumen de datos
score_duration	Componente del score: duración temporal
activity_ratio	Proporción de días activos respecto al período total
score_regularity	Componente del score: regularidad
score_accuracy	Componente del score: precisión GPS
score_mobility	Componente del score: movilidad
score_diversity	Componente del score: diversidad espacial
quality_score	Score compuesto final de calidad (0-100)

Cuadro 4.4: Características utilizadas para el análisis de clustering

3. **Limpieza de datos**: Se manejan valores nulos rellenándolos con la media de cada columna correspondiente, preservando la integridad estadística del dataset.
4. **Estandarización**: Todas las características se estandarizan utilizando `StandardScaler` para transformarlas a una distribución con media 0 y desviación estándar 1, un requisito esencial para el algoritmo K-Means que es sensible a la escala de las variables.

Análisis exploratorio preliminar

Antes de aplicar el clustering, se realiza un análisis exploratorio para comprender la estructura de los datos:

1. **Matriz de correlación**: Se genera un **heatmap** de correlación que visualiza las relaciones lineales entre las 14 características, identificando posibles

multicolinealidades.

2. **Distribuciones de características**: Se crean histogramas de las 4 características más importantes:
   - `quality_score`: Score compuesto de calidad
   - `movement_points`: Cantidad de movimientos
   - `spatial_range`: Diversidad espacial
   - `active_days_count`: Días activos
3. **Guardado de visualizaciones**: Los resultados se almacenan en:
   - `pedestrian_analysis/correlation_matrix.png`
   - `pedestrian_analysis/feature_distributions.png`

Determinación del número óptimo de clusters

Para determinar el número adecuado de clusters (k), se implementan dos métodos complementarios:

Método del codo (Elbow Method)

- Se ejecuta K-Means con valores de k desde 1 hasta 10.
- Para cada k, se calcula la **inercia**: suma de las distancias cuadradas de cada punto a su centroide asignado.
- Se grafica la inercia en función de k. El punto donde la curva forma un çodo"(reducción marginal en inercia al aumentar k) indica el número óptimo de clusters.

Método del coeficiente de Silhouette

- Para cada k desde 2 hasta 10, se calcula el **Silhouette Score**, una métrica que mide:
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{4.3}$$
donde $a(i)$ es la distancia promedio del punto i a otros puntos en su mismo cluster, y $b(i)$ es la distancia promedio a puntos en el cluster más cercano.
- Un score cercano a 1 indica clusters bien separados, mientras que scores cercanos a 0 o negativos sugieren solapamiento entre clusters.

Ejecución del algoritmo K-Means

Una vez determinado el número óptimo de clusters:

1. **Aplicación del algoritmo**: Se ejecuta K-Means con el valor k seleccionado, utilizando inicialización `k-means++` para una convergencia más rápida y estable.

2. **Asignación de clusters**: Cada trayectoria se asigna al cluster cuyo centroide es más cercano en el espacio de características de 14 dimensiones.
3. **Evaluación de calidad**: Se calcula el Silhouette Score global para evaluar la calidad del agrupamiento resultante.
4. **Integración de resultados**: Se agrega una columna `cluster` al dataset de clasificación con las asignaciones correspondientes.

Análisis e interpretación de clusters

Para cada cluster identificado, se realiza un análisis interpretativo detallado:

1. **Estadísticas descriptivas**: Se calculan para cada característica:
   - Media y desviación estándar
   - Valores mínimos y máximos
   - Cuartiles 25 %, 50 % (mediana), y 75 %
2. **Caracterización cualitativa**: Cada cluster se caracteriza según umbrales predefinidos:

Dimensión	ALTA	MEDIA	BAJA
Calidad (`quality_score`)	> 80	$70 - 80$	< 70
Movilidad (`spatial_range`)	$> 2{,}0$	$0{,}5 - 2{,}0$	$< 0{,}5$
Actividad (`movement_points`)	> 500	$300 - 500$	< 300
Persistencia (`active_days`)	> 7	$3 - 7$	< 3

Cuadro 4.5: Umbrales para caracterización cualitativa de clusters

3. **Resumen interpretativo**: Se genera un resumen descriptivo para cada cluster que incluye:
   - Tamaño del cluster (número de individuos)
   - Perfil característico
   - Posibles interpretaciones comportamentales
   - Recomendaciones para análisis posterior

El *pipeline* genera los siguientes archivos de salida:

1. `clustering_results.csv`: Dataset completo con las asignaciones de cluster para cada individuo.
2. `cluster_summary.csv`: Características promedio y estadísticas por cluster.
3. `clustering_report.txt`: Resumen interpretativo en formato de texto.
4. Archivos PNG con todas las visualizaciones generadas.

La implementación de *clustering* se justifica por múltiples razones analíticas:

1. **Descubrimiento de patrones naturales**: Permite identificar tipologías de usuarios que emergen naturalmente de los datos, sin sesgos de categorización predefinida.
2. **Validación independiente del *score* de calidad**: Si los *clusters* se alinean consistentemente con los *scores* de calidad, valida que las métricas capturan diferencias reales en patrones de movilidad.
3. **Identificación de subgrupos ocultos**: Puede revelar grupos que el *score* compuesto no discrimina (ej., usuarios con alta calidad pero patrones de movilidad cualitativamente diferentes).
4. **Reducción de dimensionalidad**: El *clustering* simplifica la complejidad del conjunto de datos agrupando individuos similares, facilitando análisis posteriores.
5. **Caracterización comprehensiva**: Proporciona una taxonomía basada en datos para clasificar diferentes tipos de usuarios de movilidad.

Basado en principios de análisis de *cluster* y patrones típicos de movilidad, se anticipan los siguientes resultados:

1. **Número óptimo de *clusters***: Se espera que los métodos del codo y Silhouette sugieran entre **3-5 *clusters***. Menos de 3 indicaría subagrupamiento, mientras que más de 5 probablemente representaría sobreajuste.
2. **Calidad del agrupamiento**: El Silhouette Score debería estar en el rango **0.3-0.6**, indicando una separación moderada pero significativa (*scores* $> 0{,}7$ son raros en datos reales, $< 0{,}2$ indica agrupamiento pobre).
3. **Separación visual**: Los *clusters* deberían diferenciarse claramente en la visualización PCA 2D, mostrando agrupaciones distintas con mínima superposición.
4. **Tipologías esperadas**: Se anticipa identificar *clusters* como:

Cluster	**Características**	**Interpretación**
Cluster 1	Alta calidad, alta movilidad, muchos días activos	Usuarios rutinarios ideales para modelado
Cluster 2	Calidad media, movilidad moderada, actividad intermitente	Usuarios regulares con patrones menos consistentes
Cluster 3	Baja calidad, baja movilidad, pocos días	Usuarios ocasionales o datos limitados
Cluster 4	Alta calidad pero baja diversidad espacial	Usuarios con rutinas muy fijas y repetitivas

Cuadro 4.6: Tipologías de clusters esperadas

5. **Correlaciones internas**: Se espera alta correlación entre características dentro de cada cluster y diferencias estadísticamente significativas entre clusters, confirmando la validez del agrupamiento.

6. **Distribución de tamaños**: Probablemente un cluster dominante (usuarios ocasionales) que contenga la mayoría de los individuos, y clusters más pequeños pero cualitativamente distintos para usuarios especializados.

Los resultados del *clustering* tienen importantes implicaciones:

1. **Segmentación para modelado**: Permite desarrollar modelos específicos para cada tipo de usuario, mejorando la precisión predictiva.
2. **Priorización de análisis**: Identifica los *clusters* más valiosos para análisis en profundidad (ej., usuarios rutinarios de alta calidad).
3. **Validación de métricas**: Confirma que las métricas de calidad capturan dimensiones relevantes de variabilidad en los datos.
4. **Descubrimiento de patrones**: Puede revelar patrones inesperados o contra-intuitivos en los datos de movilidad.
5. **Base para análisis comparativo**: Proporciona un marco para comparar diferentes subpoblaciones en términos de comportamiento de movilidad.

La implementación de este *pipeline* completo de *clustering* proporciona una herramienta poderosa para descubrir y caracterizar patrones naturales en los datos de movilidad peatonal, complementando y validando los análisis supervisados basados en *scores* de calidad predefinidos.

4.4 Determinación de tiempos de pausa

El análisis de tiempos de pausa es fundamental para modelar el comportamiento estacionario entre desplazamientos peatonales. El script `wait_time.py` implementa un algoritmo para identificar, medir y analizar sistemáticamente las pausas en las trayectorias peatonales filtradas.

Preparación de datos y segmentación temporal

El proceso inicia con una preparación cuidadosa de los datos y su segmentación en sesiones temporales coherentes:

1. **Carga de datos**: Se carga el dataset `pedestrian_trajectories_all.csv` que contiene todas las trayectorias peatonales previamente filtradas y validadas.
2. **Ordenamiento cronológico**: Los datos se ordenan secuencialmente por `identifier` y `timestamp` para asegurar un procesamiento temporal coherente.
3. **Identificación de sesiones**: Para cada usuario individualmente, se analiza la secuencia temporal de registros GPS para identificar **sesiones de actividad**:
   - Se definen **rupturas de sesión** cuando existe un intervalo temporal mayor a `max_time_gap_minutes` (por defecto 120 minutos) entre puntos consecutivos.
   - Esto segmenta la trayectoria en sesiones discretas temporalmente, por ejemplo: "Sesión mañana: 8:00-12:30", "Sesión tarde: 14:00-19:00".
   - La segmentación en sesiones es esencial para distinguir entre pausas intradía (ej., descansos breves) y períodos de inactividad prolongada (ej., noche, fines de semana).

Detección de movimiento y clasificación de puntos

Dentro de cada sesión, se implementa un algoritmo de detección de movimiento basado en umbrales espaciales:

1. **Cálculo de distancia**: Para cada par de puntos consecutivos dentro de una sesión, se calcula la distancia real utilizando la **fórmula de Haversine**.
2. **Umbral de movimiento**: Se define un umbral de distancia configurable (`movement_threshold_km = 0.05 km = 50 metros`) para clasificar cada punto:
3. **Agrupación de puntos en pausa**: Se identifican secuencias continuas de puntos clasificados como .$^{\text{en}}$ pausaz se agrupan en **ubicaciones de pausa** coherentes. El algoritmo:
   - Inicia una nueva pausa al detectar el primer punto con $d < 50$ m.
   - Continúa agregando puntos mientras la distancia entre consecutivos permanezca bajo el umbral.

Condición	Clasificación	Interpretación
$d < 50$ m	**Pausa**	La persona está esencialmente detenida en la misma ubicación o moviéndose mínimamente dentro de un radio pequeño.
$d \geq 50$ m	**Movimiento**	La persona se está desplazando significativamente entre mediciones.

Cuadro 4.7: Clasificación de puntos basada en umbral de distancia

- Finaliza la pausa al detectar $d \geq 50$ m, marcando el inicio de un nuevo movimiento.

Cálculo de métricas de pausa

Para cada pausa identificada, se calcula un conjunto completo de métricas descriptivas:

Métrica	Descripción
`pause_start`	Timestamp del último punto antes de que la persona se detenga (inicio de la pausa)
`pause_end`	Timestamp del primer punto al reanudar el movimiento (fin de la pausa)
`pause_duration_min`	Duración total de la pausa en minutos: ($pause\_end - pause\_start$)
`pause_lat, pause_lon`	Coordenadas promedio de todos los puntos durante la pausa
`movement_before_points`	Número de puntos GPS en el movimiento inmediatamente anterior a la pausa
`movement_after_points`	Número de puntos GPS en el movimiento inmediatamente posterior a la pausa
`session_id`	Identificador de la sesión a la que pertenece la pausa

Cuadro 4.8: Métricas calculadas para cada pausa identificada

Filtrado de calidad : Se aplican filtros de calidad:

- Solo se consideran pausas con `pause_duration_min` > 0, descartando transiciones instantáneas que podrían ser artefactos de medición.
- Las pausas se asocian al contexto de su sesión correspondiente, permitiendo análisis por contexto temporal.

Análisis estadístico exhaustivo

El script realiza un análisis estadístico completo en múltiples dimensiones:

Estadísticas globales

- Número total de pausas detectadas en todo el dataset.
- Estadísticas de duración: media, mediana, desviación estándar, mínima y máxima.
- Distribución por categorías de duración: < 5 min, $5 - 15$ min, $15 - 30$ min, > 30 min.
- Número de usuarios únicos con pausas detectadas y número total de sesiones.

Patrones temporales

- **Distribución horaria**: Frecuencia de pausas por hora del día (0-23), revelando horarios típicos de actividad estacionaria.
- **Duración por hora**: Duración promedio de pausas por hora, identificando si ciertos horarios tienen pausas más prolongadas.
- **Distribución por usuario**: Número de pausas por individuo, caracterizando patrones personales de actividad.

Relaciones exploratorias

- **Duración de pausa vs. actividad previa**: Correlación entre la duración de una pausa y la extensión del movimiento inmediatamente anterior.
- **Patrones espacio-temporales**: Relación entre ubicación geográfica de la pausa y su duración/horario.

Visualización comprehensiva

El script genera dos figuras principales con múltiples subplots para visualización comprehensiva:

Figura 1: Análisis de duración de pausas

1. **Histograma de distribución**: Frecuencia de pausas por duración (todas las pausas).
2. **Histograma zoom**: Enfocado en pausas cortas (< 60 minutos) para mayor detalle.
3. **Distribución logarítmica**: Histograma con escala logarítmica para visualizar todo el rango de duraciones.
4. **Boxplot**: Visualización de estadísticas descriptivas (mediana, cuartiles, outliers).
5. **Gráfico de barras por categorías**: Porcentaje de pausas en cada categoría de duración con anotaciones.
6. **Función de distribución acumulativa (CDF)**: Porcentaje de pausas que duran menos de X minutos.

Figura 2: Análisis temporal y contextual

1. **Pausas por hora del día**: Histograma de frecuencia de pausas por hora.
2. **Duración promedio por hora**: Gráfico de barras mostrando duración media por hora.
3. **Distribución por usuario**: Histograma de número de pausas por persona.
4. **Scatter plot**: Duración de pausa vs. puntos en movimiento previo.

Salida de resultados

El pipeline genera múltiples archivos de salida para análisis posterior:

Archivo	Contenido
`pause_times_detailed.csv`	Detalles completos de cada pausa individual con todas las métricas calculadas
`session_data.csv`	Información de cada sesión: inicio, fin, duración, número de pausas
`pause_statistics_summary.csv`	Resumen estadístico global de todas las pausas
`top_100_longest_pauses.csv`	Las 100 pausas más largas detectadas para análisis de casos extremos
`pause_analysis_visualization.png`	Figuras con todos los subplots de visualización

Cuadro 4.9: Archivos de salida del análisis de tiempos de pausa

El análisis de tiempos de pausa es fundamental por múltiples razones:

1. **Modelado realista de comportamiento**: Los tiempos de pausa representan el comportamiento estacionario entre desplazamientos, esencial para modelos de movilidad realistas.
2. **Distinción de tipos de paradas**: Permite diferenciar entre:
   - **Paradas breves (1-5 min)**: Semáforos, cruces peatonales, esperas cortas.
   - **Paradas significativas (30+ min)**: Trabajo, comidas, actividades sociales.
3. **Validación de segmentación**: Pausas muy largas (> 2 horas) pueden indicar que deberían considerarse como sesiones separadas, validando la segmentación temporal.
4. **Enriquecimiento de modelos**: Los patrones de pausa proporcionan información temporal valiosa para enriquecer modelos de movilidad con comportamientos realistas.
5. **Caracterización de actividades**: Diferentes tipos de actividades (laborales, recreativas, domésticas) tienen patrones característicos de pausa.

