



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería  
Informática**

**Implementación del algoritmo  
TRA-CLUS Mejorado para el  
análisis de tráfico**



Presentado por Álvaro González Delgado  
en Universidad de Burgos — 24 de diciembre  
de 2024

Tutor: Bruno Baruque Zanon y Hector  
Cogollos Adrian







UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Álvaro González Delgado, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 24 de diciembre de 2024

Vº. Bº. del Tutor:

D. nombre tutor

Vº. Bº. del co-tutor:

D. nombre co-tutor





## **Resumen**

El presente proyecto tiene como objetivo el desarrollo de una aplicación web interactiva para la visualización y análisis de trayectorias mediante la implementación del algoritmo de agrupamiento de trayectorias TRACLUS y el estudio de sus variantes. La solución integra herramientas de procesamiento de datos y visualización para facilitar la interpretación de patrones espaciales en trayectorias, con énfasis en eficiencia y aplicabilidad en entornos de Big Data. Se emplearon metodologías ágiles y tecnologías modernas como Dash y scikit-learn para garantizar un diseño modular y adaptable.

## **Descriptores**

TRACLUS, análisis de trayectorias, agrupamiento, visualización de datos, Dash, scikit-learn, Big Data.

**Abstract**

This project aims to develop an interactive web application for the visualization and analysis of trajectories through the implementation of the TRACLUS trajectory clustering algorithm and the study of its variants. The solution integrates data processing and visualization tools to facilitate the interpretation of spatial patterns in trajectories, focusing on efficiency and applicability in Big Data environments. Agile methodologies and modern technologies such as Dash and scikit-learn were employed to ensure a modular and adaptable design.

**Keywords**

TRACLUS, trajectory analysis, clustering, data visualization, Dash, scikit-learn, Big Data.

---

# Índice general

---

Índice general	iii
Índice de figuras	v
Índice de tablas	vii
<b>1. Introducción</b>	<b>1</b>
<b>2. Objetivos del proyecto</b>	<b>3</b>
2.1. Objetivos generales . . . . .	3
2.2. Objetivos técnicos . . . . .	3
2.3. Objetivos personales . . . . .	4
<b>3. Conceptos teóricos</b>	<b>5</b>
3.1. TRACLUS . . . . .	5
3.2. Clustering . . . . .	7
<b>4. Técnicas y herramientas</b>	<b>11</b>
4.1. GitHub . . . . .	11
4.2. Texmaker . . . . .	12
4.3. CSS . . . . .	12
4.4. Python . . . . .	13
<b>5. Aspectos relevantes del desarrollo del proyecto</b>	<b>17</b>
5.1. Herramientas de control de tareas: Kanban y Scrum . . . . .	17
5.2. Formación y aprendizaje necesario . . . . .	18
5.3. Investigación y experimentación . . . . .	22
5.4. Despliegue de la aplicación . . . . .	39

5.5. Pruebas funcionales . . . . .	41
5.6. Testing . . . . .	52
<b>6. Trabajos relacionados</b>	<b>57</b>
6.1. Estudio del algoritmo TRA-CLUS . . . . .	57
6.2. Biblioteca TRA-CLUS . . . . .	58
6.3. Impacto en el proyecto . . . . .	59
<b>7. Conclusiones y Líneas de trabajo futuras</b>	<b>61</b>
7.1. Conclusiones . . . . .	61
7.2. Líneas de Trabajo Futuras . . . . .	62

---

# Índice de figuras

---

5.1.	Representación de clusters . . . . .	30
5.2.	Segmentos por cada cluster . . . . .	30
5.3.	Representación de trayectorias . . . . .	31
5.4.	Representación de clusters . . . . .	31
5.5.	Segmentos por cada cluster . . . . .	32
5.6.	Representación de trayectorias . . . . .	32
5.7.	Representación de clusters . . . . .	33
5.8.	Segmentos por cada cluster . . . . .	33
5.9.	Representación de trayectorias . . . . .	34
5.10.	Representación de clusters . . . . .	35
5.11.	Segmentos por cada cluster . . . . .	35
5.12.	Representación de trayectorias . . . . .	36
5.13.	Representación de clusters . . . . .	36
5.14.	Segmentos por cada cluster . . . . .	37
5.15.	Representación de trayectorias . . . . .	37
5.16.	Mapa de trayectorias sin tratar del conjunto Citi Bikes . . . . .	43
5.17.	Mapa de trayectorias Geolife, 10 filas . . . . .	49
5.18.	Mapa trayectorias resultantes OPTICS y DBSCAN, 10 filas . . . . .	49
5.19.	Mapa trayectorias resultantes HDBSCAN, 10 filas . . . . .	50
5.20.	Mapa trayectorias resultantes Agglomerative, 10 filas . . . . .	50
5.21.	Mapa trayectorias resultantes Spectral, 10 filas . . . . .	51
5.22.	Mapa de trayectorias MoveBank, 10 filas . . . . .	51
5.23.	Mapa de trayectorias Taxis, 10 filas . . . . .	52
5.24.	Mapa trayectorias resultantes OPTICS y DBSCAN, 10 filas . . . . .	52
5.25.	Mapa trayectorias resultantes HDBSCAN, 10 filas . . . . .	53
5.26.	Mapa trayectorias resultantes Agglomerative, 10 filas . . . . .	53
5.27.	Mapa trayectorias resultantes Spectral, 10 filas . . . . .	54

5.28. Error nomenclatura. . . . .	55
5.29. Código innecesario. . . . .	55
5.30. Variable sin usar necesaria. . . . .	55
5.31. Importación excesiva. . . . .	55
5.32. Complejidad grande. . . . .	56

---

## **Índice de tablas**

---



---

# 1. Introducción

---

El presente trabajo aborda el desarrollo e implementación de un sistema orientado al análisis de grandes volúmenes de datos geoespaciales mediante técnicas de agrupamiento o *clustering*. En concreto, se ha centrado en el algoritmo TRA-CLUS, diseñado para segmentar y agrupar trayectorias geográficas, y en la creación de una aplicación web interactiva que facilite la interpretación y comparación de los resultados obtenidos con diferentes algoritmos de clustering.

El análisis de trayectorias es una tarea fundamental en campos como la movilidad urbana, la gestión del tráfico y la planificación territorial. Sin embargo, la complejidad y el volumen de datos asociados a estas áreas hacen necesario el uso de algoritmos eficientes y herramientas visuales que permitan extraer información útil de manera intuitiva. Este proyecto tiene como objetivo ofrecer una solución integral que combine una implementación optimizada del algoritmo TRA-CLUS con una interfaz visual de fácil uso.

La estructura de esta memoria está diseñada para presentar de manera clara y ordenada el desarrollo del proyecto:

- **Capítulo 1 - Introducción:** Introduce el contexto general del proyecto, sus motivaciones y la organización de esta memoria.
- **Capítulo 2 - Objetivos del Proyecto:** Detalla los objetivos generales, técnicos y personales que han guiado el desarrollo del trabajo.
- **Capítulo 3 - Conceptos Teóricos:** Expone los fundamentos teóricos relacionados con el análisis de trayectorias, el clustering y las técnicas utilizadas en el desarrollo del proyecto.

- **Capítulo 4 - Técnicas y Herramientas:** Describe las tecnologías y herramientas empleadas, justificando su elección y explicando su aplicación dentro del proyecto.
- **Capítulo 5 - Aspectos Relevantes del Proyecto:** Presenta las decisiones clave tomadas durante el desarrollo, los desafíos enfrentados y las soluciones implementadas.
- **Capítulo 6 - Trabajos Relacionados:** Revisa proyectos y estudios previos que guardan relación con este trabajo, destacando sus aportaciones y diferencias con el presente proyecto.
- **Capítulo 7 - Conclusiones y Líneas de Trabajo Futuras:** Resume los resultados alcanzados, las lecciones aprendidas y propone posibles mejoras y extensiones para el proyecto.

En conclusión, este proyecto pretende no solo resolver un problema específico del análisis de trayectorias, sino también sentar las bases para futuras investigaciones y desarrollos en el ámbito del Big Data aplicado a datos geoespaciales.

---

## **2. Objetivos del proyecto**

---

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

### **2.1. Objetivos generales**

- Desarrollar un algoritmo de Big Data denominado TRA-CLUS para el análisis y agrupación de trayectorias geoespaciales.
- Facilitar la interpretación de los datos recogidos mediante representaciones gráficas claras y precisas.
- Realizar comparativas de rendimiento y precisión entre diferentes algoritmos de clustering.
- Diseñar y desarrollar una aplicación web interactiva para mostrar y analizar los resultados obtenidos por el algoritmo.

### **2.2. Objetivos técnicos**

- Desarrollar el algoritmo TRA-CLUS optimizado para grandes volúmenes de datos utilizando herramientas eficientes y escalables.
- Diseñar y construir una aplicación web basada en *Dash* que soporte la visualización de datos y permita la interacción del usuario.

- Aplicar una arquitectura modular y bien documentada que facilite el mantenimiento y la extensibilidad del proyecto.
- Implementar herramientas de visualización como mapas interactivos para representar trayectorias y clusters.
- Aplicar metodologías ágiles como *Scrum* para organizar y gestionar las etapas del desarrollo del proyecto.
- Realizar pruebas unitarias, de integración y de interfaz para garantizar la calidad y fiabilidad del sistema.
- Utilizar *Git* como sistema de control de versiones distribuido y alojar el proyecto en la plataforma GitHub.
- Gestionar el desarrollo y el seguimiento del proyecto utilizando *GitHub Projects* para organizar tareas y monitorear el progreso.

### **2.3. Objetivos personales**

- Realizar una contribución significativa al desarrollo experimental de software aplicado al análisis de Big Data.
- Reforzar y consolidar los conocimientos adquiridos durante la carrera en el ámbito del desarrollo de software y análisis de datos.
- Explorar y aplicar metodologías y herramientas actuales utilizadas en entornos laborales profesionales.
- Adentrarme en el campo del Big Data y el análisis geoespacial para comprender mejor sus desafíos y oportunidades.
- Profundizar en el desarrollo de aplicaciones web modernas que integren múltiples componentes tecnológicos.
- Mejorar habilidades en gestión de proyectos, trabajo colaborativo y resolución de problemas técnicos complejos.