Basado en principios de movilidad urbana y comportamiento humano, se anticipan los siguientes patrones:

Distribución multimodal de duraciones

- **Pausas muy cortas (< 5 min)**: Deberían ser muy frecuentes (40-60 % del total), representando semáforos, cruces peatonales, paradas breves en tiendas.
- **Pausas cortas (5-15 min)**: Frecuencia moderada (20-30 %), correspondiendo a compras rápidas, cafés, esperas breves.
- **Pausas medias (15-60 min)**: Menos frecuentes (10-20 %), representando comidas, reuniones cortas, actividades recreativas breves.
- **Pausas largas (> 60 min)**: Raras pero importantes (5-10 %), correspondiendo a trabajo, hogar, eventos prolongados.

Patrones temporales esperados

- **Distribución horaria**: Picos de pausas alrededor de horarios de comida (13:00-15:00) y en horas laborales típicas.
- **Variación diaria**: Mayor frecuencia de pausas en días laborables comparado con fines de semana.
- **Actividad nocturna**: Pausas nocturnas (23:00-6:00) probablemente subrepresentadas debido a limitaciones en la recolección de datos (dispositivos en modo ahorro, personas durmiendo).

Características de usuarios

- **Número de pausas por día**: La mayoría de usuarios deberían mostrar 5-20 pausas por día, reflejando un número razonable de paradas durante actividades diarias.
- **Heterogeneidad individual**: Variabilidad significativa entre usuarios en términos de frecuencia y duración de pausas, reflejando diferentes estilos de vida y patrones de actividad.

Implicaciones para el modelado de movilidad

Los resultados del análisis de tiempos de pausa tienen importantes implicaciones:

1. **Calibración de modelos**: Los parámetros de distribución de tiempos de pausa pueden calibrar modelos de simulación de movilidad.
2. **Segmentación de actividades**: Las categorías de duración pueden usarse para inferir tipos de actividades.
3. **Validación de calidad**: Patrones anómalos (ej., pausas físicamente imposibles) pueden indicar errores en los datos o en el procesamiento.
4. **Identificación de puntos de interés**: Concentraciones de pausas en ubicaciones específicas pueden identificar puntos de interés importantes.
5. **Análisis comparativo**: Permite comparar patrones de pausa entre diferentes grupos demográficos o áreas geográficas.

4.5 Determinación de longitudes de vuelo

El análisis de las distancias recorridas entre puntos de recorrido consecutivos, conocidas como **longitudes de vuelo**, es esencial para caracterizar cuantitativamente los patrones de desplazamiento peatonal. El script `distribution.py` implementa un análisis estadístico exhaustivo de estas distancias, incluyendo ajuste de distribuciones probabilísticas y análisis de patrones de movilidad.

Cálculo de longitudes de trayectoria

El proceso inicia con el cálculo de métricas espaciales para cada individuo:

1. **Carga de datos**: Se carga el dataset `pedestrian_trajectories_all.csv` que contiene todas las trayectorias peatonales previamente filtradas y validadas.
2. **Procesamiento individual**: Para cada `identifier` único:
   - Se extraen todos los registros GPS ordenados cronológicamente.
   - Se calcula la desviación estándar de latitudes (σ_{lat}) y longitudes (σ_{lon}).
   - Se estima una longitud de trayectoria aproximada mediante:

     $$L_{aproximada} = (\sigma_{lat} + \sigma_{lon}) \times 111 km/grado \tag{4.4}$$

     donde el factor 111 km/grado corresponde a la conversión aproximada de grados a kilómetros.
   - Se cuenta el número total de puntos GPS en la trayectoria.
   - Se calcula el span temporal total como la diferencia entre el primer y último `timestamp`.
3. **Generación de dataset**: Se crea el dataset `trajectory_lengths.csv` donde cada fila representa un individuo con sus métricas calculadas.

Análisis de distribución básico

Se genera un análisis visual básico mediante un gráfico con 4 subplots:

Visualización	**Propósito y características**
Histograma simple	Distribución de frecuencias de longitudes en km con escala lineal en ambos ejes
Histograma normalizado	Densidad de probabilidad (área total = 1) para análisis estadístico
Boxplot	Visualización de estadísticas descriptivas: mediana, cuartiles, rango intercuartílico y outliers
Grágico Q-Q (Quantile-Quantile)	Comparación directa entre los cuantiles observados y los esperados para una distribución normal teórica

Cuadro 4.10: Visualizaciones del análisis de distribución básica

Interpretación del gráfico Q-Q :

- **Puntos alineados en la diagonal**: Indican que los datos siguen una distribución normal.
- **Desviaciones en las colas**: Sugieren colas más pesadas (curva convexa) o más ligeras (curva cóncava) que la distribución normal.
- **Sesgo visible**: Curvatura sistemática indica asimetría en la distribución.

Los resultados se guardan en `Distributions/basic_distribution_analysis.png`.

Ajuste de distribuciones teóricas

Se evalúa el ajuste de 6 distribuciones probabilísticas teóricas comunes a los datos observados:

Distribución	Parámetros	Características y aplicabilidad
Normal (Gaussiana)	μ, σ	Distribución simétrica, apropiada para fenómenos con variaciones aleatorias aditivas
Lognormal	μ, σ del log	Valores positivos con cola derecha larga, común en fenómenos multiplicativos
Exponencial	λ (tasa)	Sin memoria, apropiada para tiempos entre eventos independientes
Gamma	α (forma), β (escala)	Generaliza exponencial, flexible para datos positivos sesgados
Weibull	k (forma), λ (escala)	Extremadamente flexible, modela tasas de fallo variables
Rayleigh	σ (escala)	Caso especial de Weibull con $k = 2$, apropiada para magnitudes

Cuadro 4.11: Distribuciones teóricas evaluadas para ajuste

Métodos de ajuste y evaluación :

1. **Estimación de parámetros**: Para cada distribución, los parámetros se estiman usando **Máxima Verosimilitud (Maximum Likelihood Estimation, MLE)**.
2. **Métricas de bondad de ajuste**:
   - **Estadístico Kolmogorov-Smirnov (KS)**:

$$D_n = \sup_x |F_n(x) - F(x)| \tag{4.5}$$

   Mide la máxima diferencia absoluta entre la función de distribución empírica $F_n(x)$ y la teórica $F(x)$. Valores menores indican mejor ajuste.
   - **p-valor**: Probabilidad de observar los datos si provienen de la distribución teórica. Valores altos ($> 0{,}05$) sugieren buen ajuste.

- **Criterio de Información de Akaike (AIC)**:

$$AIC = 2k - 2\ln(\hat{L}) \tag{4.6}$$

donde k es el número de parámetros y $\hat{L}$ es la verosimilitud máxima. Valores menores indican mejor balance entre bondad de ajuste y complejidad.

Los resultados se ordenan por estadístico KS (menor a mayor) y se guardan en `distribution_fit_results.csv`. Se generan dos figuras para visualizar los resultados del ajuste:

Figura 1: Comparación de top distribuciones

- Muestra un histograma normalizado de los datos reales (en gris).
- Superpone las curvas de función de densidad de probabilidad (PDF) de las 4 mejores distribuciones con colores diferentes.
- Incluye una leyenda que indica el nombre de cada distribución y su estadístico KS correspondiente.
- Permite comparación visual directa de qué distribución se ajusta mejor a la forma general de los datos.

Figura 2: Análisis individual por distribución

- Configuración 2×2 con 4 subplots, uno para cada distribución top.
- Cada subplot muestra el histograma de datos con la curva de distribución ajustada superpuesta.
- El título incluye el nombre de la distribución, estadístico KS y p-valor.
- Anotaciones muestran los parámetros estimados específicos.

Estas visualizaciones se guardan en *best_fitted_distributions.png* y *individual_distribution_fits.png*.

Se calcula un conjunto completo de estadísticas descriptivas guardadas en *trajectory_lengths_statistics.csv*:

Estadística	**Interpretación**
Count (n)	Número total de observaciones (individuos)
Mean ($\bar{x}$)	Longitud promedio de trayectoria
Std (s)	Desviación estándar, medida de dispersión absoluta
Percentiles (25 %, 50 %, 75 %)	Cuartiles de la distribución
Min, Max	Rango completo observado
Skewness (γ_1)	Medida de asimetría: > 0 = cola derecha larga, < 0 = cola izquierda larga, $= 0$ = simétrica
Kurtosis (γ_2)	Medida de "peso"de colas y nitidez del pico: > 0 = colas pesadas y pico alto, < 0 = colas ligeras y pico bajo
CV (Coeficiente de Variación)	$s/\bar{x}$, medida de variabilidad relativa

Cuadro 4.12: Estadísticas descriptivas calculadas

Se genera una figura de cuadrícula 3×3 con 9 visualizaciones diferentes que proporcionan múltiples perspectivas sobre la distribución:

1. **Distribución básica**: Histograma estándar de frecuencias.
2. **Distribución logarítmica**: Histograma de $\log(longitud + 1)$, útil para visualizar distribuciones con amplio rango.
3. **Boxplot**: Estadísticas descriptivas visuales.
4. **Violin plot**: Combina boxplot con estimación de densidad kernel, mostrando la distribución completa.
5. **Función de distribución acumulativa empírica (ECDF)**: $F_n(x) = \frac{obs \leq x}{n}$.
6. **Estimación de densidad kernel (KDE)**: Aproximación suavizada de la PDF.
7. **Gráfico Q-Q vs. Normal**: Verificación específica de normalidad.
8. **Función de supervivencia**: $S(x) = 1 - F(x)$, en escala logarítmica en Y.
9. **Percentiles acumulativos**: Mapeo de percentiles 0-100 a valores correspondientes.

Esta figura se guarda como `comprehensive_distribution_analysis.png`. El análisis de distribución de longitudes de vuelo es fundamental por múltiples razones:

1. **Generación de trayectorias sintéticas realistas**: Un simulador puede muestrear de la distribución ajustada para generar distancias de desplazamiento plausibles en modelos generativos.
2. **Caracterización de patrones de movilidad**:
   - **Distribuciones con cola pesada (heavy-tailed)**: Indican que la mayoría de movimientos son cortos pero ocasionalmente hay desplazamientos

muy largos.
- **Distribuciones exponenciales**: Sugieren procesos sin memoria (Markovianos).
- **Distribuciones lognormales**: Indican procesos multiplicativos.

3. **Validación de modelos teóricos**: Comparación con supuestos de modelos establecidos de movilidad humana:
   - **Lévy flights**: Asumen distribuciones power-law.
   - **Random Waypoint**: Suele usar distribuciones uniformes o normales.
   - **Continuous Time Random Walks**: Emplean distribuciones específicas de paso.
4. **Identificación del tipo de movilidad**: La forma de la distribución revela características fundamentales del proceso de movimiento.

Basado en estudios previos de movilidad humana y principios de biomecánica peatonal, se anticipan los siguientes patrones:

Forma de la distribución

- **No normalidad**: La distribución normal probablemente ajustará pobremente debido a:
  - Valores no negativos (las longitudes no pueden ser negativas).
  - Presencia de cola derecha larga (right-skewed).
  - Evidencia en gráficos Q-Q mostrando desviaciones en las colas.
- **Distribuciones plausibles**: Mayor probabilidad de buen ajuste con:
  - **Lognormal**: Común en fenómenos de movilidad con efectos multiplicativos.
  - **Weibull**: Flexible y capaz de modelar diversas formas.
  - **Gamma**: Adecuada para datos positivos sesgados.
- **Distribuciones menos probables**:
  - **Exponencial**: Solo si los desplazamientos son completamente aleatorios sin memoria.
  - **Rayleigh**: Caso especial con aplicación más específica.

Estadística	**Rango esperado**	**Interpretación**
Media ($\bar{x}$)	0.5-1.5 km	Longitud típica de caminatas urbanas
Mediana	$< \bar{x}$	Distribución sesgada positivamente
Skewness (γ_1)	> 0 (positiva)	Cola derecha larga (ocasionales caminatas muy largas)
Kurtosis (γ_2)	> 0 (positiva)	Colas más pesadas que la distribución normal
CV	0.8-1.5	Alta variabilidad relativa entre individuos
Percentil 90 %	2-3 km	Solo el 10 % de trayectorias excede esta longitud

Cuadro 4.13: Resultados estadísticos esperados

El análisis exhaustivo de distribución de longitudes de vuelo proporciona una caracterización cuantitativa fundamental de los patrones de desplazamiento peatonal, estableciendo bases estadísticas sólidas para modelado y simulación de movilidad urbana.

Capítulo 5

Resultados

5.1 Resultados de Clasificación de Calidad de Trayectorias

El proceso de clasificación de calidad produjo los siguientes resultados a partir del dataset procesado:

Categoría	Cantidad	Porcentaje	Interpretación
REGULAR	83,430	64.3 %	Trayectorias con información suficiente para análisis básico
BUENA	40,373	31.3 %	Trayectorias con buena cobertura temporal y espacial
MUY BUENA	5,863	4.5 %	Trayectorias con alta calidad y consistencia
EXCELENTE	254	0.2 %	Trayectorias ideales con máxima calidad de datos
Total	**129,920**	**100 %**	**Individuos con calidad REGULAR o superior (>35 puntos)**

Cuadro 5.1: Distribución de individuos por categoría de calidad

Análisis de la distribución

- **Dataset procesado**: Total de 51,077,925 registros después de la limpieza.
- **Umbral de calidad**: Se identificaron 129,920 individuos con calidad REGULAR o superior (puntuación $\geq$35).
- **Representatividad**: Estos individuos representan aproximadamente el 2.15 % del total original (129,920 de 6,022,772 identificadores únicos).

- **Implicación**: El algoritmo de puntuación compuesta logró identificar eficientemente una fracción pequeña pero significativa de individuos con datos de calidad suficiente para análisis de movilidad.

5.2 Resultados de Extracción de Trayectorias Peatonales

La aplicación del filtro peatonal basado en rangos de velocidad (0.6 - 2.08 m/s o 2.16 - 7.49 km/h) produjo los siguientes resultados:

Métrica	Valor	Interpretación
Puntos peatonales válidos	171,461	Total de puntos GPS dentro del rango de velocidad peatonal
Usuarios únicos con trayectorias peatonales	41,882	32.5 % de los individuos clasificados
Periodo temporal cubierto	6-14 Nov 2022	9 días de actividad monitoreada
Velocidad promedio	1.17 m/s	Consistente con caminata humana normal
Velocidad máxima registrada	2.08 m/s	Límite superior del filtro aplicado

Cuadro 5.2: Resultados de la extracción de trayectorias peatonales

Análisis de reducción por filtrado

- **Reducción significativa**: De 129,920 individuos clasificados se redujo a 41,882 con trayectorias peatonales válidas, lo que representa una reducción del 67.8 %.
- **Interpretación**: Aproximadamente dos tercios de los individuos clasificados usaban predominantemente transporte motorizado o tenían patrones de movimiento fuera del rango peatonal.
- **Calidad del dataset resultante**: Los 41,882 individuos restantes representan una muestra de alta calidad para análisis de movilidad peatonal específica.

5.3 Resultados del Análisis de Clustering (K-Means)

5.3.1 Configuración y distribución general

Se aplicó el algoritmo K-Means con $k = 4$ clusters sobre 14 características extraídas de las trayectorias de calidad:

Calidad del clustering

- **Varianza explicada por PCA**: Las dos primeras componentes principales explican el 47.62 % de la varianza total:

Cluster	Cantidad	Porcentaje	Denominación
Cluster 0	∼26,000	26 %	Bajo volumen
Cluster 1	∼25,000	25 %	Super usuarios
Cluster 2	∼24,000	24 %	Usuarios moderados
Cluster 3	∼25,000	25 %	Usuarios irregulares

Cuadro 5.3: Distribución de individuos por cluster (valores aproximados)

- PC1: 27.91 % (mayor variabilidad)
- PC2: 19.71 % (segunda mayor variabilidad)

- **Separación visual**: Los clusters muestran separación clara en el espacio PCA, indicando agrupaciones bien definidas y diferenciadas.

5.3.2 Caracterización detallada de clusters

Cluster / Métrica	Quality Score	Movement Points	Spatial Range	Active Days	Records Co
Cluster 1: Super Usuarios	66.32	35.82	0.56	5.74	486.79
Cluster 2: Usuarios Moderados	49.65	8.69	0.47	5.75	-
Cluster 0: Bajo Volumen	43.51	4.21	0.42	3.36	-
Cluster 3: Usuarios Irregulares	45.82	3.33	0.21	2.94	-

Cuadro 5.4: Caracterización cuantitativa de los clusters identificados

Perfiles comportamentales

1. **Cluster 1: "Super Usuarios"** (25 %):
   - **Perfil**: Usuarios con patrones de movilidad intensos y excelente calidad de datos.
   - **Interpretación**: Representan el segmento ideal para modelado de trayectorias, con suficiente información temporal y espacial para capturar patrones complejos de movilidad urbana.
2. **Cluster 2: Üsuarios Moderados"** (24 %):

- **Característica distintiva**: Precisión GPS promedio de 21.28 metros (la más baja de todos los clusters).
- **Perfil**: Usuarios regulares con actividad sostenida pero menor precisión GPS, sugiriendo uso de A-GPS o triangulación WiFi.
- **Aplicación**: Útiles para análisis de patrones generales pero con menor resolución espacial.

3. **Cluster 0: "Bajo Volumen"** (26 %):

   - **Perfil**: Usuarios con baja actividad y movilidad reducida.
   - **Interpretación**: Probablemente usuarios ocasionales o personas con rutinas muy localizadas.
   - **Aplicación**: Menos útiles para modelado general pero valiosos para estudiar patrones de movilidad local.

4. **Cluster 3: Üsuarios Irregulares"** (25 %):

   - **Características distintivas**: Time Span de solo 2.32 días (período de observación muy corto).
   - **Perfil**: Usuarios con patrones muy irregulares o períodos de monitoreo breves.
   - **Interpretación**: Probablemente representan uso esporádico de la aplicación o instalaciones/desinstalaciones frecuentes.

5.3.3 Análisis de correlaciones entre características

La matriz de correlación reveló relaciones significativas que validan el diseño del algoritmo de puntuación:

Relación	Coeficiente (r)	Interpretación
`quality_score` ↔ `records_count`	0.67	Más datos implica mejor calidad (correlación positiva fuerte)
`quality_score` ↔ `score_volume`	0.75	El volumen impacta directamente la calidad
`movement_points` ↔ `score_mobility`	0.82	Coherencia entre métricas de movilidad
`time_span_days` ↔ `score_duration`	1.00	Relación perfecta por diseño del algoritmo
`score_accuracy` ↔ `avg_accuracy_meters`	-1.00	Relación inversa lógica: menor error = mayor score

Cuadro 5.5: Correlaciones significativas identificadas

Implicación : Las características seleccionadas capturan aspectos complementarios de la calidad de las trayectorias, validando el diseño del algoritmo de puntuación compuesta.

5.3.4 Distribuciones de características clave

1. **Quality Score**:
   - Distribución sesgada hacia valores altos (concentración en 60-80 puntos).
   - Refleja el filtrado previo por calidad mínima ($\geq$35 puntos).
2. **Movement Points**:
   - Gran concentración en valores bajos (<100 puntos).
   - Cola larga hacia valores altos (hasta 500+ puntos).
   - Distribución típica de datos de movilidad con pocos usuarios muy activos.
3. **Spatial Range**:
   - Mayoría con rango espacial pequeño (<2.0 grados, $\sim$220 km).
   - Indica que la mayoría de usuarios se mueve dentro de áreas urbanas locales.
4. **Active Days**:
   - Distribución bimodal con picos en:
     - 3-4 días (usuarios de fin de semana o muy ocasionales).
     - 8-9 días (usuarios con cobertura casi completa del período estudiado).

5.4 Resultados del Análisis de Longitudes de Vuelo

El análisis de distribución de longitudes de trayectoria reveló características extremas que evidencian patrones de movilidad muy heterogéneos.

5.4.1 Estadísticas descriptivas básicas

Métrica	**Valor**	**Interpretación**
Total de individuos	41,882	Muestra analizada
Media	3.48 km	Longitud promedio de trayectoria
Desviación estándar	28.14 km	Variabilidad extremadamente alta
Mínimo	0.0 km	Usuarios sin desplazamiento detectable
Q1 (25 %)	0.0 km	Un cuarto de usuarios no se movió
Mediana (50 %)	0.0 km	La mitad de usuarios tiene longitud cero
Q3 (75 %)	0.63 km	Tercer cuartil apenas 630 metros
Máximo	1,746.08 km	Outlier extremo (Tijuana a CDMX)

Cuadro 5.6: Estadísticas descriptivas de longitudes de vuelo

Métrica	Valor	Interpretación
Asimetría (Skewness)	31.06	Valor extremadamente alto (>3 se considera muy sesgado)
Curtosis (Kurtosis)	1,341.15	Valor excepcional (>3 indica colas pesadas)
Coeficiente de Variación (CV)	8.08	Variabilidad extrema relativa a la media

Cuadro 5.7: Métricas de forma de la distribución de longitudes

5.4.2 Métricas de forma de la distribución

Interpretación de las métricas de forma :

- **Skewness = 31.06**: Indica cola derecha extraordinariamente larga, con la mayoría de datos concentrados en valores bajos con outliers masivos.
- **Kurtosis = 1,341.15**: Sugiere distribución leptocúrtica extrema: pico muy pronunciado en cero y colas muy pesadas.
- **CV = 8.08**: Confirma heterogeneidad extrema en los patrones de movilidad; la desviación estándar es 8 veces mayor que la media.

5.4.3 Distribución inflada en cero

El hallazgo más significativo es la distribución inflada en cero con tres segmentos claramente diferenciados:

5.4.4 Ajuste de distribuciones teóricas

Se intentó ajustar seis distribuciones probabilísticas comunes a los datos:

Conclusiones del ajuste :

- **Ninguna distribución estándar ajusta adecuadamente** los datos debido a:
  - Inflación de ceros (50 % de observaciones en cero).
  - Heterogeneidad extrema entre segmentos.
- **KS Statistics >0.1** indican mal ajuste (valores obtenidos $\gg 0.4$).
- **p-values = 0.0** rechazan todas las hipótesis de ajuste.

Implicación para modelado : Se requiere un **modelo de mezcla (mixture model)** que combine:

- Componente discreta para la masa de probabilidad en cero (50 %).
- Distribución continua (posiblemente lognormal o Weibull truncada) para valores >0.

Segmento	Rango	Cantidad	Interpretación y posibles causas
Usuarios Sedentarios	0.0 km	20,941 (50 %)	▪ Usuarios que permanecieron en una única ubicación ▪ Dispositivos estáticos (olvidados en casa/oficina) ▪ Datos de referencia sin movimiento ▪ Errores en estimación para trayectorias muy cortas
Usuarios Locales	0.01 - 0.63 km	10,470 (25 %)	▪ Desplazamientos dentro del vecindario ▪ Compras en tiendas cercanas ▪ Paseos cortos recreativos ▪ Movilidad intra-edificio
Usuarios Móviles	>0.63 km	10,471 (25 %)	▪ Commuters (casa-trabajo) ▪ Usuarios de transporte público ▪ Deportistas (corredores, ciclistas) ▪ Viajeros ocasionales

Cuadro 5.8: Segmentación de usuarios por patrón de movilidad

5.4.5 Análisis de outliers extremos

Outlier máximo : 1,746.08 km (equivalente a la distancia entre Tijuana y Ciudad de México).

- **Possibles causas**:
  - Errores de GPS (saltos de ubicación por mala señal).
  - Inclusión de segmentos de viaje no peatonal a pesar del filtrado.
  - Múltiples días de actividad sumados incorrectamente.
  - Trayectorias de viajes largos (vacaciones, viajes de negocios).
- **Impacto en el análisis**:
  - Distorsiona la media hacia arriba (3.48 km no es representativa del usuario típico).
  - La mediana (0.0 km) es más robusta pero oculta información de usuarios móviles.
  - Justifica el uso de transformación logarítmica para visualizaciones.

Recomendaciones para modelado futuro :

Distribución	KS Statistic	p-value	AIC	Interpretación
Rayleigh	0.4550	0.0	368,039.06	Ajuste muy pobre
Gamma	0.7378	0.0	-1,978,645.13	Ajuste extremadamente pobre
Lognormal	-	-	-	No convergió
Normal	-	-	-	No aplicable (datos no negativos)
Exponencial	-	-	-	No ajustó
Weibull	-	-	-	No ajustó

Cuadro 5.9: Resultados del ajuste de distribuciones teóricas

1. Limitar longitudes máximas (ej., usando percentil 99).
2. Analizar separadamente los tres segmentos identificados (sedentarios, locales, móviles).
3. Usar medianas en lugar de medias para reportar tendencias centrales.

5.5 Resultados del Análisis de Tiempos de Pausa

El análisis de pausas reveló patrones de comportamiento estacionario que complementan la caracterización de movimiento.

5.5.1 Cobertura y estadísticas generales

Métrica	Valor	Interpretación
Total de pausas detectadas	1,134	Eventos de pausa identificados
Usuarios con pausas detectables	739	Solo 1.8 % del total de usuarios
Sesiones únicas con pausas	867	Períodos de actividad analizados
Ratio pausas/sesión	1.3	Promedio de pausas por período activo

Cuadro 5.10: Cobertura del análisis de tiempos de pausa

Observación crítica : Solo el 1.8 % de usuarios (739 de 41,882) mostraron pausas detectables, sugiriendo:

- Los umbrales de detección (distancia <50m, tiempo >1 segundo) son muy restrictivos.
- La mayoría de trayectorias son demasiado cortas para contener pausas significativas.
- Muchos usuarios tienen solo unos pocos puntos GPS, insuficientes para detectar pausas.

Estadística	Valor	Interpretación
Media	32.0 minutos	*Promedio* de duración de pausas
Mediana	14.0 minutos	*Valor típico* de duración de pausas
Desviación estándar	40.1 minutos	Variabilidad extremadamente alta
Mínimo	0.0167 min (1s)	Pausa más corta detectada
Máximo	472.3 min (7.9h)	Pausa más larga detectada

Cuadro 5.11: Estadísticas de duración de pausas

5.5.2 Distribución de duraciones de pausa

Análisis de la dispersión :

- **Media (32 min) $\gg$ Mediana (14 min)**: Distribución fuertemente sesgada a la derecha.
- **Desviación estándar (40.1 min) > Media (32 min)**: Variabilidad extremadamente alta.
- **Interpretación**: La mayoría de pausas son cortas, pero algunas son muy largas (colas pesadas).

5.5.3 Segmentación por duración

Categoría	Duración	Cantidad	Porcentaje
Pausas muy cortas	<5 min	367	32.4 %
Pausas cortas	5-15 min	215	19.0 %
Pausas medias	15-30 min	145	12.8 %
Pausas largas	>30 min	407	35.9 %

Cuadro 5.12: Distribución de pausas por categoría de duración

Patrón bimodal identificado :

1. **Pico 1**: Pausas muy cortas (32.4 %) - Componente de actividades cotidianas rápidas.
2. **Valle**: Pausas medias (12.8 %) - Zona de transición poco común.
3. **Pico 2**: Pausas largas (35.9 %) - Componente de estancias prolongadas.

Interpretación comportamental :

- **Pausas muy cortas** (<5 min): Semáforos, esperas breves, paradas de tránsito.
- **Pausas cortas** (5-15 min): Compras rápidas, encuentros sociales, café rápido.
- **Pausas medias** (15-30 min): Comidas ligeras, reuniones breves, descansos.
- **Pausas largas** (>30 min): Trabajo, hogar, comidas completas, actividades programadas.

Implicación fundamental : Las personas tienden a hacer pausas muy cortas (pocos minutos) o muy largas (más de media hora), pero raramente pausas de duración intermedia. Esto refleja la naturaleza discreta de las actividades humanas: se está .$^{\text{en}}$ movimiento.$^{\text{o}}$ "detenido por tiempo extendido"para realizar actividades significativas.

Capítulo 6

Conclusiones y Trabajo futuro

6.1 Conclusiones Generales

Este proyecto ha logrado desarrollar una metodología completa para la caracterización de trayectorias individuales a partir de datos GPS masivos, con el objetivo de fundamentar el diseño de modelos de movilidad para la evaluación de protocolos en redes móviles. A través de un proceso sistemático de limpieza, clasificación, filtrado y análisis estadístico, se han obtenido resultados significativos que revelan tanto las posibilidades como las limitaciones inherentes a este tipo de datos.

El procesamiento inicial del dataset, que contenía 69.98 millones de registros distribuidos en 19 columnas con un peso de 22 GB, permitió reducirlo a 51 millones de registros con 7 columnas esenciales (7 GB), representando una optimización del 68 % en tamaño sin pérdida de información relevante. Esta reducción no solo mejoró la eficiencia computacional de los análisis posteriores, sino que también evidenció la importancia de una adecuada selección de características en proyectos de big data.