---

## 3. Conceptos teóricos

---

En este apartado se presentan los conceptos teóricos fundamentales que permiten comprender el marco conceptual en el que se desarrolla este trabajo. Estos conceptos proporcionan el contexto necesario para el análisis y desarrollo del estudio realizado.

La discusión se centrará en los principios relacionados con el algoritmo TRACLUS, dado que este ha sido el enfoque principal del estudio y representa la mayor complejidad en su implementación y análisis.

### 3.1. TRACLUS

TRACLUS es un algoritmo de agrupación (*clustering*) especializado en datos de trayectorias, diseñado para identificar patrones comunes de movimiento en conjuntos de datos de trayectorias, como rutas de vehículos, movimientos de animales o trayectorias de fenómenos meteorológicos. A diferencia de los algoritmos tradicionales de *clustering* que agrupan puntos individuales, TRACLUS se enfoca en el agrupamiento de trayectorias completas, descomponiendo cada trayectoria en segmentos y detectando patrones comunes en subtrayectorias específicas. Este enfoque es útil en estudios donde los objetos presentan secuencias de movimiento en el espacio y el tiempo, permitiendo identificar similitudes parciales dentro de grandes volúmenes de datos.

#### Principios y Funcionamiento de TRACLUS

El funcionamiento de TRACLUS se basa en dos etapas principales: la segmentación de trayectorias y la agrupación de segmentos de trayectorias.

Ambas etapas están diseñadas para abordar la naturaleza secuencial y direccional de las trayectorias, empleando un enfoque basado en densidad que permite una identificación precisa de subtrayectorias similares.

1. **Segmentación de Trayectorias:** La primera etapa de TRACLUS es dividir cada trayectoria en segmentos de línea más cortos en función de cambios direccionales o puntos de inflexión. Estos puntos característicos dividen la trayectoria en subtrayectorias que pueden ser más fácilmente comparables. Este paso es crucial porque permite detectar patrones comunes en segmentos específicos, en lugar de requerir una coincidencia exacta en toda la trayectoria.
2. **Agrupación Basada en Densidad de Segmentos:** En lugar de agrupar puntos aislados, TRACLUS agrupa segmentos que se encuentran en regiones densas del espacio de trayectoria mediante una adaptación del algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*). Esta agrupación basada en densidad identifica áreas de alta concentración de segmentos similares que constituyen patrones de movimiento comunes. Los clusters se forman en áreas de alta densidad de segmentos, separadas por regiones de baja densidad, permitiendo agrupar segmentos que comparten características similares, incluso si otras partes de la trayectoria son diferentes.

## Métrica de Similitud en TRACLUS

TRACLUS emplea una métrica de distancia específica para medir la similitud entre segmentos individuales en lugar de comparar la trayectoria completa. Esto contrasta con métricas como DTW (*Dynamic Time Warping*) o LCSS (*Longest Common Subsequence*), que calculan la similitud entre series temporales completas. En cambio, TRACLUS se enfoca en la dirección y la longitud de cada segmento, permitiendo detectar subtrayectorias similares en conjuntos de datos con trayectorias complejas.

## Ventajas de TRACLUS en el Análisis de Trayectorias

El algoritmo TRACLUS ofrece varias ventajas que lo hacen adecuado para el análisis de datos de trayectoria:

- **Identificación de Subtrayectorias Similares:** TRACLUS no se limita a identificar patrones en trayectorias completas, sino que permite detectar similitudes en segmentos específicos. Esta capacidad es

esencial en contextos donde solo algunas secciones de las trayectorias son comparables, mientras que otras presentan variaciones.

- **Adaptación a Escalas y Densidades Variables:** Gracias a su enfoque basado en densidad, TRACLUS es menos sensible a las variaciones en la escala y la densidad de los datos, lo que facilita su aplicación en contextos heterogéneos, como en estudios de tráfico o movimientos de fauna en diferentes ecosistemas.
- **Selección Automática de Parámetros:** Mediante el uso de heurísticas para definir automáticamente los valores de parámetros clave (como el radio de vecindad  $\epsilon$  y el número mínimo de puntos vecinos), TRACLUS reduce la necesidad de ajustes manuales, lo cual incrementa su precisión y simplifica su aplicación en distintos conjuntos de datos.

### Aplicaciones de TRACLUS en la Investigación

TRACLUS se puede aplicar en múltiples campos de investigación, donde el análisis de patrones de movimiento es fundamental:

- **Biología y Ecología:** Para analizar trayectorias de animales, identificando patrones de comportamiento, rutas migratorias o territorios de caza.
- **Meteorología:** Para el estudio de trayectorias de fenómenos climáticos como huracanes, permitiendo identificar patrones comunes en ciertos eventos.
- **Gestión del Tráfico y Transporte:** En el análisis de rutas vehiculares, detectando patrones de congestión, flujo de tráfico y rutas populares.

## 3.2. Clustering

El **clustering** o agrupamiento es una técnica de aprendizaje no supervisado utilizada para organizar datos en grupos o "clusters" basados en características similares. Cada *cluster* está compuesto por elementos más similares entre sí que a elementos de otros *clusters*. Esta técnica es esencial en análisis exploratorio, permitiendo descubrir estructuras subyacentes en grandes volúmenes de datos y encontrar patrones, sin necesidad de tener etiquetas o categorías predefinidas.

En el contexto de análisis de datos, el clustering se aplica en múltiples áreas como la segmentación de clientes, detección de patrones de comportamiento, agrupación de imágenes y análisis de redes sociales. Entre los métodos de clustering más utilizados en la práctica se incluyen algoritmos basados en densidad y en conectividad, que proporcionan flexibilidad y adaptabilidad para manejar datos complejos y de alta dimensionalidad.

## Algoritmos de Clustering en scikit-learn

A continuación, se presenta una descripción de los algoritmos de clustering más relevantes y ampliamente utilizados en la biblioteca **scikit-learn** de Python:

### 1. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN es un algoritmo basado en densidad que agrupa puntos que están en áreas de alta densidad y considera como ruido aquellos que se encuentran en áreas de baja densidad. Los clusters se forman alrededor de puntos densamente conectados y son identificados por dos parámetros: el radio de vecindad ( $\epsilon$ ) y el número mínimo de puntos necesarios ( $minPts$ ) para formar un cluster. DBSCAN es especialmente útil para datos con formas irregulares y ruido, ya que ignora puntos aislados que no pertenecen a ninguna agrupación significativa.

### 2. OPTICS (Ordering Points To Identify the Clustering Structure)

OPTICS es una extensión de DBSCAN que aborda el problema de la sensibilidad a la elección de  $\epsilon$ . En lugar de identificar clusters individuales directamente, OPTICS produce una ordenación de los puntos que muestra su estructura de densidad subyacente. Esto permite descubrir clusters a múltiples escalas y niveles de densidad, haciendo posible una mayor flexibilidad en la agrupación de datos.

### 3. HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise)

HDBSCAN es una variante jerárquica de DBSCAN que forma clusters de densidad utilizando una estructura jerárquica en lugar de depender de un valor fijo de  $\epsilon$ . A diferencia de DBSCAN y OPTICS, HDBSCAN construye una jerarquía de clusters que permite identificar agrupaciones

en diferentes niveles de densidad sin requerir parámetros estrictos. Este algoritmo es particularmente útil cuando la densidad de los clusters varía significativamente.

#### 4. Spectral Clustering

Spectral Clustering es un algoritmo de agrupación basado en teoría de grafos y técnicas de álgebra lineal. Utiliza los valores propios de una matriz de similitud de los datos para realizar la agrupación. Este enfoque es particularmente adecuado para datos que presentan estructuras de clusters no lineales o formas complejas. Spectral Clustering convierte el problema de agrupación en uno de corte de grafos, dividiendo el conjunto de datos en  $k$  clusters minimizando la similitud entre los clusters.

#### 5. Agglomerative Clustering

El **Agglomerative Clustering** es una técnica jerárquica de clustering donde cada punto comienza como su propio cluster, y los clusters se fusionan iterativamente en función de una métrica de distancia (como la distancia euclídea, de Manhattan, o de enlace promedio) hasta que se alcanza el número deseado de clusters o se completa la jerarquía. Este método es particularmente útil cuando se requiere una representación visual de los clusters en forma de dendrograma.

### Comparación de los Algoritmos de Clustering

Los algoritmos de clustering mencionados se diferencian principalmente en su enfoque de agrupación (por densidad, jerárquico o basado en similitud), su sensibilidad a la elección de parámetros y su capacidad para manejar clusters de diferentes formas y densidades. A continuación, se presenta una tabla comparativa de los algoritmos:

Algoritmo	Tipo de Clustering	Ventaja Principal
DBSCAN	Densidad	Maneja ruido y clusters de formas arbitrarias
OPTICS	Densidad	Detecta clusters a diferentes niveles de densidad
HDBSCAN	Jerárquico basado en densidad	Sin parámetros estrictos
Spectral Clustering	Basado en grafos	Captura estructuras complejas
Agglomerative Clustering	Jerárquico	Ofrece dendrograma jerárquico



---

## 4. Técnicas y herramientas

---

En esta sección se presentan las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Se han considerado diferentes alternativas de metodologías y herramientas, y se ofrece un resumen de los aspectos más destacados de cada opción, junto con una justificación de las elecciones realizadas.

### 4.1. GitHub

GitHub es una plataforma de desarrollo colaborativo basada en la web que utiliza el sistema de control de versiones Git. Es ampliamente utilizado en la comunidad de desarrollo de software por varias razones:

- **Control de versiones:** GitHub facilita el seguimiento de cambios en el código, lo que permite a los desarrolladores revertir a versiones anteriores si es necesario. Esto es esencial para la gestión de errores y la mejora continua del software.
- **Integración continua:** GitHub se puede integrar con diversas herramientas de automatización, como GitHub Actions, que facilitan la construcción, prueba y despliegue automático del código, mejorando la eficiencia del flujo de trabajo.
- **Documentación y seguimiento de issues:** Proporciona herramientas para documentar el código y gestionar tareas o problemas a través de un sistema de “issues”, lo que facilita la organización y planificación del desarrollo.