6.2 Conclusiones por Objetivo

6.2.1 Caracterización de la base de datos

La caracterización exhaustiva del dataset reveló características fundamentales sobre la calidad y naturaleza de los datos de movilidad:

Calidad GPS:

- El 68.73 % de los registros poseen precisión GPS satelital (1-20 metros), lo cual representa una base sólida para análisis de trayectorias peatonales.
- El 31.27 % restante corresponde a tecnologías de geolocalización menos precisas (A-GPS y triangulación WiFi/celular).
- Esta distribución confirma que la mayoría de usuarios tienen dispositivos con capacidades GPS activas y funcionales.

Distribución de identificadores:

- De 6,022,772 identificadores únicos, el 98.17 % aparecen menos de 100 veces en el período de estudio.
- Solo el 1.83 % (110,335 individuos) tienen 100 o más registros.
- La eliminación de duplicados redujo el dataset en 27 % (19 millones de registros) sin afectar el número de individuos únicos, mejorando significativamente la calidad de los datos.

Cobertura temporal:

- Aproximadamente 80 % de los individuos aparecen en un solo día del período estudiado.
- Solo un 5 % de individuos muestran actividad durante 7 o más días.
- Esta distribución indica que la mayoría de usuarios son ocasionales o transitorios, limitando las posibilidades de análisis de rutinas a largo plazo.

Implicación principal: Los datos GPS masivos de aplicaciones móviles contienen una gran cantidad de información, pero solo una pequeña fracción (aproximadamente 2 %) tiene la calidad y persistencia temporal suficiente para análisis detallados de patrones de movilidad.

6.2.2 Algoritmo de clasificación de calidad

El desarrollo del algoritmo de puntuación compuesta basado en seis métricas complementarias (volumen, duración, regularidad, precisión, movilidad y diversidad) demostró ser efectivo para identificar trayectorias útiles:

Resultados del algoritmo:

- Identificó 129,920 individuos (2.15 % del total) con calidad REGULAR o superior.
- La distribución por categorías muestra preponderancia de calidad REGULAR (64.3 %), seguida por BUENA (31.1 %), con solo 0.2 % EXCELENTE.
- El umbral mínimo de 50 registros, 1 día de cobertura y 2 días activos demostró ser apropiado para garantizar información mínima analizable.

Validación del algoritmo: El análisis de clustering posterior validó la efectividad del score de calidad:

- La separación clara en 4 clusters con características diferenciadas confirma que las métricas capturan dimensiones reales de la calidad de datos.
- La correlación fuerte entre `quality_score` y `records_count` ($r = 0{,}67$) y `score_volume` ($r = 0{,}75$) indica que el volumen de datos es el factor más determinante de calidad.

- La varianza explicada del 47.62 % por las dos primeras componentes principales sugiere que las 14 características contienen redundancia moderada pero también información complementaria valiosa.

Fortalezas del algoritmo:

- **Objetivo y reproducible**: Elimina subjetividad en la selección de trayectorias.
- **Multidimensional**: Considera aspectos complementarios (temporal, espacial, precisión).
- **Escalable**: Puede calcularse eficientemente en bases de datos mediante SQL.
- **Interpretable**: Los scores componentes permiten diagnosticar deficiencias específicas.

Limitaciones identificadas:

- **Sensibilidad al volumen**: Los pesos actuales (25 % volumen, 20 % duración) favorecen excesivamente cantidad sobre calidad.
- **Falta de normalización contextual**: No considera que 100 registros en 1 día es diferente que 100 registros en 10 días.
- **Umbral único**: Un umbral de 35 puntos puede ser insuficiente para ciertos tipos de análisis que requieren mayor calidad.

6.2.3 Identificación de trayectorias peatonales

La aplicación del filtro de velocidad basado en parámetros biomecánicos (0.6-2.08 m/s) fue crucial para aislar movimiento peatonal genuino:

Efectividad del filtrado:

- De 129,920 individuos clasificados, solo 41,882 (32.5 %) tienen trayectorias peatonales válidas.
- La reducción del 67.5 % indica que la mayoría de usuarios se desplazan predominantemente en vehículos motorizados.
- La velocidad promedio resultante (1.17 m/s $\approx$ 4.21 km/h) es consistente con estudios de marcha humana normal.
- El límite superior alcanzado exactamente (2.08 m/s) confirma que el filtro opera correctamente.

Segmentación de trayectorias:

- La división de trayectorias en segmentos continuos permitió separar períodos de caminata de períodos en vehículo.
- Se generaron 171,461 puntos de recorrido peatonal validados, organizados en múltiples segmentos por usuario.

- Esta segmentación preserva la integridad temporal de cada modo de transporte.

Cobertura geográfica:

- Las trayectorias abarcan desde el sur de México (Chiapas, lat. 14.7°) hasta el norte (Baja California, lat. 32.7°).
- Rango longitudinal desde la costa del Pacífico (-117.1°) hasta la península de Yucatán (-86.7°).
- Esta amplia cobertura permite generalización de resultados a diferentes contextos urbanos mexicanos.

Desafío principal: La alta proporción de usuarios descartados (67.5 %) plantea la pregunta de si los umbrales son demasiado restrictivos o si genuinamente la movilidad peatonal pura es rara en contextos urbanos contemporáneos donde el transporte motorizado domina.

6.2.4 Aplicación de inteligencia artificial: Clustering K-Means

El clustering no supervisado reveló cuatro perfiles de usuario claramente diferenciados, validando la hipótesis de que existen subgrupos naturales en los datos:

Hallazgos principales:

1. **Super Usuarios (Cluster 1, 25 %):**
    - Representan el segmento ideal para modelado: alta calidad (66.32 puntos), movilidad significativa (0.56 spatial range), consistencia temporal (5.74 días activos).
    - Son el grupo objetivo prioritario para extracción de patrones de movilidad representativos.
    - Probablemente corresponden a usuarios que usan la aplicación como herramienta principal de navegación o tracking.
2. **Usuarios Moderados (Cluster 2, 24 %):**
    - Presentan un desafío específico: actividad regular pero menor precisión GPS (21.28m promedio).
    - La menor precisión sugiere uso de A-GPS o triangulación en lugar de GPS puro.
    - Útiles para análisis de patrones generales pero no para modelado de alta resolución.
3. **Bajo Volumen (Cluster 0, 26 %):**
    - Movilidad muy localizada (0.42 spatial range) sugiere usuarios que no salen frecuentemente de su vecindario.

- Valiosos para estudiar movilidad intra-barrial o comportamiento de poblaciones con limitaciones de movilidad.
- Menor utilidad para modelar desplazamientos urbanos de largo alcance.

4. **Usuarios Irregulares (Cluster 3, 25 %):**
   - Períodos de observación muy cortos (2.32 días time_span) limitan severamente su utilidad.
   - Probablemente representan instalaciones temporales de la app o uso turístico.
   - Menos relevantes para modelado de rutinas pero potencialmente útiles para estudiar movilidad de visitantes.

Validación cruzada:

- La distribución relativamente uniforme (24-26 % en cada cluster) indica que el algoritmo no está sesgado hacia crear clusters desbalanceados.
- La correlación entre características dentro de cada cluster confirma consistencia interna.
- La separación clara en el espacio PCA (47.62 % varianza explicada) demuestra que los clusters capturan diferencias reales, no artificiales.

Implicación para el modelado: Los modelos de movilidad deberían considerar estos perfiles diferenciados, potencialmente creando modelos especializados para cada cluster en lugar de un modelo único que intente capturar toda la heterogeneidad.

6.2.5 Caracterización de elementos del modelo de movilidad

Puntos de recorrido: Los 171,461 puntos de recorrido peatonal identificados representan ubicaciones validadas donde los individuos se encuentran durante su movilidad a pie. La distribución geográfica amplia (latitud 14.7°-32.7°, longitud -117.1° a -86.7°) sugiere que los patrones encontrados pueden generalizarse a diferentes contextos urbanos mexicanos.

Características destacadas:

- Los puntos están concentrados en áreas urbanas, como se esperaba.
- La precisión promedio de estos puntos está dentro del rango de GPS satelital.
- La segmentación temporal (por día) permite rastrear evolución de patrones.

Limitación: No se identificaron explícitamente "puntos de interés"(POIs) recurrentes como hogar, trabajo, etc. Este análisis requeriría clustering espacial adicional que no se implementó en este proyecto.

Tiempos de pausa: El análisis de pausas reveló un patrón bimodal significativo:

Hallazgos principales:

- **Distribución bimodal clara:**

  - 32.4 % de pausas son muy cortas (¡5 minutos): semáforos, esperas breves.
  - 35.9 % de pausas son largas (¿30 minutos): trabajo, hogar, comidas.
  - Solo 12.8 % en el rango intermedio (15-30 minutos).
- **Estadísticas centrales:**
  - Media: 32 minutos (influenciada por colas largas).
  - Mediana: 14 minutos (más representativa del comportamiento típico).
  - Máximo: 7.9 horas (períodos de sueño o estancias muy prolongadas).
- **Alta variabilidad:**
  - Desviación estándar (40.1 min) ¿Media (32 min).
  - Coeficiente de variación alto indica heterogeneidad extrema.
  - No es posible un modelo de pausa único; se requiere modelo de mezcla.

Limitación crítica: Solo el 1.8 % de usuarios (739 de 41,882) mostraron pausas detectables. Esto sugiere que:

- Los umbrales de detección (distancia ¡50m, tiempo ¿1 segundo) son muy restrictivos.
- La mayoría de trayectorias son demasiado cortas para contener pausas.
- Muchos usuarios tienen datos GPS dispersos temporalmente sin suficiente resolución.

Para un modelo de movilidad efectivo, se recomienda:

- Distribución de mezcla con dos componentes: pausas cortas (exponencial, $\lambda \approx 0{,}2$) y pausas largas (lognormal, $\mu \approx 3{,}5$, $\sigma \approx 1{,}2$).
- Probabilidad de pausa corta vs. larga: 55 % vs. 45 %.
- Filtrado de pausas ¡30 segundos como artefactos.

Longitudes de vuelo: Este análisis reveló los resultados más complejos y problemáticos del proyecto:

Hallazgo fundamental: Distribución inflada en cero:

- 50 % de usuarios: longitud = 0 km (sin desplazamiento detectable).
- 25 % de usuarios: longitud 0.01-0.63 km (movilidad muy local).
- 25 % de usuarios: longitud ¿0.63 km (movilidad significativa).

Métricas extremas:

- Skewness: 31.06 (asimetría extrema).
- Kurtosis: 1,341.15 (colas extraordinariamente pesadas).
- CV: 8.08 (variabilidad 8 veces la media).
- Máximo: 1,746 km (outlier claro).

Fracaso de modelos paramétricos:

- Ninguna distribución estándar (normal, lognormal, exponencial, gamma, Weibull, Rayleigh) ajustó adecuadamente.
- KS statistics ¿0.45 (valores ¿0.1 ya indican mal ajuste).
- p-values = 0.0 (rechazo completo de todas las hipótesis).

Diagnóstico del problema:

- **Método de cálculo simplificado**: Se usó $(\sigma_{lat} + \sigma_{lon}) \times 111$ como proxy de distancia total, no la suma de distancias reales entre puntos consecutivos.
- **Definición ambigua de "longitud de trayectoria"**: ¿Es la distancia total recorrida? ¿Es el rango espacial (máx-mín)? ¿Es la distancia entre primer y último punto?
- **Heterogeneidad extrema no modelable**: La mezcla de usuarios sedentarios (50 %), locales (25 %) y móviles (25 %) no puede capturarse con una sola distribución.

Recomendación crítica para trabajo futuro: El concepto de "longitud de vuelo"debe redefinirse como la distancia entre puntos de recorrido consecutivos (no la longitud total de trayectoria). Esto requiere:

- Calcular distancia haversine entre cada par de puntos sucesivos.
- Analizar la distribución de estas distancias individuales.
- Esto produciría una distribución más homogénea y modelable.

6.3 Limitaciones del Estudio

6.3.1 Limitaciones de los datos

1. **Período temporal limitado:**
   - Solo 10 días de datos (6-15 noviembre 2022).
   - Imposible capturar patrones de largo plazo, estacionalidad, o variación mensual/anual.
   - Sesgos potenciales por eventos específicos de esas fechas.
2. **Cobertura geográfica sesgada:**
   - Datos concentrados en áreas urbanas.
   - Posible subrepresentación de zonas rurales, áreas de bajos ingresos, o grupos demográficos específicos.
   - No se tiene información demográfica de los usuarios para validar representatividad.
3. **Usuarios mayoritariamente ocasionales:**

- 80 % de usuarios aparecen solo 1 día.
- Dificulta análisis de rutinas y patrones recurrentes.
- Sesgo hacia movilidad no-rutinaria (turismo, eventos especiales).

4. **Baja tasa de detección de pausas:**

   - Solo 1.8 % de usuarios con pausas detectables.
   - Umbrales de detección quizás demasiado restrictivos.
   - Resolución temporal del GPS (muestreo) insuficiente para capturar todas las pausas.

6.3.2 Limitaciones metodológicas

1. **Cálculo simplificado de longitudes de trayectoria:**

   - Uso de $(\sigma_{lat} + \sigma_{lon}) \times 111$ en lugar de suma de distancias reales.
   - Produce estimaciones imprecisas especialmente para trayectorias con retornos al origen.
   - Imposibilita ajuste de distribuciones paramétricas estándar.

2. **Falta de validación externa:**

   - No se contrastaron resultados con datos de referencia (surveys, GPS de alta precisión).
   - No se validó que los "Super usuarios"genuinamente tengan patrones de movilidad superiores.
   - Asunciones no verificadas (ej. velocidad 0.6-2.08 m/s cubre toda movilidad peatonal real).

3. **Clustering sin validación semántica:**

   - Los 4 clusters identificados no se validaron contra información real de usuarios.
   - No se sabe si los "Super usuarios"son realmente personas más móviles o simplemente usuarios que dejaron la app activa más tiempo.
   - Falta análisis de estabilidad (¿los mismos usuarios permanecen en el mismo cluster en diferentes períodos?).

4. **Ausencia de análisis de POIs:**

   - No se identificaron lugares recurrentes (hogar, trabajo, lugares frecuentes).
   - Imposible distinguir entre movimiento exploratorio vs. rutinario.
   - Limita aplicabilidad para modelado de rutinas.

6.3.3 Limitaciones computacionales

1. **Restricciones de memoria:**
   - Procesamiento por chunks limita algunos análisis (ej. clustering global de todos los usuarios).
   - No se pudieron aplicar técnicas más sofisticadas que requieren carga completa en RAM.
   - Algunos cálculos (ej. detección de pausas) son computacionalmente costosos y se limitaron a subconjuntos.
2. **Tiempo de procesamiento:**
   - Varios scripts toman horas en ejecutarse completamente.
   - Limita iteración y refinamiento de parámetros.
   - Imposibilita análisis interactivo o exploratorio ágil.

6.4 Trabajo Futuro

6.4.1 Mejoras inmediatas al pipeline actual

1. **Recalcular longitudes de vuelo correctamente:**
   - Implementar cálculo de suma de distancias haversine entre puntos consecutivos.
   - Analizar distribución de distancias individuales (step lengths) en lugar de longitud total de trayectoria.
   - Ajustar distribuciones estándar (lognormal, Weibull, gamma) a las distancias individuales.
2. **Refinar algoritmos de detección de pausas:**
   - Ajustar umbrales de distancia (¿20m en lugar de 50m?) y tiempo mínimo.
   - Implementar detección multi-escala para capturar diferentes tipos de pausas.
   - Validar resultados contra datasets etiquetados o simulados.
3. **Mejorar el algoritmo de clasificación de calidad:**
   - Revisar pesos de las métricas (disminuir peso de volumen, aumentar peso de regularidad).
   - Introducir normalización contextual (registros por día en lugar de total absoluto).
   - Implementar umbrales adaptativos por contexto geográfico/temporal.
4. **Optimizaciones computacionales:**

- Vectorizar cálculos de distancias haversine.
- Implementar procesamiento paralelo para scripts más costosos.
- Migrar algunos cálculos a bases de datos (PostGIS) para mayor eficiencia.

6.4.2 Extensiones metodológicas

1. **Identificación de puntos de interés (POIs):**
   - Aplicar clustering espacial (DBSCAN, OPTICS) para identificar lugares recurrentes.
   - Inferir funciones de POIs (hogar, trabajo, ocio) basado en patrones temporales.
   - Integrar datos externos (OpenStreetMap, Google Places) para enriquecer contexto.
2. **Análisis de rutinas temporales:**
   - Identificar patrones diarios/semanales de movilidad.
   - Detectar rutinas (home-work-home) y variaciones.
   - Analizar regularidad vs. aleatoriedad en patrones de desplazamiento.
3. **Análisis de redes de movilidad:**
   - Construir grafos de movilidad (nodos = ubicaciones, aristas = transiciones).
   - Calcular métricas de red (centralidad, modularidad, clusters).
   - Identificar hubs de actividad y rutas principales.
4. **Validación cruzada con datos externos:**
   - Comparar resultados con surveys de movilidad (ENMODO, INEGI).
   - Validar perfiles de usuario con datos demográficos (si disponibles).
   - Contrastar con datos GPS de alta precisión (estudios controlados).

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:

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

```
45          - "5432:5432"
46          volumes:
47          - postgres_data:/var/lib/postgresql/data
48          networks:
49          - data-network
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          ports:
62          - "8080:8080"
63          depends_on:
64          - postgres
65          networks:
66          - data-network
67
68          volumes:
69          postgres_data:
70
71          networks:
72          data-network:
73          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 "
       status=running" | grep -q data-analysis; then
4  echo "Contenedor␣no␣esta␣corriendo.␣Levantando␣con␣docker-
       compose..."
5  docker-compose up -d
6  echo "Esperando␣que␣se␣instalen␣las␣dependencias..."
7  while ! docker exec data-analysis pip show colorama &> /dev/
       null; do
8  sleep 2
9  done
10 echo "Dependencias␣instaladas␣correctamente."
11 else
12 echo "Contenedor␣ya␣esta␣corriendo.␣Usando␣instancia␣existente.
       "
13 fi
14
15 echo "Ejecutando␣script..."
16 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.

Para poder ejecutar un script dentro del contenedor, y dejarlo corriendo en segundo plano sin necesidad de una conexión SSH activa. Hay que instalar tmux dentro del contenedor, como se muestra a continuación:

```
sudo docker exec -u root -it data-analysis bash
apt update && apt install tmux -y
```

Una vez instalado tmux dentro del contenedor se pueden usar los siguiente comandos:

1. **Crear una sesión tmux**

```
sudo docker exec -it data-analysis tmux new-session -d -s
   {SessionName} 'cd /app && python3 {PythonScript}.py'
```

2. **Listas las sesiones tmux activas**

```
sudo docker exec -it data-analysis tmux list-sessions
```

3. **Conectarse a la sesión tmux creada** Para poder salir de la sesión sin detener el proceso, presione `Ctrl + b` y luego `d` (de detach).

```
sudo docker exec -it data-analysis tmux attach -t {
   SessionName}
```

Apéndice B

Scripts

```
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.

```
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␣(paciencia␣para␣archivos␣
        grandes)...")
19  total_registros = ddf.shape[0].compute()
20
21  print(f"\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=
              columnas_deseadas)
14
15        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         import sys
6         from tqdm import tqdm
7
8         def classify_tech(valor):
9         if 1 <= valor <= 20:
10        return 'GPS Satelital'
11        elif 5 <= valor <= 50:
12        return 'A-GPS (Asistido por red)'
13        elif 20 <= valor <= 500:
14        return 'Triangulacion WiFi/Redes Moviles'
15        else:
16        return 'Fuera de rango'
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: Debe especificar un archivo CSV como
              argumento")
28        sys.exit(1)
29
30        csv_file = sys.argv[1]
31        filename = os.path.splitext(os.path.basename(csv_file))
              [0]
32        column = "device_horizontal_accuracy"
33        bins = 100
34
35        print(f"\nIniciando procesamiento del archivo: {
              csv_file}")
36        print(f"Columna analizada: {column}")
37
38        os.makedirs("img", exist_ok=True)
39        print("Directorio 'img' verificado/creado")
40
41        print("\nProcesando datos y clasificando tecnologias...
              ")
42        frequency = pd.Series(dtype=float)
43        tech_counts = {
44                'GPS Satelital': 0,
45                'A-GPS (Asistido por red)': 0,
46                'Triangulacion WiFi/Redes Moviles': 0,
```

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

```
83          bbox=dict(facecolor='white', alpha=0.8, edgecolor='r'))
84          plt.text(40, max(counts)*0.8, agps_str, ha='center',
                color='g', fontsize=10,
85          bbox=dict(facecolor='white', alpha=0.8, edgecolor='g'))
86          plt.text(190, max(counts)*0.7, wifi_str, ha='center',
                color='b', fontsize=10,
87          bbox=dict(facecolor='white', alpha=0.8, edgecolor='b'))
88
89
90          plt.title(f"DistribuciOn␣de␣precisiones␣de␣{column}\
                nArchivo:␣{filename}", fontsize=14)
91          plt.xlabel(f"Valores␣de␣{column}␣(metros)", fontsize
                =12)
92          plt.ylabel("Frecuencia␣(Millones)", fontsize=12)
93          plt.xticks(edges[::5], rotation=45)
94          plt.grid(axis='y', linestyle='--')
95          plt.tight_layout()
96
97          output_path = os.path.join("img", f"histograma_{column}
                _{filename}.png")
98          plt.savefig(output_path, dpi=300, bbox_inches='tight')
99          plt.close()
100
101         print("\n===␣DISTRIBUCION␣DE␣TECNOLOGIAS␣DE␣
                GEOLOCALIZACION␣===")
102         for tech, count in tech_counts.items():
103         print(f"{tech}:␣{count:,}␣registros␣({percentage[tech
                ]:.2f}%)")
104
105         print(f"\nHistograma␣generado␣exitosamente")
106         print(f"Archivo␣guardado␣en:␣{output_path}")
107         print(f"Total␣registros␣analizados:␣{total:,}\n")
108
109         if __name__ == "__main__":
110         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,
              usecols=[column], chunksize=chunksize))
36
37        with tqdm(total=total_chunks, unit=' chunk') as pbar:
38        for chunk in pd.read_csv(csv_file, usecols=[column],
              chunksize=chunksize):
39        counter.update(chunk[column].dropna().astype(str))
40        pbar.update(1)
41
42        frecuency = pd.Series(counter)
43        total_unique_values = len(frecuency)
44        max_freq = frecuency.max()
45
```

```
46          print(f"Datos␣procesados␣correctamente")
47          print(f"Total␣de␣valores␣Unicos:␣{total_unique_values
               :,}")
48          print(f"Frecuencia␣maxima:␣{max_freq:,}")
49
50          bins = [0] + [10**i for i in range(0, int(np.log10(
               max_freq)) + 2)]
51          group_freq = pd.cut(frecuency, bins=bins, right=False).
               value_counts().sort_index()
52
53          total_ocurrence = frecuency.sum()
54          percentage_per_range = (group_freq /
               total_unique_values * 100).round(2)
55
56          print("\nGenerando␣histograma␣con␣estadisticas...")
57          plt.figure(figsize=(16, 9))
58          ax = group_freq.plot(kind='bar', logy=True, alpha=0.7,
               edgecolor='black')
59
60          formatted_labels = []
61          for interval in group_freq.index.categories:
62          left = int(interval.left)
63          right = int(interval.right - 1)
64          formatted_labels.append(f"{left}-{right}" if left !=
               right else f"{left}")
65
66          plt.xticks(range(len(formatted_labels)),
               formatted_labels, rotation=45, ha='right')
67
68          plt.title(f"Histograma␣de␣Frecuencias␣de␣
               Identificadores\nArchivo:␣{filename}", fontsize=16,
               pad=20)
69          plt.xlabel("Rango␣de␣Frecuencia", fontsize=14)
70          plt.ylabel("Cantidad␣de␣Valores␣Unicos␣(log)", fontsize
               =14)
71          plt.grid(True, which="both", ls="--", axis='y')
72
73          stats_text = (
74          f"Total␣valores␣unicos:␣{format_count(
               total_unique_values)}\n"
75          f"Total␣ocurrencias:␣{format_count(total_ocurrence)}\n"
76          f"Frecuencia␣maxima:␣{format_count(max_freq)}"
77          )
78          plt.annotate(stats_text,
79          xy=(0.95, 0.95),
80          xycoords='axes␣fraction',
81          fontsize=15,
82          ha='right',
83          va='top',
84          bbox=dict(boxstyle='round', facecolor='white', alpha
               =0.9))
```

```
85
86          max_val = group_freq.max()
87          min_y = 0.9
88
89          for i, (count, porcent) in enumerate(zip(group_freq.
               values, percentage_per_range.values)):
90          if count > 0:
91          y_pos = count * 1.1 if count * 1.1 > min_y else min_y *
                1.2
92
93          text = f"{porcent}%\n({format_count(count)})"
94
95          ax.text(
96          i, y_pos, text,
97          ha='center', va='bottom',
98          fontsize=15,
99          fontweight='bold',
100         bbox=dict(
101         facecolor='white',
102         alpha=0.85,
103         edgecolor='lightgray',
104         boxstyle='round,pad=0.3'
105         )
106         )
107
108         output_path = os.path.join("img", f"histograma_{column}
               _{filename}.png")
109         plt.tight_layout()
110         plt.savefig(output_path, dpi=300, bbox_inches='tight')
111         plt.close()
112
113         print("\n=== DISTRIBUCION DE FRECUENCIAS ===")
114         for i, (intervalo, count) in enumerate(group_freq.items
               ()):
115         print(f"Rango {formatted_labels[i]}: {count:,}")
116
117         print(f"\nHistograma generado exitosamente")
118         print(f"Archivo guardado en: {output_path}")
119         print(f"Total ocurrencias analizadas: {total_ocurrence
               :,}\n")
120
121         if __name__ == "__main__":
122         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).
              value_counts().sort_index()
18        total_values = len(data)
19        max_count = grouped.max()
20
21        plt.figure(figsize=(14, 8))
22        ax = grouped.plot(kind='bar', color=color, edgecolor='
              black', alpha=0.7, logy=log_scale)
23
24        bin_labels = []
25        for interval in grouped.index.categories:
26        left = int(interval.left)
27        right = int(interval.right)
28        bin_labels.append(f"{left}-{right-1}" if right-left > 1
               else str(left))
29
30        plt.xticks(range(len(bin_labels)), bin_labels, rotation
              =45, ha='right')
31        plt.title(f"{title}\nTotal␣valores␣unicos:␣{
              format_count(total_values)}", fontsize=14, pad=20)
32        plt.xlabel("Rango␣de␣repeticiones", fontsize=12)
33        plt.ylabel("Cantidad␣de␣valores␣unicos" + ("␣(log)" if
              log_scale else ""), fontsize=12)
34        plt.grid(True, which="both", ls="--", axis='y')
35
36        min_y = 0.9
37        for i, (count, interval) in enumerate(zip(grouped.
              values, grouped.index)):
38        if count > 0:
39        percentage = (count / total_values) * 100
40        y_pos = count * 1.1 if count * 1.1 > min_y else min_y *
               1.2
41        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, edgecolor='
                lightgray', boxstyle='round,pad=0.2'))
47
48          output_path = os.path.join("img", filename)
49          plt.tight_layout()
50          plt.savefig(output_path, dpi=300, bbox_inches='tight')
51          plt.close()
52          return output_path
53
54          def main():
55          if len(sys.argv) < 2:
56          print("Error: Debe especificar un archivo CSV")
57          sys.exit(1)
58
59          csv_file = sys.argv[1]
60          filename_base = os.path.splitext(os.path.basename(
                csv_file))[0]
61          column = "identifier"
62          chunksize = 1_000_000
63          os.makedirs("img", exist_ok=True)
64
65          print(f"\nIniciando analisis de: {csv_file}")
66          print(f"Columna analizada: {column}")
67
68          print("\nContando frecuencias...")
69          counter = Counter()
70
71          with tqdm(desc="  Contando filas totales", unit=' filas
                ') as pbar:
72          total_rows = 0
73          for chunk in pd.read_csv(csv_file, usecols=[column],
                chunksize=chunksize):
74          total_rows += len(chunk)
75          pbar.update(len(chunk))
76
77          with tqdm(total=total_rows, desc="  Procesando datos",
                unit=' filas') as pbar:
78          for chunk in pd.read_csv(csv_file, usecols=[column],
                chunksize=chunksize):
79          counter.update(chunk[column].dropna().astype(str))
80          pbar.update(len(chunk))
81
82          frequencies = pd.Series(counter)
83          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}_{filename_base}.
               png",
126         color='#55A868',
```

```
127         log_scale=True
128         )
129         pbar.update(1)
130 
131         high_path = create_histogram(
132         high_freq,
133         bins=high_bin,
134         title="Distribucion␣de␣Frecuencias␣(1001-10,000␣
                repeticiones)",
135         filename=f"histograma_1k-10k_{column}_{filename_base}.
                png",
136         color='#C44E52',
137         log_scale=True
138         )
139         pbar.update(1)
140 
141         print("\nGraficos␣generados␣exitosamente:")
142         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', '
              device_lat'],
14        keep='first'
15        )
16
17        print(f"Numero␣de␣registros␣despues␣de␣eliminar␣
              duplicados:␣{len(ddf_deduplicate):,}")
18
19        ddf_deduplicate.to_csv(
20        output_file,
21        index=False,
22        single_file=True
23        )
24
25        print(f"\nArchivo␣sin␣duplicados␣guardado␣en:␣{
              output_file}")
26
27        if __name__ == "__main__":
28        if len(sys.argv) < 2:
29        print("Error:␣Debe␣especificar␣un␣archivo␣CSV␣como␣
              argumento")
30        sys.exit(1)
31
32        input_csv = sys.argv[1]
33        base_name = os.path.splitext(input_csv)[0]
34        output_csv = f"{base_name}_DeDuplicate.csv"
35
36        delete_duplicates(input_csv, output_csv)
```