## 4.2. Texmaker

Texmaker es un editor de texto multiplataforma para la creación de documentos en L<sup>A</sup>T<sub>E</sub>X. Es una herramienta esencial para la redacción académica y técnica, y ha sido fundamental en la redacción de este trabajo por las siguientes razones:

- **Interfaz amigable:** Proporciona un entorno intuitivo y fácil de usar que facilita la edición de documentos en L<sup>A</sup>T<sub>E</sub>X, incluso para aquellos que son nuevos en el sistema.
- **Compilación rápida:** Permite compilar documentos L<sup>A</sup>T<sub>E</sub>X rápidamente con un solo clic, lo que agiliza el proceso de revisión y mejora la productividad.
- **Herramientas integradas:** Incluye herramientas para la gestión de bibliografías, la inserción de gráficos y tablas, así como un visor PDF integrado que facilita la revisión del documento final.
- **Plantillas y ejemplos:** Ofrece diversas plantillas y ejemplos que ayudan a los usuarios a comenzar rápidamente con sus documentos, promoviendo buenas prácticas en la redacción científica.

## 4.3. CSS

Cascading Style Sheets (CSS) es un lenguaje utilizado para describir la presentación de documentos HTML y XML. En este proyecto, CSS se ha utilizado para mejorar la estética y la usabilidad de la interfaz de la aplicación desarrollada en Python con Dash. Las razones para su elección incluyen:

- **Separación de contenido y estilo:** CSS permite mantener el contenido HTML separado de su presentación, lo que facilita el mantenimiento del código y la implementación de cambios en el diseño sin afectar el contenido.
- **Responsividad:** Facilita el diseño responsivo, asegurando que la aplicación se vea bien en diferentes dispositivos y tamaños de pantalla. Esto es crucial para mejorar la accesibilidad y la experiencia del usuario.

- **Personalización:** Proporciona flexibilidad para personalizar la apariencia de la aplicación de manera sencilla, permitiendo la creación de un diseño atractivo y funcional que se alinee con los objetivos del proyecto.
- **Compatibilidad:** CSS es compatible con todos los navegadores modernos, lo que asegura que el diseño se mantenga consistente en diferentes plataformas.

## 4.4. Python

Python es un lenguaje de programación versátil y fácil de aprender, ampliamente utilizado en el desarrollo de aplicaciones web, análisis de datos y machine learning. En este proyecto, se han utilizado varias bibliotecas de Python que han enriquecido el desarrollo y la funcionalidad de la aplicación:

- **scikit-learn:** Una biblioteca para machine learning que ofrece herramientas eficientes para el análisis predictivo y la implementación de algoritmos de clustering, como DBSCAN y OPTICS. Su diseño optimizado permite realizar análisis complejos de manera rápida y sencilla.
- **Pandas:** Una biblioteca de análisis de datos que proporciona estructuras de datos y herramientas de manipulación para trabajar con datos tabulares. Permite la limpieza y transformación de datos, lo que es esencial para preparar los datos antes del análisis.
- **GeoPandas:** Extiende las capacidades de Pandas para trabajar con datos geoespaciales, permitiendo realizar análisis y visualizaciones de datos geográficos de forma eficiente. Esto es crucial para proyectos que requieren análisis de datos basados en ubicación.
- **Matplotlib:** Una biblioteca para crear visualizaciones estáticas, animadas e interactivas en Python. Es esencial para la representación gráfica de los resultados del análisis, facilitando la comunicación de hallazgos a través de gráficos claros y efectivos.
- **NumPy:** Una biblioteca fundamental para realizar cálculos numéricos en Python, que proporciona soporte para matrices y funciones matemáticas. NumPy es la base sobre la cual se construyen muchas otras bibliotecas de ciencia de datos y machine learning.

- **Shapely:** Utilizada para manipular y analizar geometrías en Python, es crucial en la gestión de datos geoespaciales, permitiendo realizar operaciones geométricas como intersecciones y uniones de formas.
- **JSON:** Una biblioteca que permite trabajar con datos en formato JSON, facilitando la interacción con APIs y el manejo de configuraciones. Esto es importante para integrar servicios externos en la aplicación.
- **Zipfile:** Utilizada para crear y leer archivos ZIP, facilitando la gestión de datos comprimidos y mejorando la eficiencia en la transferencia de datos.
- **Contextily:** Permite añadir mapas de fondo a las visualizaciones geográficas, mejorando la contextualización de los datos y ayudando a los usuarios a interpretar la información espacial.
- **io:** Proporciona funciones para manejar flujos de entrada y salida, útil en la manipulación de datos en memoria, permitiendo una gestión eficiente de archivos y datos temporales.
- **pyproj:** Una biblioteca para realizar transformaciones de coordenadas, crucial para el trabajo con datos geográficos y la interoperabilidad entre diferentes sistemas de referencia espacial.
- **Time y threading:** Utilizadas para gestionar la temporización y la ejecución de múltiples hilos de ejecución en la aplicación, lo que mejora la eficiencia y la capacidad de respuesta de la aplicación en tareas concurrentes.
- **Base64:** Facilita la codificación y decodificación de datos en formato Base64, útil para la transferencia de datos binarios, como imágenes o archivos, en formatos que requieren representación textual.
- **Dash:** Dash es un marco de trabajo desarrollado por Plotly que permite la creación de aplicaciones web analíticas e interactivas utilizando Python. Es especialmente popular en la comunidad de ciencia de datos y visualización debido a sus características y beneficios:
  - **Interactividad:** Dash permite crear aplicaciones que responden a las interacciones del usuario, como clics, desplazamientos y entradas de datos. Esto es crucial para el análisis de datos en tiempo real y la visualización interactiva.

- **Integración con Plotly:** Las visualizaciones de Dash se basan en la biblioteca Plotly, que permite crear gráficos complejos y visualizaciones de alta calidad con facilidad. Esto enriquece la presentación de datos y facilita la comunicación de resultados.
- **Composición de componentes:** Dash permite combinar diferentes componentes (gráficos, tablas, controles de entrada) en una sola interfaz, lo que facilita la creación de aplicaciones integrales que ofrecen una experiencia de usuario fluida.
- **Despliegue sencillo:** Las aplicaciones construidas con Dash se pueden desplegar fácilmente en servidores web, lo que permite compartir los resultados del análisis con un público más amplio sin requerir instalación adicional por parte del usuario final.
- **Flexibilidad:** Al estar basado en Python, los desarrolladores pueden aprovechar la amplia gama de bibliotecas disponibles para manipular datos, realizar análisis y crear visualizaciones personalizadas, lo que proporciona gran flexibilidad en el desarrollo de aplicaciones.

Estas herramientas y bibliotecas han sido elegidas por su capacidad para facilitar el desarrollo, mejorar la eficiencia del trabajo y proporcionar funcionalidades que son fundamentales para el éxito del proyecto. Cada una de ellas contribuye a un enfoque integral que permite abordar las necesidades del análisis de datos y la creación de aplicaciones web interactivas.



---

## 5. Aspectos relevantes del desarrollo del proyecto

---

### 5.1. Herramientas de control de tareas: Kanban y Scrum

Para llevar un control efectivo de las tareas y asegurar la evolución continua del proyecto, se han utilizado metodologías ágiles como **Kanban** y **Scrum**.

- **Kanban:** Se utilizó Kanban debido a su flexibilidad y enfoque visual. Kanban nos permitió gestionar el flujo de trabajo mediante tarjetas en un tablero visual, donde las tareas se movían de una columna a otra (de "Por hacer" .<sup>a</sup> "En progreso" .<sup>b</sup> "Hecho"). Esto facilitó una vista clara de las tareas pendientes, así como una priorización y reestructuración rápida de las mismas según los requerimientos que surgían.
- **Scrum:** Se implementó Scrum en ciclos de trabajo denominados *sprints*, con duraciones de dos semanas. Cada *sprint* comenzaba con una reunión en la que se revisaban en retrospectiva las tareas realizadas el último *sprint* y se planificaban los objetivos y metas para la siguiente reunión. Este ciclo permitió adaptar las prioridades en función de los avances realizados y los desafíos encontrados, proporcionando un proceso estructurado que se ajustó bien a las necesidades de fases más avanzadas del desarrollo.

El uso combinado de ambas metodologías facilitó tanto la flexibilidad como la estructura necesarias en diferentes fases del proyecto, contribuyendo a un flujo de trabajo eficiente y orientado a resultados.

## 5.2. Formación y aprendizaje necesario

El desarrollo del proyecto requirió adquirir nuevos conocimientos y profundizar en diversas tecnologías y técnicas de análisis. A continuación, se describe la formación llevada a cabo en las herramientas y algoritmos más relevantes.

### Estudio del algoritmo TRACLUS

El enfoque principal de este proyecto ha sido estudiar en profundidad el algoritmo **TRACLUS**, destinando una gran parte de las horas de investigación a comprender su funcionamiento y potencial para el análisis de trayectorias. Este algoritmo se basa en la segmentación y agrupación de trayectorias, permitiendo identificar sub-trayectorias comunes dentro de un conjunto de datos. Esto lo convierte en una herramienta valiosa para descubrir patrones significativos en datos de trayectorias. La investigación incluyó la revisión de artículos académicos y la exploración de cómo ajustar los parámetros de TRACLUS para maximizar su efectividad en el contexto específico del proyecto.

Adicionalmente, se evaluó el uso de una implementación de TRACLUS disponible en una biblioteca externa, explorando su viabilidad y características. Aunque esta versión fue útil para los primeros experimentos, se consideraron también posibles variantes y adaptaciones del algoritmo, con el fin de entender mejor el alcance y la adaptabilidad de TRACLUS.