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

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

```
44          for chunk in pd.read_csv(csv_file, usecols=[
               timestamp_col, identifier_col], chunksize=chunksize)
               :
45          try:
46          chunk['date'] = pd.to_datetime(
47          chunk[timestamp_col],
48          format='mixed',
49          errors='coerce'
50          ).dt.date
51
52          chunk = chunk.dropna(subset=['date'])
53
54          for date, group in chunk.groupby('date'):
55          if date not in date_freqs:
56          date_freqs[date] = Counter()
57
58          date_freqs[date].update(group[identifier_col].dropna().
               astype(str))
59
60          current_max = date_freqs[date].most_common(1)[0][1] if
               date_freqs[date] else 0
61          if current_max > max_freq:
62          max_freq = current_max
63          except Exception as e:
64          print(f"\nError procesando chunk: {str(e)}")
65          continue
66          finally:
67          pbar.update(1)
68
69          if not date_freqs:
70          print("\nError: No se encontraron datos validos para
               procesar")
71          sys.exit(1)
72
73          bins = [0] + [10**i for i in range(0, int(np.log10(
               max_freq)) + 2)] if max_freq > 0 else [0, 1]
74
75          print("\nGenerando histogramas por dia...")
76
77          for date, counter in tqdm(date_freqs.items(), total=len
               (date_freqs), unit=' dia'):
78          frecuency = pd.Series(counter)
79          total_unique_values = len(frecuency)
80          total_ocurrence = frecuency.sum()
81
82          group_freq = pd.cut(frecuency, bins=bins, right=False).
               value_counts().sort_index()
83          percentage_per_range = (group_freq /
               total_unique_values * 100).round(2)
84
85          plt.figure(figsize=(16, 9))
```

```
86          ax = group_freq.plot(kind='bar', logy=True, alpha=0.7,
                edgecolor='black')
87
88          formatted_labels = []
89          for interval in group_freq.index.categories:
90          left = int(interval.left)
91          right = int(interval.right - 1)
92          formatted_labels.append(f"{left}-{right}" if left !=
                right else f"{left}")
93
94          plt.xticks(range(len(formatted_labels)),
                formatted_labels, rotation=45, ha='right')
95
96          date_str = date.strftime('%Y-%m-%d')
97          plt.title(f"Histograma de Frecuencias de
                Identificadores\nArchivo: {filename} - Fecha: {
                date_str}", fontsize=16, pad=20)
98          plt.xlabel("Rango de Frecuencia", fontsize=14)
99          plt.ylabel("Cantidad de Valores Unicos (log)", fontsize
                =14)
100         plt.grid(True, which="both", ls="--", axis='y')
101
102         stats_text = (
103         f"Total valores unicos: {(total_unique_values):,}\n"
104         f"Total ocurrencias: {(total_ocurrence):,}\n"
105         f"Frecuencia maxima: {(frecuency.max()):,}"
106         )
107         plt.annotate(stats_text,
108         xy=(0.95, 0.95),
109         xycoords='axes fraction',
110         fontsize=15,
111         ha='right',
112         va='top',
113         bbox=dict(boxstyle='round', facecolor='white', alpha
                =0.9))
114
115         max_val = group_freq.max()
116         min_y = 0.9
117
118         for i, (count, porcent) in enumerate(zip(group_freq.
                values, percentage_per_range.values)):
119         if count > 0:
120         y_pos = count * 1.1 if count * 1.1 > min_y else min_y *
                 1.2
121         text = f"{porcent}%\n({format_count(count)})"
122         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", f
                "histograma_{identifier_col}_{filename}_{date_str}.
                png")
136         plt.tight_layout()
137         plt.savefig(output_path, dpi=300, bbox_inches='tight')
138         plt.close()
139 
140         print("\nHistogramas generados exitosamente")
141         print(f"Archivos guardados en: img/daily_histograms/")
142         print(f"Total dias procesados: {len(date_freqs)}")
143         print(f"Frecuencia maxima global encontrada: {
                format_count(max_freq)}\n")
144 
145         if __name__ == "__main__":
146         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 - %(levelname)s - %(message)s')
9         logger = logging.getLogger(__name__)
10
11        class MobilityDataLoader:
12        def __init__(self):
13        self.db_config = {
14                'host': os.getenv('DB_HOST', 'localhost'),
15                'port': os.getenv('DB_PORT', '5432'),
16                'user': os.getenv('DB_USER', 'postgres'),
17                'password': os.getenv('DB_PASSWORD', '
                     postgres123'),
18                'default_db': os.getenv('DB_NAME', 'postgres')
                      # DB por defecto para crear la nueva
19        }
20        self.target_db = 'trajectories'
21        self.csv_file = 'Mobility_Data_Slim_DeDuplicate.csv'
22
23        def create_database(self):
24        try:
25        conn = psycopg2.connect(
26        host=self.db_config['host'],
27        port=self.db_config['port'],
28        user=self.db_config['user'],
29        password=self.db_config['password'],
30        database=self.db_config['default_db']
31        )
32        conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
33        cur = conn.cursor()
34        cur.execute("SELECT 1 FROM pg_catalog.pg_database WHERE
             datname = %s", (self.target_db,))
35        exists = cur.fetchone()
36
37        if not exists:
38        cur.execute(f'CREATE DATABASE {self.target_db}')
39        logger.info(f"Base de datos '{self.target_db}' creada
             exitosamente")
40        else:
41        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:␣{e}")
48          raise
49
50          def analyze_csv_structure(self):
51          try:
52          if not os.path.exists(self.csv_file):
53          raise FileNotFoundError(f"Archivo␣{self.csv_file}␣no␣
                encontrado")
54
55          df_sample = pd.read_csv(self.csv_file, nrows=5)
56          logger.info(f"Estructura␣del␣CSV:")
57          logger.info(f"Columnas:␣{list(df_sample.columns)}")
58          logger.info(f"Tipos␣de␣datos:")
59          for col, dtype in df_sample.dtypes.items():
60          logger.info(f"␣␣{col}:␣{dtype}")
61
62          return df_sample
63
64          except Exception as e:
65          logger.error(f"Error␣al␣analizar␣CSV:␣{e}")
66          raise
67
68          def create_table_from_csv(self, df_sample):
69          try:
70          engine = create_engine(
71          f"postgresql://{self.db_config['user']}:{self.db_config
                ['password']}@"
72          f"{self.db_config['host']}:{self.db_config['port']}/{
                self.target_db}"
73          )
74          type_mapping = {
75                  'object': 'TEXT',
76                  'int64': 'BIGINT',
77                  'int32': 'INTEGER',
78                  'float64': 'DOUBLE␣PRECISION',
79                  'float32': 'REAL',
80                  'bool': 'BOOLEAN',
81                  'datetime64[ns]': 'TIMESTAMP'
82          }
83          columns_ddl = []
84          for col, dtype in df_sample.dtypes.items():
85          pg_type = type_mapping.get(str(dtype), 'TEXT')
86          clean_col = col.lower().replace('␣', '_').replace('-',
                '_').replace('.', '_')
87          columns_ddl.append(f"{clean_col}␣{pg_type}")
88
89          create_table_sql = f"""
90  ␣␣␣␣␣␣␣␣CREATE␣TABLE␣IF␣NOT␣EXISTS␣mobility_data␣(
```

```
91  ␣␣␣␣␣␣␣␣id␣SERIAL␣PRIMARY␣KEY,
92  ␣␣␣␣␣␣␣␣{',␣'.join(columns_ddl)},
93  ␣␣␣␣␣␣␣␣created_at␣TIMESTAMP␣DEFAULT␣CURRENT_TIMESTAMP
94  ␣␣␣␣␣␣␣␣)
95  ␣␣␣␣␣␣␣␣"""
96
97          with engine.connect() as conn:
98          conn.execute(text("DROP␣TABLE␣IF␣EXISTS␣mobility_data")
                )
99          conn.execute(text(create_table_sql))
100         conn.commit()
101
102         logger.info("Tabla␣'mobility_data'␣creada␣exitosamente"
                )
103
104         return engine
105
106         except Exception as e:
107         logger.error(f"Error␣al␣crear␣la␣tabla:␣{e}")
108         raise
109
110         def load_csv_to_table(self, engine):
111         try:
112         logger.info(f"Leyendo␣archivo␣CSV:␣{self.csv_file}")
113         df = pd.read_csv(self.csv_file)
114
115         # Limpiar nombres de columnas
116         df.columns = [col.lower().replace('␣', '_').replace('-'
                , '_').replace('.', '_')
117         for col in df.columns]
118
119         logger.info(f"Cargando␣{len(df)}␣registros␣a␣la␣tabla
                ...")
120
121         chunk_size = 1000
122         total_chunks = len(df) // chunk_size + (1 if len(df) %
                chunk_size else 0)
123
124         for i, chunk in enumerate(pd.read_csv(self.csv_file,
                chunksize=chunk_size)):
125         # Limpiar nombres de columnas del chunk
126         chunk.columns = [col.lower().replace('␣', '_').replace(
                '-', '_').replace('.', '_')
127         for col in chunk.columns]
128
129         chunk.to_sql('mobility_data', engine, if_exists='append
                ', index=False, method='multi')
130         logger.info(f"Procesado␣chunk␣{i+1}/{total_chunks}")
131
132         logger.info("Datos␣cargados␣exitosamente")
133
```

```
134         with engine.connect() as conn:
135         result = conn.execute(text("SELECT␣COUNT(*)␣FROM␣
                mobility_data"))
136         count = result.fetchone()[0]
137         logger.info(f"Total␣de␣registros␣en␣la␣tabla:␣{count}")
138
139         except Exception as e:
140         logger.error(f"Error␣al␣cargar␣datos:␣{e}")
141         raise
142
143         def run(self):
144         try:
145         logger.info("===␣Iniciando␣proceso␣de␣carga␣de␣datos␣de
                ␣movilidad␣===")
146
147         self.create_database()
148         df_sample = self.analyze_csv_structure()
149         engine = self.create_table_from_csv(df_sample)
150         self.load_csv_to_table(engine)
151
152         logger.info("===␣Proceso␣completado␣exitosamente␣===")
153
154         except Exception as e:
155         logger.error(f"Error␣en␣el␣proceso:␣{e}")
156         raise
157
158         if __name__ == "__main__":
159         loader = MobilityDataLoader()
160         loader.run()
```

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

```
1         WITH estadisticas AS (
2         SELECT
3         COUNT(id) AS total_individuos,
4         SUM(CASE WHEN device_horizontal_accuracy < 20 THEN 1
              ELSE 0 END) AS total_precision_gps
5         FROM mobility_data
6         )
7         SELECT
8         total_individuos,
9         total_precision_gps,
10        ROUND((total_precision_gps * 100.0 / total_individuos),
              2) AS porcentaje_precision_gps
11        FROM estadisticas;
```

Código B.11: Porcentaje de individuos con precisión de GPS mejor a 20 metros

```
1         WITH estadisticas AS (
2         SELECT
3         (SELECT COUNT(DISTINCT identifier) FROM mobility_data)
             AS total_individuos,
4         (SELECT COUNT(*)
5         FROM (SELECT identifier
6         FROM mobility_data
7         GROUP BY identifier
8         HAVING COUNT(*) > 3) AS individuos_filtrados
9         ) AS individuos_con_mas_de_3
10        )
11        SELECT
12        total_individuos,
13        individuos_con_mas_de_3,
14        ROUND((individuos_con_mas_de_3 * 100.0 /
             total_individuos), 2) AS
             porcentaje_individuos_con_mas_de_3
15        FROM estadisticas;
```

Código B.12: Query para contar individuos con más de 3 registros

```
1         WITH estadisticas AS (
2         SELECT
3         (SELECT COUNT(DISTINCT identifier) FROM mobility_data)
             AS total_individuos,
4         (SELECT COUNT(*)
5         FROM (SELECT identifier
6         FROM mobility_data
7         WHERE device_horizontal_accuracy < 20
8         GROUP BY identifier
9         HAVING COUNT(*) > 3) AS individuos_filtrados
10        ) AS individuos_con_mas_de_3_y_precision
11        )
12
13
14        SELECT
15        total_individuos,
16        individuos_con_mas_de_3_y_precision,
17        ROUND((individuos_con_mas_de_3_y_precision * 100.0 /
             total_individuos), 2) AS
             porcentaje_individuos_condicion
18        FROM estadisticas;
```

Código B.13: Query para contar individuos con más de 3 registros y precisión de GPS

```
1         WITH movement_data AS (
2         SELECT
3         identifier,
4         timestamp::timestamp as ts,
```

```
5           device_lat,
6           device_lon,
7           device_horizontal_accuracy,
8           LAG(device_lat) OVER (PARTITION BY identifier ORDER BY
               timestamp) as prev_lat,
9           LAG(device_lon) OVER (PARTITION BY identifier ORDER BY
               timestamp) as prev_lon,
10          CASE
11          WHEN ABS(COALESCE(LAG(device_lat) OVER (PARTITION BY
               identifier ORDER BY timestamp), device_lat) -
               device_lat) > 0.001
12          OR ABS(COALESCE(LAG(device_lon) OVER (PARTITION BY
               identifier ORDER BY timestamp), device_lon) -
               device_lon) > 0.001
13          THEN 1 ELSE 0
14          END as is_movement
15          FROM mobility_data
16          WHERE device_lat IS NOT NULL
17          AND device_lon IS NOT NULL
18          AND device_horizontal_accuracy < 100
19          ),
20
21          best_trajectories AS (
22          SELECT
23          identifier,
24          COUNT(*) as records_count,
25          EXTRACT(DAYS FROM (MAX(ts) - MIN(ts))) as
               time_span_days,
26          COUNT(DISTINCT DATE(ts)) as active_days_count,
27          AVG(device_horizontal_accuracy) as avg_accuracy_meters,
28          SUM(is_movement) as movement_points,
29          (MAX(device_lat) - MIN(device_lat)) + (MAX(device_lon)
               - MIN(device_lon)) as spatial_range
30          FROM movement_data
31          GROUP BY identifier
32          HAVING COUNT(*) >= 50
33          AND EXTRACT(DAYS FROM (MAX(ts) - MIN(ts))) >= 1
34          AND COUNT(DISTINCT DATE(ts)) >= 2
35
36          trajectory_scores AS (
37          SELECT
38          *,
39          -- Score compuesto final
40          (LEAST(100, records_count / 5.0) * 0.25 +
41          LEAST(100, time_span_days / 0.3) * 0.2 +
42          LEAST(100, active_days_count::float / NULLIF(
               time_span_days, 0) * 100) * 0.2 +
43          GREATEST(0, 100 - avg_accuracy_meters) * 0.15 +
44          LEAST(100, movement_points / 1.0) * 0.1 +
45          LEAST(100, spatial_range / 0.01 * 100) * 0.1
46          ) as trajectory_quality_score
```

```
47          FROM best_trajectories
48          )
49          SELECT
50          identifier,
51          records_count,
52          time_span_days,
53          active_days_count,
54          ROUND(active_days_count::numeric / NULLIF(
               time_span_days, 0), 2) as activity_ratio,
55          ROUND(avg_accuracy_meters::numeric, 2) as
               avg_accuracy_meters,
56          movement_points,
57          ROUND(spatial_range::numeric, 6) as spatial_diversity,
58          ROUND(trajectory_quality_score::numeric, 2) as
               quality_score,
59
60          CASE
61          WHEN trajectory_quality_score >= 80 THEN 'EXCELENTE'
62          WHEN trajectory_quality_score >= 65 THEN 'MUY␣BUENA'
63          WHEN trajectory_quality_score >= 50 THEN 'BUENA'
64          WHEN trajectory_quality_score >= 35 THEN 'REGULAR'
65          ELSE 'BAJA'
66          END as quality_category,
67
68          ROW_NUMBER() OVER (ORDER BY trajectory_quality_score
               DESC) as overall_rank
69
70          FROM trajectory_scores
71          WHERE trajectory_quality_score >= 35
72          ORDER BY trajectory_quality_score DESC
73          LIMIT 100;
```

Código B.14: Query para calcular la calidad de las trayectorias de movilidad

```
1         import pandas as pd
2         import numpy as np
3         import os
4         from pathlib import Path
5         import logging
6         from tqdm import tqdm
7         import sys
8
9         logging.basicConfig(level=logging.INFO, format='%(
              asctime)s - %(levelname)s - %(message)s')
10        logger = logging.getLogger(__name__)
11
12        class PedestrianTrajectoryFilter:
13        def __init__(self, csv_file):
14        self.csv_file = csv_file
15
16        self.PEDESTRIAN_MEAN = 1.34  # m/s
17        self.PEDESTRIAN_STD = 0.37   # m/s
18        self.MIN_SPEED = 0.6   # m/s (media - 2*std)
19        self.MAX_SPEED = 2.08  # m/s (media + 2*std)
20
21        self.MIN_SPEED_KMH = self.MIN_SPEED * 3.6  # 2.16 km/h
22        self.MAX_SPEED_KMH = self.MAX_SPEED * 3.6  # 7.488 km/h
23
24        self.MIN_QUALITY_SCORE = 35  # Incluye REGULAR, BUENA,
              MUY BUENA y EXCELENTE
25
26        logger.info(f"Rango de velocidad peatonal: {self.
              MIN_SPEED:.2f} - {self.MAX_SPEED:.2f} m/s")
27        logger.info(f"Rango en km/h: {self.MIN_SPEED_KMH:.2f} -
              {self.MAX_SPEED_KMH:.2f} km/h")
28        logger.info(f"Calidad mnima requerida: {self.
              MIN_QUALITY_SCORE} puntos (REGULAR o superior)")
29
30        def classify_trajectories(self, df):
31        logger.info(" Clasificando trayectorias por calidad..."
              )
32
33        # Convertir timestamp a datetime
34        logger.info("   Convirtiendo timestamps...")
35        df['timestamp'] = pd.to_datetime(df['timestamp'],
              errors='coerce')
36        df = df.dropna(subset=['timestamp'])
37
38        df_valid = df[
39        (df['device_lat'].notna()) &
40        (df['device_lon'].notna()) &
41        (df['device_horizontal_accuracy'] < 100)
42        ].copy()
43
```

```
44          logger.info(f"␣␣␣Registros␣vlidos:␣{len(df_valid):,}")
45          logger.info("␣␣␣Calculando␣mtricas␣de␣movimiento...")
46
47          # Detectar movimiento (cambio significativo en
                coordenadas)
48          df_valid = df_valid.sort_values(['identifier', '
                timestamp'])
49          df_valid['prev_lat'] = df_valid.groupby('identifier')['
                device_lat'].shift(1)
50          df_valid['prev_lon'] = df_valid.groupby('identifier')['
                device_lon'].shift(1)
51
52          df_valid['is_movement'] = (
53          (np.abs(df_valid['device_lat'] - df_valid['prev_lat'])
                > 0.001) |
54          (np.abs(df_valid['device_lon'] - df_valid['prev_lon'])
                > 0.001)
55          ).astype(int)
56
57          logger.info("␣␣␣Agregando␣mtricas␣por␣persona...")
58          trajectory_metrics = df_valid.groupby('identifier').agg
                ({
59                  'timestamp': ['count', 'min', 'max'],
60                  'device_horizontal_accuracy': 'mean',
61                  'is_movement': 'sum',
62                  'device_lat': ['min', 'max'],
63                  'device_lon': ['min', 'max']
64          }).reset_index()
65
66          trajectory_metrics.columns = [
67          'identifier', 'records_count', 'ts_min', 'ts_max',
68          'avg_accuracy_meters', 'movement_points',
69          'lat_min', 'lat_max', 'lon_min', 'lon_max'
70          ]
71
72          trajectory_metrics['time_span_days'] = (
73          trajectory_metrics['ts_max'] - trajectory_metrics['
                ts_min']
74          ).dt.total_seconds() / 86400
75
76          logger.info("␣␣␣Calculando␣das␣activos...")
77          active_days = df_valid.groupby('identifier')['timestamp
                '].apply(
78          lambda x: x.dt.date.nunique()
79          ).reset_index()
80          active_days.columns = ['identifier', 'active_days_count
                ']
81          trajectory_metrics = trajectory_metrics.merge(
                active_days, on='identifier')
82
83          trajectory_metrics['spatial_range'] = (
```

```
84          (trajectory_metrics['lat_max'] - trajectory_metrics['
               lat_min']) +
85          (trajectory_metrics['lon_max'] - trajectory_metrics['
               lon_min'])
86          )
87
88          trajectory_metrics = trajectory_metrics[
89          (trajectory_metrics['records_count'] >= 50) &
90          (trajectory_metrics['time_span_days'] >= 1) &
91          (trajectory_metrics['active_days_count'] >= 2)
92          ]
93
94          logger.info(f"␣␣␣Trayectorias␣que␣cumplen␣criterios␣
               mnimos:␣{len(trajectory_metrics)}")
95          logger.info("␣␣␣Calculando␣scores␣de␣calidad...")
96
97          trajectory_metrics['score_volume'] = np.minimum(100,
               trajectory_metrics['records_count'] / 5.0) * 0.25
98          trajectory_metrics['score_duration'] = np.minimum(100,
               trajectory_metrics['time_span_days'] / 0.3) * 0.2
99
100         trajectory_metrics['activity_ratio'] = (
101         trajectory_metrics['active_days_count'] /
102         trajectory_metrics['time_span_days'].replace(0, np.nan)
103         )
104         trajectory_metrics['score_regularity'] = np.minimum
               (100, trajectory_metrics['activity_ratio'] * 100) *
               0.2
105
106         trajectory_metrics['score_accuracy'] = np.maximum(0,
               100 - trajectory_metrics['avg_accuracy_meters']) *
               0.15
107         trajectory_metrics['score_mobility'] = np.minimum(100,
               trajectory_metrics['movement_points'] / 1.0) * 0.1
108         trajectory_metrics['score_diversity'] = np.minimum(100,
                trajectory_metrics['spatial_range'] / 0.01 * 100) *
                0.1
109
110         trajectory_metrics['quality_score'] = (
111         trajectory_metrics['score_volume'] +
112         trajectory_metrics['score_duration'] +
113         trajectory_metrics['score_regularity'] +
114         trajectory_metrics['score_accuracy'] +
115         trajectory_metrics['score_mobility'] +
116         trajectory_metrics['score_diversity']
117         )
118
119         trajectory_metrics['quality_category'] = pd.cut(
120         trajectory_metrics['quality_score'],
121         bins=[0, 35, 50, 65, 80, 100],
122         labels=['BAJA', 'REGULAR', 'BUENA', 'MUY␣BUENA', '
```

```
            EXCELENTE'],
123         include_lowest=True
124         )
125
126         trajectory_metrics = trajectory_metrics[
               trajectory_metrics['quality_score'] >= self.
               MIN_QUALITY_SCORE]
127         trajectory_metrics['overall_rank'] = trajectory_metrics
               ['quality_score'].rank(
128         ascending=False, method='first'
129         ).astype(int)
130
131         trajectory_metrics = trajectory_metrics.sort_values('
               overall_rank')
132         logger.info(f"␣␣␣␣Trayectorias␣clasificadas␣con␣calidad
               ␣REGULAR␣o␣superior:␣{len(trajectory_metrics)}")
133         return trajectory_metrics
134
135         def get_quality_identifiers(self, trajectory_metrics):
136         return trajectory_metrics['identifier'].tolist()
137
138         def calculate_haversine_distance(self, lat1, lon1, lat2
               , lon2):
139         R = 6371000  # Radio de la Tierra en metros
140
141         lat1_rad = np.radians(lat1)
142         lon1_rad = np.radians(lon1)
143         lat2_rad = np.radians(lat2)
144         lon2_rad = np.radians(lon2)
145
146         dlat = lat2_rad - lat1_rad
147         dlon = lon2_rad - lon1_rad
148
149         a = np.sin(dlat/2)**2 + np.cos(lat1_rad) * np.cos(
               lat2_rad) * np.sin(dlon/2)**2
150         c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
151
152         return R * c
153
154         def calculate_speeds_for_trajectory(self, trajectory_df
               ):
155         if len(trajectory_df) < 2:
156         return trajectory_df
157
158         trajectory_df = trajectory_df.sort_values('timestamp').
               reset_index(drop=True)
159
160         speeds = []
161
162         for i in range(1, len(trajectory_df)):
163         prev_row = trajectory_df.iloc[i-1]
```

```
164         curr_row = trajectory_df.iloc[i]
165
166         distance = self.calculate_haversine_distance(
167         prev_row['device_lat'], prev_row['device_lon'],
168         curr_row['device_lat'], curr_row['device_lon']
169         )
170
171         time_diff = (curr_row['timestamp'] - prev_row['
               timestamp']).total_seconds()
172
173         if time_diff > 1:
174         speed_ms = distance / time_diff
175         speeds.append(speed_ms)
176         else:
177         speeds.append(np.nan)
178
179         trajectory_df['speed_ms'] = [np.nan] + speeds
180
181         return trajectory_df
182
183         def segment_trajectory_by_speed(self, trajectory_df):
184         if trajectory_df.empty or len(trajectory_df) < 2:
185         return []
186
187         trajectory_df = self.calculate_speeds_for_trajectory(
               trajectory_df)
188
189         is_valid = trajectory_df['speed_ms'].isna() | \
190         ((trajectory_df['speed_ms'] >= self.MIN_SPEED) &
191         (trajectory_df['speed_ms'] <= self.MAX_SPEED))
192
193         segments = []
194         current_segment = []
195
196         for idx in trajectory_df.index:
197         if is_valid.loc[idx]:
198         current_segment.append(idx)
199         else:
200         if len(current_segment) >= 2:
201         segment_df = trajectory_df.loc[current_segment].copy()
202         segments.append(segment_df)
203         current_segment = []
204
205         if len(current_segment) >= 2:
206         segment_df = trajectory_df.loc[current_segment].copy()
207         segments.append(segment_df)
208
209         return segments
210
211         def process_person_trajectories(self, identifier, df):
212         try:
```

```
213         person_data = df[df['identifier'] == identifier].copy()
214
215         if person_data.empty:
216         return []
217
218         person_data['timestamp'] = pd.to_datetime(person_data['
               timestamp'], errors='coerce')
219         person_data = person_data.dropna(subset=['timestamp'])
220
221         if person_data.empty:
222         return []
223
224         person_data = person_data.sort_values('timestamp')
225         person_data['date_group'] = person_data['timestamp'].dt
               .date
226
227         all_pedestrian_segments = []
228
229         for date, day_data in person_data.groupby('date_group')
               :
230         if len(day_data) < 2:
231         continue
232
233         segments = self.segment_trajectory_by_speed(day_data)
234
235         for seg_idx, segment in enumerate(segments):
236         segment = segment.copy()
237         segment['segment_id'] = f"{identifier}_{date}_{seg_idx}
               "
238         segment['segment_date'] = date
239         all_pedestrian_segments.append(segment)
240
241         return all_pedestrian_segments
242
243         except Exception as e:
244         logger.error(f"␣Error␣procesando␣{identifier}:␣{e}")
245         return []
246
247         def generate_summary_statistics(self, all_segments_df,
               trajectory_metrics):
248         try:
249         logger.info("\n" + "="*60)
250         logger.info("ESTADSTICAS␣DE␣TRAYECTORIAS␣PEATONALES")
251         logger.info("="*60)
252
253         total_points = len(all_segments_df)
254         total_persons = all_segments_df['identifier'].nunique()
255         total_segments = all_segments_df['segment_id'].nunique
               ()
256         total_days = all_segments_df['segment_date'].nunique()
257
```

```
258         speed_data = all_segments_df[all_segments_df['speed_ms'
               ].notna()]['speed_ms']
259
260         logger.info(f"Total␣de␣puntos:␣{total_points:,}")
261         logger.info(f"Total␣de␣personas:␣{total_persons}")
262         logger.info(f"Total␣de␣segmentos:␣{total_segments:,}")
263         logger.info(f"Total␣de␣das:␣{total_days}")
264
265         if len(speed_data) > 0:
266         logger.info(f"Velocidad␣promedio:␣{speed_data.mean():.3
               f}␣m/s␣({speed_data.mean()*3.6:.2f}␣km/h)")
267         logger.info(f"Velocidad␣mnima:␣{speed_data.min():.3f}␣m
               /s␣({speed_data.min()*3.6:.2f}␣km/h)")
268         logger.info(f"Velocidad␣mxima:␣{speed_data.max():.3f}␣m
               /s␣({speed_data.max()*3.6:.2f}␣km/h)")
269
270         logger.info("\n␣Distribucin␣por␣categora␣de␣calidad:")
271         quality_counts = trajectory_metrics['quality_category'
               ].value_counts().sort_index()
272         for category, count in quality_counts.items():
273         percentage = (count / len(trajectory_metrics)) * 100
274         logger.info(f"␣␣␣{category}:␣{count}␣personas␣({
               percentage:.1f}%)")
275
276         stats_per_person = all_segments_df.groupby('identifier'
               ).agg({
277               'segment_id': lambda x: x.nunique(),
278               'speed_ms': lambda x: x[x.notna()].mean() if x.
                  notna().any() else np.nan
279         }).reset_index()
280
281         stats_per_person.columns = ['identifier', 'segments', '
               avg_speed']
282         stats_per_person['points'] = all_segments_df.groupby('
               identifier').size().values
283
284         quality_map = trajectory_metrics[['identifier', '
               quality_score', 'quality_category']].set_index('
               identifier')
285         stats_per_person = stats_per_person.merge(quality_map,
               left_on='identifier', right_index=True, how='left')
286
287         stats_per_person = stats_per_person.sort_values('
               quality_score', ascending=False)
288
289         logger.info("\n␣Estadsticas␣por␣persona␣(top␣10␣por␣
               calidad):")
290         for idx, row in stats_per_person.head(10).iterrows():
291         logger.info(f"␣␣{row['identifier']}␣({row['
               quality_category']}):␣{row['points']:,}␣puntos,␣"
292         f"{row['segments']}␣segmentos,␣vel.␣prom:␣{row['
```

```
            avg_speed']:.3f} m/s")

        Path("pedestrian_analysis").mkdir(exist_ok=True)
        stats_per_person.to_csv('pedestrian_analysis/
            statistics_per_person.csv', index=False)
        logger.info("\n Estadsticas guardadas en
            pedestrian_analysis/statistics_per_person.csv")

        return stats_per_person

        except Exception as e:
        logger.error(f" Error generando estadsticas: {e}")
        return None

        def run_filtering(self):
        try:
        logger.info("="*60)
        logger.info("FILTRADO DE TRAYECTORIAS PEATONALES - 
            TODAS LAS REGULARES+")
        logger.info("="*60)
        logger.info(f"Archivo CSV: {self.csv_file}")
        logger.info(f"Rango de velocidad: {self.MIN_SPEED:.2f} 
            - {self.MAX_SPEED:.2f} m/s")
        logger.info(f"                    {self.MIN_SPEED_KMH
            :.2f} - {self.MAX_SPEED_KMH:.2f} km/h")
        logger.info(f"Calidad mnima: {self.MIN_QUALITY_SCORE} 
            puntos (REGULAR o superior)")
        logger.info("="*60)

        if not os.path.exists(self.csv_file):
        logger.error(f" Archivo no encontrado: {self.csv_file}"
            )
        return

        logger.info("\n Cargando datos del CSV...")
        chunk_size = 1_000_000
        chunks = []

        for chunk in tqdm(pd.read_csv(self.csv_file, chunksize=
            chunk_size),
        desc="Leyendo CSV"):
        chunks.append(chunk)

        df = pd.concat(chunks, ignore_index=True)
        logger.info(f" Datos cargados: {len(df):,} registros")

        trajectory_metrics = self.classify_trajectories(df)

        Path("pedestrian_analysis").mkdir(exist_ok=True)
        trajectory_metrics.to_csv('pedestrian_analysis/
            trajectory_classification.csv', index=False)
```

```
334         logger.info(f"␣Clasificacin␣guardada␣en␣
               pedestrian_analysis/trajectory_classification.csv")
335
336         logger.info("\n␣Distribucin␣de␣calidad␣(REGULAR␣o␣
               superior):")
337         quality_dist = trajectory_metrics['quality_category'].
               value_counts().sort_index()
338         for category, count in quality_dist.items():
339         percentage = (count / len(trajectory_metrics)) * 100
340         logger.info(f"␣␣␣{category}:␣{count}␣personas␣({
               percentage:.1f}%)")
341
342         quality_identifiers = self.get_quality_identifiers(
               trajectory_metrics)
343         logger.info(f"\n␣Total␣de␣identificadores␣seleccionados
               :␣{len(quality_identifiers)}")
344
345         logger.info("\nTop␣10␣personas␣por␣calidad:")
346         top_10 = trajectory_metrics.nsmallest(10, 'overall_rank
               ')
347         for _, row in top_10.iterrows():
348         logger.info(f"␣␣␣#{row['overall_rank']:2d}.␣{row['
               identifier']}:␣"
349         f"{row['quality_score']:.2f}␣puntos␣({row['
               quality_category']})")
350
351         logger.info(f"\n␣Filtrando␣datos␣de␣{len(
               quality_identifiers)}␣personas...")
352         df_quality = df[df['identifier'].isin(
               quality_identifiers)].copy()
353         logger.info(f"␣␣␣Registros␣seleccionados:␣{len(
               df_quality):,}")
354
355         logger.info(f"\n␣Procesando␣trayectorias␣peatonales..."
               )
356
357         all_segments = []
358
359         for identifier in tqdm(quality_identifiers, desc="
               Procesando␣personas"):
360         segments = self.process_person_trajectories(identifier,
                df_quality)
361         if segments:
362         all_segments.extend(segments)
363
364         if not all_segments:
365         logger.error("␣No␣se␣generaron␣segmentos␣peatonales")
366         return
367
368         logger.info(f"\n␣Guardando␣resultados...")
369         all_segments_df = pd.concat(all_segments, ignore_index=
```

```
            True)
370
371         output_file = 'pedestrian_analysis/
               pedestrian_trajectories_all.csv'
372         all_segments_df.to_csv(output_file, index=False)
373         logger.info(f"␣Trayectorias␣guardadas␣en␣{output_file}"
               )
374         logger.info(f"␣␣␣Total␣de␣puntos␣guardados:␣{len(
               all_segments_df):,}")
375
376         self.generate_summary_statistics(all_segments_df,
               trajectory_metrics)
377
378         logger.info("\n" + "="*60)
379         logger.info("␣PROCESO␣COMPLETADO␣EXITOSAMENTE")
380         logger.info("="*60)
381         logger.info(f"\nPersonas␣procesadas:␣{len(
               quality_identifiers)}")
382         logger.info(f"Puntos␣peatonales␣extrados:␣{len(
               all_segments_df):,}")
383         logger.info("\nArchivos␣generados:")
384         logger.info("␣␣1.␣pedestrian_analysis/
               trajectory_classification.csv")
385         logger.info("␣␣2.␣pedestrian_analysis/
               pedestrian_trajectories_all.csv")
386         logger.info("␣␣3.␣pedestrian_analysis/
               statistics_per_person.csv")
387
388         except Exception as e:
389         logger.error(f"␣Error␣en␣el␣proceso␣de␣filtrado:␣{e}")
390         import traceback
391         logger.error(traceback.format_exc())
392
393         def main():
394         if len(sys.argv) < 2:
395         logger.error("Uso:␣python␣pedestrian_trajectory_filter.
               py␣<archivo.csv>")
396         logger.info("Ejemplo:␣python␣
               pedestrian_trajectory_filter.py␣
               Mobility_Data_Slim_DeDuplicate.csv")
397         sys.exit(1)
398
399         csv_file = sys.argv[1]
400         filter = PedestrianTrajectoryFilter(csv_file)
401         filter.run_filtering()
402
403         if __name__ == "__main__":
404         main()
```