### Propuesta de variantes de TRACLUS

Durante el desarrollo del proyecto, se analizaron varias propuestas de variantes del algoritmo TRACLUS que han surgido en estudios recientes. Aunque finalmente no se utilizaron, la investigación detallada de estos derivados fue enriquecedora y ayudó a contrastar enfoques para una implementación eficiente. Las variantes de TRACLUS estudiadas fueron:

- **GTraclass:** Una variante diseñada para ejecutarse en unidades de procesamiento gráfico (GPU), optimizando la eficiencia del algoritmo al aprovechar la capacidad de procesamiento paralelo de las GPUs [?].

- **ST-TRACLUS:** Esta versión incorpora una dimensión temporal además de la espacial, permitiendo realizar agrupamientos espacio-temporales de trayectorias, lo cual mejora la calidad del análisis para datos donde el tiempo es una variable relevante [?].
- **ND-TRACLUS:** Una extensión de TRACLUS que permite realizar el clustering en espacios de n dimensiones, expandiendo su aplicabilidad a trayectorias de mayor dimensionalidad [?].
- **Neighborhood-Based Trajectory Clustering:** Una alternativa basada en densidad local de vecindad en lugar de densidad global. Este enfoque busca mantener la eficiencia de TRACLUS mientras reduce la necesidad de múltiples parámetros de entrada.
- **Adaptive Trajectory Clustering based on Grid and Density (ATCGD):** Un método que introduce una cuadrícula y criterios de densidad para el análisis de patrones móviles, buscando reducir la complejidad computacional y la carga de trabajo en la calibración de parámetros, especialmente en aplicaciones a gran escala como la de trayectorias de vehículos en sistemas de transporte inteligente [?].

### Técnicas de clustering complementarias

Paralelamente a la investigación de las variantes de TRACLUS, se estudiaron técnicas adicionales de clustering, dado que el algoritmo requiere la clusterización de segmentos para su evaluación final. Con el objetivo de evaluar la viabilidad y robustez de TRACLUS, se analizaron diversas técnicas de clustering, y tras descartar varias por inviabilidad, se optó finalmente por implementar las siguientes metodologías de `scikit-learn`:

- **DBSCAN:** Algoritmo basado en densidad que identifica clusters de alta densidad separados por regiones de menor densidad, adecuado para datos espaciales y resistente al ruido.
- **OPTICS:** Similar a DBSCAN, pero permite una sensibilidad ajustable a la densidad, lo que lo hace más flexible para analizar datos con densidades variadas.
- **HDBSCAN:** Variante jerárquica de DBSCAN que ajusta automáticamente los parámetros de densidad, simplificando su aplicación en datos de densidad variable.

- **Spectral Clustering:** Algoritmo basado en el análisis de valores propios que es particularmente útil en datos no lineales o con clusters de formas complejas.
- **Agglomerative Clustering:** Método jerárquico ascendente que agrupa iterativamente elementos basándose en la proximidad, resultando útil en análisis donde la estructura jerárquica es relevante.

Cada técnica fue investigada en términos de sus características, sensibilidad a la densidad, capacidad de manejar ruido y aplicabilidad a datos espaciales. La exploración de estas técnicas permitió seleccionar aquellas que mejor se adaptaran a las necesidades específicas del proyecto y que complementaran el uso de TRACLUS en el análisis de trayectorias.

## Tiempos de Ejecución

El que podríamos considerar como el segundo punto más relevante, y sin duda uno de los aspectos críticos del proyecto, han sido los tiempos de ejecución. Debido a las grandes cantidades de datos a procesar, cada cálculo y visualización requerían una considerable cantidad de tiempo, por lo que la optimización de tiempos se convirtió en una prioridad clave para asegurar la viabilidad del proyecto.

## Visualización y Dibujado de Mapas

Uno de los mayores desafíos de rendimiento fue la visualización de mapas, especialmente al intentar realizar representaciones detalladas y visualmente atractivas. Al analizar distintas bibliotecas de Python para dibujar y visualizar mapas, se probaron y compararon múltiples opciones combinadas entre si, como `pandas`, `geopandas`, `folium`, `folium.plugins`, `matplotlib.pyplot`, `matplotlib.colors`, `contextily`, `pyproj`, `seaborn`, `pydeck`, `shapely.geometry`, `sklearn.preprocessing` y `scipy.stats`.

Tras diversas pruebas de rendimiento y calidad visual, se optó por utilizar principalmente `contextily` y `matplotlib.pyplot` para la visualización de mapas, ya que estas bibliotecas ofrecieron la mejor relación entre rendimiento y calidad gráfica. Para la visualización de *heatmaps*, se incluyeron `numpy` como las opciones más eficientes para generar mapas de calor. Sin embargo, para lograr un equilibrio óptimo entre rendimiento y visualización, se sacrificaron ciertas funcionalidades como el zoom en mapas interactivos y detalles visuales avanzados.

## Optimización del Algoritmo TRACLUS

El algoritmo TRACLUS, por su complejidad, resultó ser las partes más lentas del sistema. Para abordar este desafío, se llevó a cabo un exhaustivo estudio de rendimiento, en el cual se identificaron las partes del código que consumían más recursos y tiempo de ejecución.

Como resultado de este análisis, se investigó la posibilidad de aplicar técnicas de paralelización y `threading` para mejorar la eficiencia del procesamiento. Sin embargo, la naturaleza del algoritmo, en combinación con el volumen de datos, limitó la efectividad de estas técnicas en algunas secciones críticas. A pesar de ello, ciertos aspectos del procesamiento pudieron ser paralelizados para aprovechar mejor los recursos de hardware, logrando una mejora en el tiempo total de ejecución.

## Librerías

- **Dash:** *Dash* fue seleccionada como la biblioteca principal para desarrollar la interfaz de usuario debido a su capacidad para crear aplicaciones web interactivas y visualizaciones de datos de manera rápida y eficiente. La necesidad de una plataforma que permitiera integrar gráficos dinámicos y controles de usuario en una sola interfaz fue clave en esta decisión. Además, *Dash* está diseñado para trabajar con *Python* y se adapta bien a flujos de trabajo de análisis de datos, lo cual facilitó la integración de otras librerías analíticas y permitió crear una experiencia interactiva para los usuarios sin requerir un extenso conocimiento de desarrollo web. La posibilidad de manejar datos en tiempo real y adaptar la visualización a los cambios en los datos fue esencial para cumplir con los objetivos del proyecto de forma ágil y eficiente.
- **Pandas:** *Pandas* fue seleccionada para el procesamiento de datos debido a su capacidad para manejar grandes conjuntos de datos de forma estructurada y eficiente. A lo largo del proyecto, la necesidad de limpiar, transformar y analizar datos en estructuras flexibles y accesibles fue un desafío recurrente. *Pandas* resultó ser la elección óptima porque permite trabajar con *DataFrames*, lo cual facilitó la manipulación de datos, la agregación y el filtrado de información en diferentes fases del proyecto. Esta elección también permitió un preprocesamiento ágil antes de la visualización en *Dash*, logrando así una presentación de los datos más clara y coherente en la interfaz final.

- **Otras Librerías:** En conjunto con Dash y Pandas, se decidió utilizar una serie de librerías complementarias que cubrieron necesidades específicas en distintas áreas del proyecto:
  - **GeoPandas y Contextily:** Estas librerías fueron indispensables para la visualización de datos geoespaciales. La necesidad de mostrar datos de ubicación de manera precisa en mapas y de superponer capas geográficas con datos de contexto geográfico impulsó la elección de GeoPandas y Contextily. GeoPandas permitió trabajar con datos espaciales en un formato similar al de Pandas, mientras que Contextily proporcionó mapas base que complementaron la visualización de los datos geográficos en el proyecto.
  - **Transformación de Coordenadas (pyproj):** La transformación y manejo de coordenadas geográficas fue una necesidad constante, especialmente cuando se combinaron datos de diversas fuentes con diferentes sistemas de referencia. La elección de *pyproj* fue motivada por su capacidad de manejar transformaciones complejas entre sistemas de coordenadas, asegurando así la precisión espacial en las visualizaciones y análisis de datos geográficos.
  - **Manipulación de Datos JSON y Archivos ZIP (json y zipfile):** Dado el gran volumen de datos y la necesidad de mantener la eficiencia en la carga y el almacenamiento, se emplearon *json* y *zipfile*. La elección de estas librerías fue motivada por su capacidad de facilitar la carga de datos estructurados en JSON y la gestión de archivos ZIP.
  - **Matplotlib y Numpy:** *Matplotlib* y *Numpy* fueron seleccionadas para complementar la visualización de datos y los cálculos matemáticos necesarios en el proyecto. *Numpy* fue esencial para realizar operaciones numéricas avanzadas y el manejo eficiente de arrays de datos. *Matplotlib*, por su parte, aportó flexibilidad en la generación de gráficos y permitió personalizar visualizaciones estáticas..

### 5.3. Investigación y experimentación

Durante el desarrollo de este proyecto se realizaron numerosas pruebas, orientadas principalmente a la investigación y experimentación con distintas herramientas y algoritmos. Estas pruebas fueron fundamentales para afianzar

conceptos, validar el funcionamiento del código y lograr avances significativos en términos de rendimiento y precisión en los resultados.

## TRACLUS por partes

Desde el inicio del proyecto, se tomó como base para el desarrollo del algoritmo TRACLUS una implementación existente llamada *TRACLUS\_library* [?], que fue crucial para establecer un punto de partida en la comprensión y adaptación de este algoritmo. *TRACLUS\_library* proporciona una implementación básica del algoritmo TRACLUS.

En una primera prueba, se intentó ejecutar el programa con un conjunto de datos seleccionado, lo cual presentó diversas complicaciones, principalmente debido a la falta de familiaridad con los formatos de datos requeridos y la preparación necesaria para que el programa pudiera leerlos y procesarlos correctamente. Una vez resueltos estos problemas iniciales de formato, se procedió a un análisis profundo del código de la librería, dividiéndolo en tres componentes principales: cálculo de distancia entre trayectorias, segmentación o partición de trayectorias, y vectorización de segmentos. Esta separación permitió estudiar cada parte de forma aislada.

### Distancia

La primera sección del código, centrada en el cálculo de distancias, resultó ser la más extensa y compleja, tanto por el volumen de código como por la lógica de cálculo empleada. Para medir la similitud entre trayectorias, el programa utiliza tres tipos de distancia: angular, perpendicular y paralela. Estas distancias se combinan para definir una medida global entre cada par de trayectorias, generando una matriz de distancias.

El cálculo de distancias fue un área de gran interés en términos de optimización, dado que la complejidad del proceso aumenta exponencialmente a medida que el número de trayectorias crece, debido a la necesidad de comparar cada trayectoria con todas las demás.

### Particiones

La segunda sección del código estaba orientada a la segmentación de las trayectorias en particiones manejables. Este paso fue crucial, ya que las particiones generadas en esta etapa se utilizaron posteriormente para la vectorización, y por ende, para el proceso de agrupamiento en el algoritmo TRACLUS. Las pruebas en esta sección se centraron en verificar que el algoritmo segmentara correctamente las trayectorias y en ajustar los parámetros

para mejorar la precisión de las particiones generadas. La segmentación fue una fase intermedia y relativamente estable, ya que dependía de funciones bien definidas dentro de la librería base y no requería grandes cambios en el código.

### Vectorización

En esta fase, los segmentos de trayectorias se transforman en vectores utilizando la biblioteca `scikit-learn`, específicamente con el algoritmo `OPTICS`. La vectorización es clave para convertir las trayectorias segmentadas en una estructura que pueda ser procesada en el espacio de *clustering*. Durante el desarrollo, esta fue una etapa de experimentación en la que se evaluaron diferentes configuraciones de *clustering* para observar cómo afectaban los resultados y qué parámetros resultaban óptimos para el proyecto.

### Representación de Trayectorias

Por último, la representación de trayectorias es el paso en el que se visualizan los *clusters* obtenidos y se presentan los patrones de movimiento descubiertos a través de TRACLUS. Esta fase es crucial, ya que permite observar de manera clara y organizada cómo se agrupan las trayectorias en función de similitudes en sus segmentos.

### Geolife Trajectories

Durante todo el proyecto se utilizó principalmente el conjunto de datos \*\*Trayectorias Taxis\*\* [?]. Este conjunto de datos fue ideal para el análisis, ya que contaba con los datos necesarios y estaba estructurado de manera adecuada para ejecutar el algoritmo TRACLUS de forma eficiente.

Sin embargo, también se exploraron otras fuentes de datos, como el conjunto \*\*Geolife Trajectories\*\* [?].

Este estudio proporciona un interesante conjunto de datos sobre los movimientos de múltiples individuos. Los datos abarcan un período de más de tres años, durante los cuales 182 usuarios contribuyeron al proyecto con un total de 17,621 trayectorias GPS. Estas trayectorias representan más de 1.2 millones de kilómetros registrados y aproximadamente 48,000 horas de actividad.

Cada trayectoria se compone de puntos con marcas de tiempo que contienen información clave como latitud, longitud y altitud. Además, la alta precisión y granularidad de estos datos los convierten en un recurso

valioso para el desarrollo y la prueba de algoritmos como TRACLUS, que requiere un análisis exhaustivo de trayectorias.

A pesar de su potencial, trabajar con \*\*Geolife Trajectories\*\* presentaba desafíos significativos. Los datos estaban organizados por usuarios individuales y cada usuario tenía sus trayectorias divididas en múltiples archivos .plt. Además, las trayectorias estaban dispersas en múltiples ubicaciones, lo que a menudo generaba inconsistencias y complicaciones para cargar los datos en masa.

Debido a estas diferencias en la estructura y el formato, el tratamiento de los datos requería un enfoque completamente distinto al utilizado con el conjunto de Trayectorias Taxis. Si bien estos datos eran válidos para ser analizados con el algoritmo TRACLUS, integrarlos en el flujo principal del proyecto habría implicado un costo significativo en términos de tiempo y esfuerzo. Por tanto, la manera más adecuada de estudiar estos datos habría sido tratarlos de manera independiente, desarrollando una línea de trabajo separada para su carga, preprocesamiento y visualización.

## Página Web

El primer paso para comenzar con el diseño fue crear un prototipo muy básico de las funciones que la página web debía tener.

La primera de las páginas debía contar con una representación de los datos sin procesar. En esta pantalla se debía poder visualizar un mapa con las trayectorias y otro con el mapa de calor Tensor. Además, se consideró incluir funcionalidades para manipular los datos, como la selección de longitudes y latitudes mínimas y máximas, o la posibilidad de realizar zoom en el mapa. También se propuso añadir un limitador de datos a utilizar, aunque esta función podría suponer mucho trabajo sin aportar un beneficio significativo frente a simplemente predeterminar el número de datos en el código. Por último, se planteó la posibilidad de incluir una tabla que mostrara los datos en bruto, sin procesar.

Para acceder a las siguientes páginas de visualización de datos, se propuso implementar un menú desplegable accesible desde las tres barras en la cabecera o mediante enlaces visibles en la cabecera o el pie de página.

La segunda página estaría dedicada a la comparativa de clusters y su representación en un mapa. Esta consistiría en cuatro mapas, cuyo contenido se seleccionaría mediante un menú de botones colocados en el centro de la pantalla. Cada usuario podría seleccionar dos mapas, uno por cada lado, para

visualizarlos en sus respectivas áreas. Aunque se evaluaron otras opciones de selección, como interruptores o menús, finalmente se optó por un diseño centralizado. También se propuso añadir una opción para seleccionar tamaños de datos, aunque esto podría ralentizar la visualización si no se optaba por medidas predefinidas y prerenderizadas.

Por último, la tercera página estaría dedicada a la comparativa de los datos entre diferentes algoritmos. En esta página se podrían incluir tablas que permitieran analizar la relación entre clusters y trayectorias o, alternativamente, una tabla única que mostrara un listado de segmentos con sus respectivas trayectorias y los clusters a los que pertenecen según cada algoritmo.

Tras la elaboración de los prototipos, se llegó a la conclusión de que, además de la biblioteca Dash, era necesario utilizar código complementario, como **CSS Grid**, para posicionar los diferentes componentes en pantalla. Además, para evitar tiempos de carga excesivos, todas las imágenes y tablas deberían estar prerenderizadas, limitando las opciones del usuario a una lista de opciones predefinidas.

Una vez definidos los prototipos, comenzó el desarrollo, que se prolongó durante la mayor parte del proyecto. Inicialmente, se intentó implementar todo tal como se había planteado, pero esto no fue posible debido a diversas limitaciones. Por ejemplo, la página inicial se dividió en dos: una para la carga de datos y otra para la visualización de las trayectorias, mejorando así la claridad y la experiencia del usuario.

Las otras dos páginas se mantuvieron con diversos cambios en su diseño, adoptando un patrón de diseño y color unificado en toda la web. Más adelante, se añadieron nuevas funcionalidades y pantallas, y la barra de navegación se convirtió en el principal medio para moverse entre las distintas páginas, en lugar del menú desplegable ideado anteriormente. Además, se incorporó un botón para descargar los datos generados durante los experimentos.

Sumado a lo anterior, se eliminaron diferentes funcionalidades que se buscaban al principio, como el zoom en los mapas, y se añadieron dos nuevas pantallas: una nueva página inicial donde se puede seleccionar si crear un nuevo experimento o reutilizar uno existente, y una pantalla previa a la carga de datos donde el usuario selecciona los algoritmos de clustering a utilizar. Estos cambios significativos durante el desarrollo requirieron reescribir y reorganizar el código en múltiples ocasiones, adoptando finalmente un enfoque basado en el modelo vista-controlador para mejorar la estructura y mantenibilidad del proyecto.

## Optimización del algoritmo

Uno de los principales desafíos del proyecto fue el tiempo de procesamiento, debido a la gran cantidad de datos que debían manejarse. El tamaño y la complejidad de los datos afectaron tanto la carga inicial como la ejecución de algoritmos y la visualización, requiriendo una optimización constante para lograr resultados en tiempos razonables.

### Carga de Datos

El primer obstáculo en términos de tiempo fue la carga de datos en la aplicación web, ya que `Dash` tiene limitaciones al cargar grandes volúmenes de datos. El archivo inicial pesaba aproximadamente dos gigabytes, y ni siquiera herramientas como `Excel` podían manejar correctamente un CSV de este tamaño, lo que llevó a errores frecuentes al intentar reducir su tamaño sin comprometer la integridad de los datos.

Tras varios intentos de conversión y reducción, se logró un tamaño adecuado que permitió cargar los datos en la aplicación. Además, para estructurar y analizar los datos en trayectorias geográficas, fue necesario procesarlos con `GeoDataFrame` mediante la biblioteca `GeoPandas`. Gracias a esta biblioteca, el tiempo de carga y procesamiento fue moderado y permitió manipular los datos de forma eficiente para futuras visualizaciones y análisis.

Para la visualización, el tamaño del conjunto de datos impactó en la calidad gráfica y los tiempos de carga. Fue necesario optar por bibliotecas de visualización que equilibraran la calidad y el tiempo de renderizado, sacrificando algunas características visuales avanzadas en favor de un rendimiento adecuado. Finalmente, se optó por `contextily` y `matplotlib.pyplot` para la representación de mapas y `numpy` para los mapas de calor.

### Distancias

El proceso más complejo y exigente en términos de tiempo fue la ejecución del algoritmo TRACLUS. Durante su ejecución, es necesario calcular las distancias perpendicular, paralela y angular entre todas las trayectorias, lo que genera una complejidad algorítmica exponencial  $O(n^2)$ . Esto se vuelve aún más costoso cuando se desean evaluar varias opciones de clustering en el mismo conjunto de datos.

Antes de las optimizaciones, el programa tardaba aproximadamente dos minutos y cuarenta y cinco segundos en cargar cien filas de datos, y una hora y cinco minutos en cargar quinientas filas, lo cual era insostenible dado

que el archivo fuente de datos del proyecto contenía más de un millón de filas.