Código B.15: pedestrian_trajectories.py, Clasificación de trajectorias peatonales

```
1         import pandas as pd
2         import sys
3         import os
4         from tqdm import tqdm
5         from collections import defaultdict
6         import matplotlib.pyplot as plt
7
8         def analyze_multi_day_users(input_file):
9         print(f"\nIniciando␣anlisis␣multi-da␣del␣archivo:␣{
              input_file}")
10        filename = os.path.splitext(os.path.basename(input_file
              ))[0]
11
12        chunksize = 1_000_000
13
14        identifier_days = defaultdict(set)
15        total_records = 0
16
17        print("\nProcesando␣datos␣por␣chunks...")
18
19        total_chunks = sum(1 for _ in pd.read_csv(input_file,
              usecols=['identifier', 'timestamp'], chunksize=
              chunksize))
20
21        with tqdm(total=total_chunks, unit='␣chunk', desc="
              Procesando") as pbar:
22        for chunk in pd.read_csv(input_file, usecols=['
              identifier', 'timestamp'], chunksize=chunksize):
23        # Limpiar datos
24        chunk = chunk.dropna()
25
26        # Manejar formato con microsegundos: '2022-11-07
              02:04:21.000'
27        chunk['timestamp'] = pd.to_datetime(chunk['timestamp'],
               format='mixed', errors='coerce')
28
29        chunk = chunk.dropna(subset=['timestamp'])
30
31        chunk['date'] = chunk['timestamp'].dt.date
32
33        for identifier, group in chunk.groupby('identifier'):
34        unique_dates = set(group['date'])
35        identifier_days[identifier].update(unique_dates)
36
37        total_records += len(chunk)
38        pbar.update(1)
39
40        print(f"␣Procesados␣{total_records:,}␣registros")
41
42        print("\nAnalizando␣patrones␣multi-da...")
```

```
43
44          days_per_identifier = {identifier: len(dates) for
               identifier, dates in identifier_days.items()}
45
46          total_identifiers = len(days_per_identifier)
47          multi_day_identifiers = sum(1 for days in
               days_per_identifier.values() if days > 1)
48          single_day_identifiers = total_identifiers -
               multi_day_identifiers
49
50          days_distribution = defaultdict(int)
51          for days_count in days_per_identifier.values():
52          days_distribution[days_count] += 1
53
54          print("\n" + "="*60)
55          print("RESULTADOS␣DEL␣ANLISIS␣MULTI-DA")
56          print("="*60)
57          print(f"Total␣de␣identificadores␣nicos:␣{
               total_identifiers:,}")
58          print(f"Identificadores␣con␣registros␣en␣un␣solo␣da:␣{
               single_day_identifiers:,}␣({single_day_identifiers/
               total_identifiers*100:.2f}%)")
59          print(f"Identificadores␣con␣registros␣en␣mltiples␣das:␣
               {multi_day_identifiers:,}␣({multi_day_identifiers/
               total_identifiers*100:.2f}%)")
60
61          if multi_day_identifiers > 0:
62          max_days = max(days_per_identifier.values())
63          avg_days_multi_day = sum(days for days in
               days_per_identifier.values() if days > 1) /
               multi_day_identifiers
64
65          print(f"\nEstadsticas␣de␣identificadores␣multi-da:")
66          print(f"Mximo␣nmero␣de␣das␣por␣identificador:␣{max_days
               }")
67          print(f"Promedio␣de␣das␣(solo␣multi-da):␣{
               avg_days_multi_day:.2f}")
68
69          print(f"\nDistribucin␣de␣das␣por␣identificador:")
70          print("-" * 40)
71
72          sorted_distribution = sorted(days_distribution.items())
73          for days, count in sorted_distribution[:20]:
74          percentage = count / total_identifiers * 100
75          print(f"{days:2d}␣da(s):␣{count:8,}␣identificadores␣({
               percentage:5.2f}%)")
76
77          if len(sorted_distribution) > 20:
78          remaining_count = sum(count for days, count in
               sorted_distribution[20:])
79          remaining_percentage = remaining_count /
```

```
            total_identifiers * 100
80          print(f"Otros:␣␣␣␣␣␣{remaining_count:8,}␣
                identificadores␣({remaining_percentage:5.2f}%)")
81
82          print(f"\nGenerando␣visualizacin...")
83
84          plot_data = dict(sorted_distribution[:30])
85
86          plt.figure(figsize=(15, 8))
87          bars = plt.bar(plot_data.keys(), plot_data.values(),
                alpha=0.7, edgecolor='black')
88
89          plt.title(f'Distribucin␣de␣Das␣por␣Identificador\
                nArchivo:␣{filename}', fontsize=16, pad=20)
90          plt.xlabel('Nmero␣de␣Das␣con␣Registros', fontsize=14)
91          plt.ylabel('Cantidad␣de␣Identificadores', fontsize=14)
92          plt.yscale('log')  # Escala logartmica para mejor
                visualizacin
93          plt.grid(True, alpha=0.3, axis='y')
94
95          for i, (days, count) in enumerate(list(plot_data.items
                ())[:10]):
96          if count > 0:
97          percentage = count / total_identifiers * 100
98          plt.text(days, count * 1.1, f'{percentage:.1f}%',
99          ha='center', va='bottom', fontsize=10, fontweight='bold
                ')
100
101         # Estadsticas en el grfico
102         stats_text = (
103         f"Total␣identificadores:␣{total_identifiers:,}\n"
104         f"Multi-da:␣{multi_day_identifiers:,}␣({
                multi_day_identifiers/total_identifiers*100:.1f}%)\n
                "
105         f"Un␣solo␣da:␣{single_day_identifiers:,}␣({
                single_day_identifiers/total_identifiers*100:.1f}%)"
106         )
107
108         plt.text(0.98, 0.98, stats_text,
109         transform=plt.gca().transAxes,
110         fontsize=12, ha='right', va='top',
111         bbox=dict(boxstyle='round', facecolor='white', alpha
                =0.9))
112
113         os.makedirs("img", exist_ok=True)
114         output_path = os.path.join("img", f"multi_day_analysis_
                {filename}.png")
115         plt.tight_layout()
116         plt.savefig(output_path, dpi=300, bbox_inches='tight')
117         plt.close()
118
```

```
119         print(f"\nGenerando␣archivo␣de␣resumen...")
120 
121         summary_data = []
122         for identifier, dates in identifier_days.items():
123         summary_data.append({
124                 'identifier': identifier,
125                 'num_days': len(dates),
126                 'first_date': min(dates),
127                 'last_date': max(dates),
128                 'date_span_days': (max(dates) - min(dates)).
                       days + 1 if len(dates) > 1 else 1
129         })
130 
131         summary_df = pd.DataFrame(summary_data)
132         summary_path = f"{filename}_multi_day_summary.csv"
133         summary_df.to_csv(summary_path, index=False)
134 
135         print(f"␣Anlisis␣completado␣exitosamente")
136         print(f"␣Grfico␣guardado␣en:␣{output_path}")
137         print(f"␣Resumen␣guardado␣en:␣{summary_path}")
138         print(f"␣Total␣registros␣procesados:␣{total_records:,}"
               )
139 
140         return {
141                 'total_identifiers': total_identifiers,
142                 'multi_day_identifiers': multi_day_identifiers,
143                 'single_day_identifiers':
                       single_day_identifiers,
144                 'days_distribution': dict(days_distribution),
145                 'max_days': max(days_per_identifier.values())
                       if days_per_identifier else 0
146         }
147 
148         def main():
149         if len(sys.argv) < 2:
150         print("Error:␣Debe␣especificar␣un␣archivo␣CSV␣como␣
               argumento")
151         print("Uso:␣python␣multi_day_analysis.py␣<archivo.csv>"
               )
152         sys.exit(1)
153 
154         csv_file = sys.argv[1]
155 
156         # Verificar que el archivo existe
157         if not os.path.exists(csv_file):
158         print(f"Error:␣El␣archivo␣'{csv_file}'␣no␣existe")
159         sys.exit(1)
160 
161         try:
162         results = analyze_multi_day_users(csv_file)
163         print(f"\n␣Anlisis␣multi-da␣completado␣exitosamente!")
```

```
164
165         except Exception as e:
166         print(f"\n␣Error␣durante␣el␣anlisis:␣{str(e)}")
167         sys.exit(1)
168
169         if __name__ == "__main__":
170         main()
```

Código B.16: routine_individuals.py, Análisis de la distribución temporal y frecuencia de individuos

Código B.17: .py,

Código B.18: .py,

Referencias

[1] Autor referencia 1

[2] Autor referencia 2

[3] Autor referencia 3