Para reducir estos tiempos, se realizaron múltiples pruebas y técnicas de optimización:

1. **Numpy para la matriz de distancia:** Inicialmente, se intentó cargar la matriz de distancia utilizando `numpy`, que es una biblioteca altamente optimizada para cálculos matemáticos. Sin embargo, los resultados generados no coincidían con los obtenidos mediante el cálculo original, lo que condujo a diferencias significativas en los resultados de clustering.
2. **Vectorización con Numpy:** En un segundo intento, se volvió a probar con `numpy`, aplicando un enfoque más completo de vectorización para las tres distancias. Nuevamente, aunque los tiempos de ejecución mejoraron, los resultados no fueron precisos, afectando la coherencia en los datos obtenidos.
3. **Threading con bucles:** Para el tercer intento, se reutilizaron las funciones originales de cálculo de distancias, pero paraleizando los tres bucles de cálculo de distancias mediante `threading`. Esta prueba mejoró los tiempos levemente, reduciendo la carga de cien filas a dos minutos y veintidós segundos, manteniendo los resultados consistentes, aunque la mejora fue insuficiente para los objetivos del proyecto.
4. **ThreadPoolExecutor y chunks:** En la cuarta prueba, se dividió la matriz de distancia en “chunks” para ser procesados en paralelo con `ThreadPoolExecutor`. Sin embargo, esta técnica solo generó una mejora marginal, con una reducción de aproximadamente seis segundos en la carga de cien filas, lo cual, aunque significativo para cantidades de datos pequeñas, resultó ineficiente para volúmenes mayores.
5. **Paralelización manual y threading:** Finalmente, se probó una combinación de `threading`, sin usar `numpy` para los cambios en las funciones de distancia, calculando las distancias en tres hilos independientes. Este enfoque proporcionó el mejor resultado, logrando una reducción de tiempo a la mitad: un minuto y cuatro segundos para cien filas, y treinta y un minutos y cuarenta y nueve segundos para quinientas filas. Sin embargo, aunque esta reducción era significativa, los tiempos seguían siendo elevados para la totalidad de los datos del proyecto.

Este proceso de optimización requirió numerosos ajustes y pruebas incrementales, en muchos casos con cambios mínimos y variaciones en el tamaño de los datos de prueba, lo que demandó una gran cantidad de horas de prueba y error sin obtener los resultados esperados. No se tiene un cálculo exacto, pero este apartado del proyecto consumió no solo decenas, sino cientos de horas dedicadas exclusivamente a la ejecución y análisis de pruebas.

En conclusión, aunque Python es un lenguaje versátil, sus limitaciones en paralelización y threading efectivo complicaron la optimización de cálculos intensivos en comparación con otros lenguajes como Java, lo que limitó el rendimiento alcanzable en este proyecto.

## Comparativa de algoritmos

Una vez que las pruebas del algoritmo se hicieron menos tediosas, se buscó comparar el funcionamiento y los resultados que podían proporcionar los diferentes algoritmos de clustering bajo condiciones relativamente equivalentes. Para ello, se utilizó la misma cantidad de datos: doscientas filas del archivo \*Trayectorias Taxis\* [?]. Respecto a los parámetros necesarios para la ejecución de cada algoritmo, se intentó mantenerlos lo más estándar posible. Por ejemplo, en los casos en que era necesario especificar el número de clusters, se utilizó una aproximación basada en el resultado obtenido por OPTICS, ya que este algoritmo no requiere definir dicho parámetro. El resto de los parámetros se dejaron en los valores predeterminados que proporciona la biblioteca `scikit-learn`.

Aunque se probaron más algoritmos, solo cinco llegaron a la etapa final del desarrollo de la página web. Algunos, como **Birch**, no lograron producir resultados satisfactorios, aunque se considera que, con una adaptación más específica de los datos, podrían haber funcionado correctamente.

A continuación, se describen los resultados obtenidos para cada uno de los algoritmos seleccionados:

1. **OPTICS**: Este fue el primer algoritmo probado de manera intensiva, ya que era el utilizado por defecto en la implementación base. De las doscientas filas procesadas, se generaron un total de 2161 segmentos, que fueron agrupados en 106 clusters. Sin embargo, no todos los segmentos se utilizaron para la creación de los clusters; un total de 1354 segmentos fueron catalogados como "basura", lo que corresponde al 62.66 % de los datos. A continuación, se muestran los resultados en tres imágenes diferentes: la representación de los clusters, un histograma con

la cantidad de trayectorias que forman cada cluster y la representación de trayectorias generadas por el algoritmo TRACLUS.

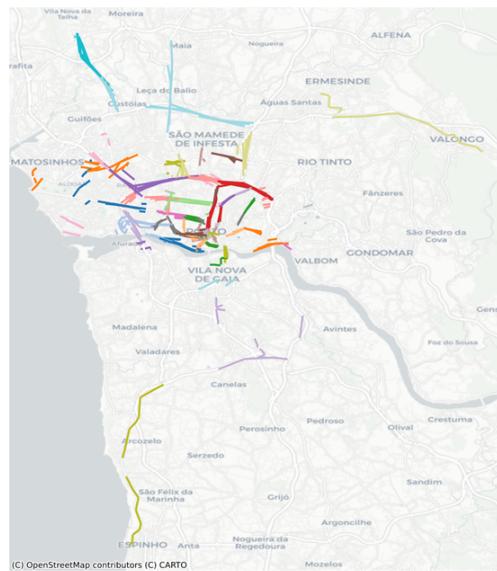


Figura 5.1: Representación de clusters.

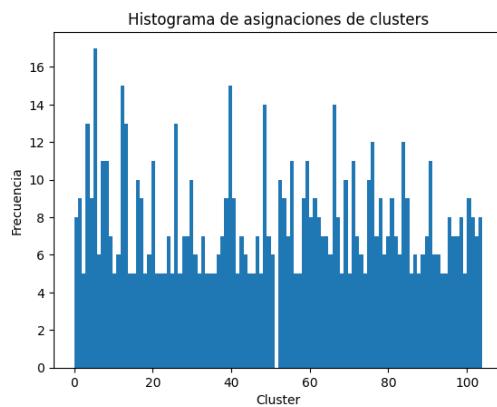


Figura 5.2: Segmentos por cada cluster.



Figura 5.3: Representación de trayectorias.

2. **DBSCAN:** Con un valor de `eps` de 0.1, los resultados de DBSCAN fueron significativamente diferentes a los de OPTICS. Aunque se generaron más segmentos (2654 en total), el número de clusters disminuyó a 37. Además, el porcentaje de datos clasificados como "basura" aumentó al 90.09 %, lo que equivale a 2391 segmentos descartados.



Figura 5.4: Representación de clusters.

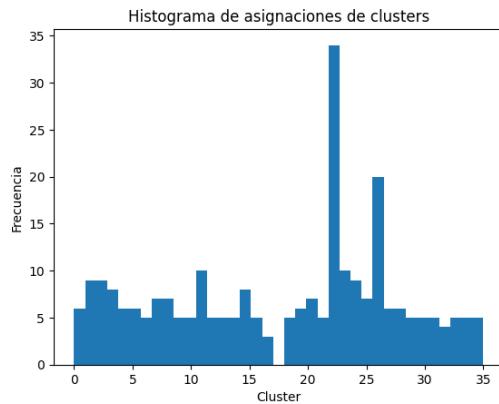


Figura 5.5: Segmentos por cada cluster.



Figura 5.6: Representación de trayectorias.

3. **HDBSCAN**: Este algoritmo no requirió ajustes en sus parámetros predeterminados de `scikit-learn`. Los resultados fueron similares a los de OPTICS en términos de segmentos (2161), aunque el número de clusters fue menor (96) y el porcentaje de segmentos descartados también disminuyó, alcanzando un 53.54 % (1157 segmentos).

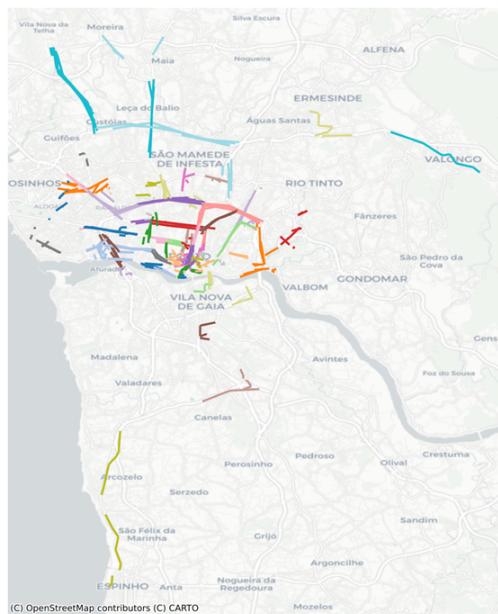


Figura 5.7: Representación de clusters.

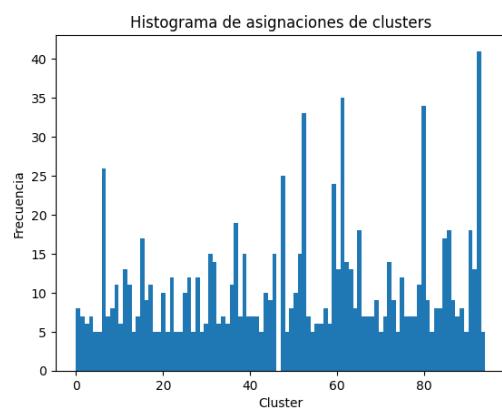


Figura 5.8: Segmentos por cada cluster.

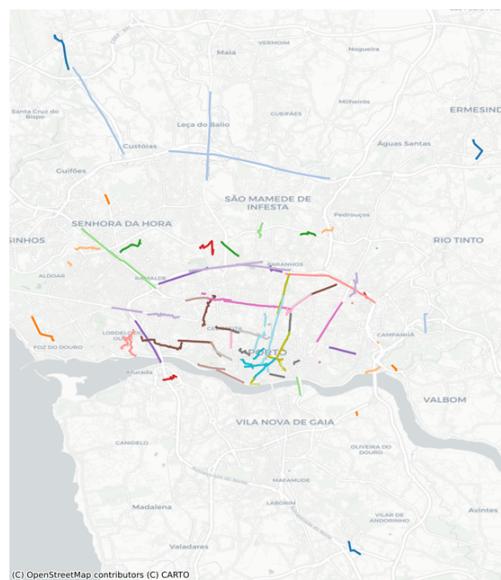


Figura 5.9: Representación de trayectorias.

**4. Agglomerative Clustering:** Este algoritmo requería definir previamente el número de clusters. En este caso, se utilizaron los 2654 segmentos generados, sin descartar ninguno, ya que no clasifica datos como "basura". Sin embargo, esta característica provoca que los clusters no se centren en las zonas más densas, lo que resulta en representaciones de trayectorias más erráticas.

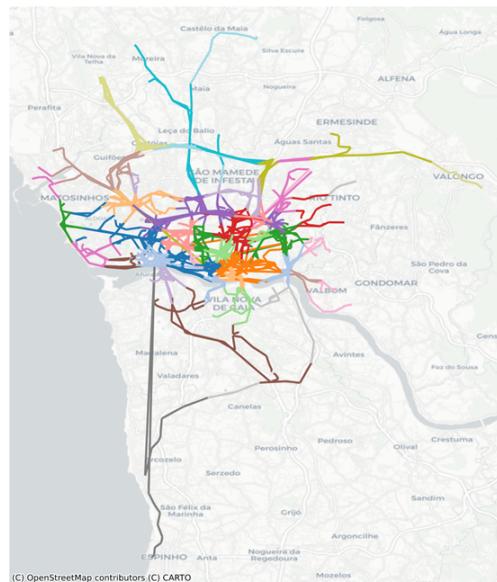
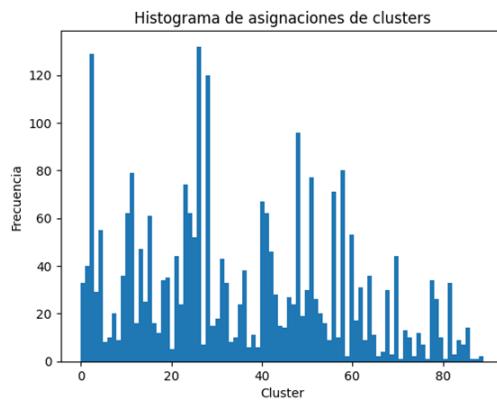


Figura 5.10: Representación de clusters.



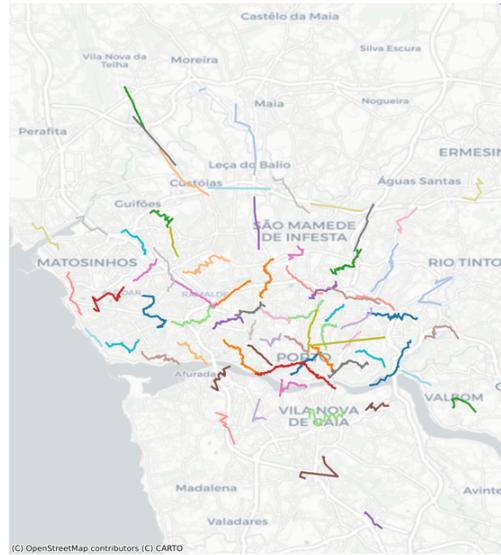


Figura 5.12: Representación de trayectorias.

5. **Spectral Clustering:** Al igual que el algoritmo anterior, no descarta datos. Aunque se generaron los mismos 2654 segmentos y clusters que en Agglomerative Clustering, los resultados finales fueron diferentes, con una distribución menos centralizada.

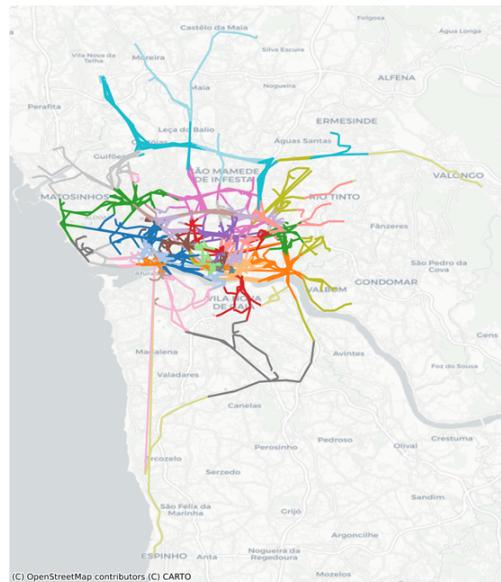


Figura 5.13: Representación de clusters.

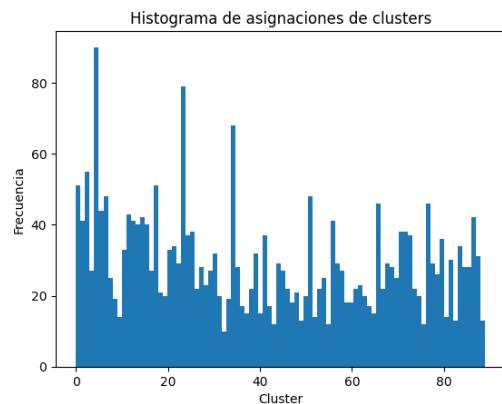


Figura 5.14: Segmentos por cada cluster.

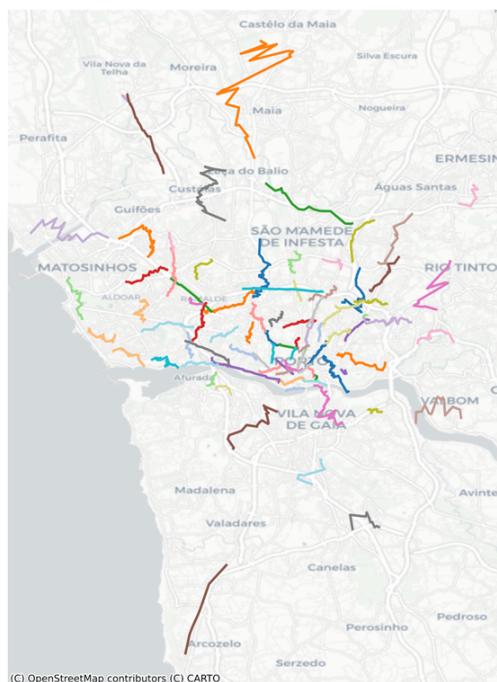


Figura 5.15: Representación de trayectorias.

## Optimización de la página web

Tras múltiples ejecuciones de la página web para comparar los resultados de los diferentes algoritmos de clustering aplicados al TRACLUS, se identificó un problema recurrente: el tiempo de ejecución elevado.

En la aplicación, se podían ejecutar hasta cinco veces consecutivas el algoritmo TRACLUS, cada vez con modificaciones en el algoritmo de clustering. Esta ejecución lineal incrementaba significativamente los tiempos, ya que incluso con pocos datos, el algoritmo mostraba lentitud.

### **Estrategia inicial: Uso de hilos**

Para abordar este problema, se reutilizó una estrategia previamente implementada: el uso de hilos. En el controlador `clustering.py`, que ya gestionaba correctamente el flujo de carga de datos, se dividió el código en funciones separadas, una por cada algoritmo de clustering. Estas funciones eran llamadas según las selecciones del usuario mediante la biblioteca `threading`.

### **Problemas con la generación de mapas y tablas**

Posteriormente, se intentó incorporar la creación de mapas y tablas a estos hilos. Sin embargo, surgieron incompatibilidades con la biblioteca `matplotlib`, que no es compatible con librerías de hilos como `concurrent.futures` o `threading`. Por este motivo, se decidió separar estas tareas. Una vez finalizados todos los hilos, se invocaba una nueva función encargada de generar los mapas necesarios para visualizar los datos.

### **Limitaciones de Python**

Aunque esta optimización redujo el tiempo de ejecución, los resultados no alcanzaron las expectativas iniciales. Se identificó que la principal limitación era Python. Su configuración interna impide utilizar múltiples núcleos del procesador de manera eficiente, lo que restringía la velocidad máxima en dispositivos locales. Sin embargo, estas limitaciones no afectaban significativamente la aplicación en entornos remotos, donde el alto rendimiento no era una prioridad debido a los costes asociados.

### **Exploración de alternativas de paralelización**

Dado el potencial escalamiento de la aplicación, se exploraron otras estrategias de paralelización: - `asyncio`: Diseñada para tareas intensivas en I/O, pero no adecuada para este caso. - `Batch Processing`: Orientada a flujos simples, tampoco era aplicable. - `Dask` y `ProcessPoolExecutor`: Ambas opciones cumplían con los requisitos del proyecto.

Se optó por `ProcessPoolExecutor` debido a su menor complejidad y capacidad para ejecutar funciones en diferentes núcleos del procesador.

### Optimización del flujo de datos

Además del uso de `multiprocessing`, se unificó el flujo de representación de los datos en las funciones de multiproceso, lo que resolvió incompatibilidades previas con `matplotlib`. Esto incrementó considerablemente el rendimiento, ya que anteriormente estas tareas se ejecutaban en serie.

### Resultados obtenidos

Tras implementar estas mejoras, se lograron resultados significativos: - **Operaciones singulares:** La ejecución de un solo TRACLUS con 100 filas de datos pasó de 312 segundos a 288 segundos. - **Ejecuciones simultáneas:** La ejecución de cinco TRACLUS con sus respectivos algoritmos de clustering se redujo drásticamente, de 18,000 segundos a 450 segundos.

Esta optimización no solo mejoró el rendimiento, sino que también eliminó errores previamente asociados al uso de hilos.

## 5.4. Despliegue de la aplicación

Para que la aplicación web funcionase correctamente y otros usuarios pudieran utilizarla, era necesario desplegarla desde el entorno local a un entorno de producción. Existen dos formas principales de realizar este proceso: crear un servidor propio o subir la aplicación a un servidor en la nube proporcionado por un tercero. Por razones económicas, se decidió optar por la segunda opción y utilizar un servidor en la nube. Para ello, se exploraron diversos servicios compatibles con aplicaciones basadas en Python.

Entre las opciones evaluadas se encontraron AWS, Azure, Heroku y Conduktor. Aunque muchas de estas plataformas ofrecían soluciones robustas con buen rendimiento, todas presentaban limitaciones económicas, ya que sus funcionalidades avanzadas suelen estar asociadas a costos recurrentes. Esto nos llevó a buscar servicios que ofrecieran una base gratuita que cubriese nuestras necesidades principales.

### Render: la solución elegida

Tras evaluar diferentes opciones, se decidió utilizar **Render**, una plataforma que permite el despliegue de aplicaciones web, APIs y otros servicios. Render ofrece un plan gratuito que resulta ideal para proyectos pequeños o

de desarrollo inicial, lo que lo convierte en una opción atractiva para aquellos con presupuestos limitados.

Render proporciona varias ventajas para aplicaciones como la nuestra:

- **Compatibilidad con Python:** Es compatible con aplicaciones basadas en frameworks como Dash, Flask o Django.
- **Despliegue automático:** Permite integrar repositorios de GitHub o GitLab para desplegar automáticamente los cambios realizados en el código.
- **Certificados SSL gratuitos:** Render ofrece certificados de seguridad SSL para garantizar conexiones seguras.
- **Facilidad de configuración:** La plataforma cuenta con una interfaz intuitiva y bien documentada, lo que facilita el proceso de configuración incluso para usuarios con experiencia limitada en despliegues.
- **Soporte para aplicaciones persistentes:** Render soporta aplicaciones que requieren bases de datos o almacenamiento adicional, ideal para aplicaciones web interactivas.

## **Proceso de despliegue en Render**

El despliegue de la aplicación en Render se realizó siguiendo los pasos descritos a continuación:

### **1. Preparación del repositorio:**

- El proyecto ya estaba alojado en un repositorio de GitHub, se incluyó un archivo `requirements.txt` que especifica las dependencias necesarias para ejecutar la aplicación.

### **2. Creación del servicio en Render:**

- Se creó una cuenta gratuita en Render.
- Desde el panel de control, se seleccionó la opción "*New Web Service*" y se vinculó el repositorio de GitHub al servicio.

### **3. Configuración del entorno:**

- Se especificó el comando de inicio de la aplicación, como `python code/app/main.py`.

- Se configuraron las variables de entorno necesarias, como claves API o configuraciones específicas de la aplicación.

#### 4. Despliegue automático:

- Render detectó automáticamente el contenido del repositorio y comenzó el proceso de construcción e implementación.
- Una vez finalizado el proceso, se asignó una URL pública para acceder a la aplicación.

#### 5. Pruebas en producción:

- Se realizaron pruebas para verificar que todas las funcionalidades de la aplicación estuvieran operativas y que no existieran problemas relacionados con dependencias o configuración.

### Consideraciones finales

Aunque Render presenta ciertas limitaciones en su plan gratuito, como tiempos de inicio más lentos para aplicaciones en estado *idle* y límites de recursos, estas no afectaron significativamente a nuestra aplicación durante el desarrollo. La elección de Render permitió concentrar esfuerzos en mejorar la funcionalidad y la experiencia del usuario, sin la necesidad de invertir en infraestructura de servidor.

Con el despliegue en Render, la aplicación quedó lista para ser utilizada por cualquier usuario con acceso a Internet, cumpliendo así el objetivo de trasladar el trabajo desde un entorno local a un entorno accesible y escalable en la nube.

## 5.5. Pruebas funcionales

Para demostrar la utilidad del algoritmo y la aplicación creados, se propuso realizar múltiples pruebas con diferentes conjuntos de datos, tamaños y configuraciones aplicados a los algoritmos de clustering.

### Conjuntos de datos

Durante el desarrollo del proyecto, se utilizó prácticamente todo el conjunto de datos de Trayectorias Taxis [?]. Para esta comprobación final, esto no era suficiente. Usar solo este conjunto de datos limitaría el proyecto a un análisis específico. Por lo tanto, se buscó encontrar múltiples conjuntos

de datos cuyo único requisito fuera contener una columna con coordenadas en formato JSON de la siguiente forma: `[[latitud_1, longitud_1], [latitud_2, longitud_2], ...]` y que el nombre de dicha columna fuera POLYLINE.

Al no encontrar conjuntos de datos con esta estructura exacta, se decidió adaptar otros conjuntos más comunes que incluyeran coordenadas geográficas en una columna.

El primer conjunto con esa estructura adaptada fue Geolife [?]. Este estudio organiza los datos en carpetas, una por cada sujeto al que se le registraron ubicaciones durante un periodo de tiempo. Dentro de cada carpeta se encontraban varios archivos .plt que contenían información sobre la latitud, longitud, hora y fecha de las mediciones.

Para organizar los datos de forma lógica para el algoritmo TRACLUS, no era viable crear una fila por cada archivo .plt, ya que algunos contenían más de 1000 mediciones. Por lo tanto, se decidió agrupar las mediciones por hora. Todas las mediciones tomadas dentro de la misma hora se combinaron en una única fila de un archivo Excel.

Se creó una función para automatizar este proceso por carpeta, lo que permitió realizar pruebas fácilmente en diferentes sujetos. Para analizar los resultados, se utilizó un archivo Excel generado con todos los datos encontrados en la carpeta del sujeto 000.

Otro conjunto de datos probado fue uno de MoveBank [?], una plataforma que contiene miles de trayectorias de animales. Debido a la inmensidad de opciones disponibles, se seleccionó de forma semi-aleatoria el conjunto de datos titulado "*Hammer-headed fruit bats (Hypsignathus monstrosus) in the Republic of Congo*". Este conjunto incluía datos de múltiples murciélagos de la fruta registrados en diferentes días.

Tras analizar el número de mediciones por hora, se decidió dividir las trayectorias por murciélagos y día, creando un conjunto de datos con filas más reducidas en comparación al anterior. Aun así, estas filas eran considerablemente más grandes que las del conjunto de Trayectorias Taxis, lo cual incrementó significativamente el tiempo de procesamiento por cada fila analizada.

Ademas de estos tres se hicieron pruebas con otros como Citi Bikes [?] o Foursquare-NY [?] pero aunque organizándonos como los anteriores si que funcionaban se daba la peculiar condicion de que no se generaban los mapas de fondo lo que perdía parte de la razon del estudio, casualmente los dos se hubicaban en los estados unidos, nueva york, lo que seguramente es la

razon auque hasta el momento se habian usado hubicaciones muy diversas, portugal, china y el congo.

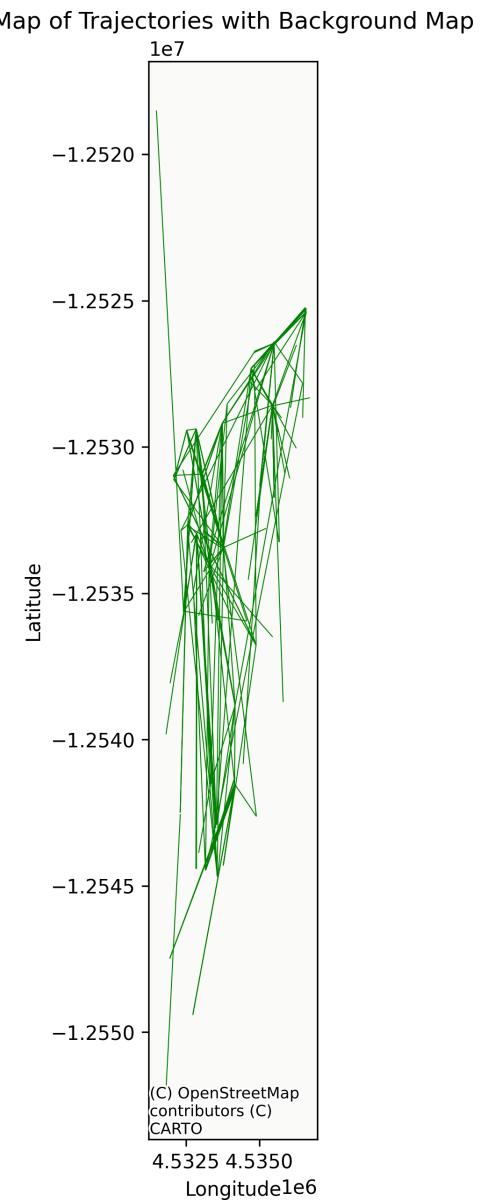


Figura 5.16: Mapa de trayectorias sin tratar del conjunto Citi Bikes.

## Explicación de las pruebas

Para realizar un análisis coherente del TRACLUS con los diferentes algoritmos de clustering, es esencial definir claramente los objetivos, identificar los parámetros ajustables y establecer los límites de las pruebas.

El propósito de estas pruebas era comprobar el funcionamiento del programa desarrollado durante el proyecto, evaluar los tiempos de ejecución bajo diversas condiciones y analizar cómo los diferentes parámetros de los algoritmos de clustering impactan los resultados obtenidos.

En cada algoritmo de clustering, existen parámetros configurables que influyen significativamente en los resultados. Los parámetros seleccionados para incluir en la aplicación son aquellos que se consideraron más relevantes para el análisis. A continuación, se explican en detalle los algoritmos y sus parámetros:

1. **OPTICS (Ordering Points To Identify the Clustering Structure):** Este algoritmo identifica clusters de diferentes densidades en los datos. Los principales parámetros configurables son:
  - a) **Metric:** Define la métrica utilizada para calcular las distancias entre puntos. Las opciones disponibles son:
    - 1) **euclidean:** Utiliza la distancia euclíadiana tradicional.
    - 2) **l1:** Calcula la distancia Manhattan.
    - 3) **l2:** Similar a euclidean, pero con ajustes para ciertas aplicaciones.
    - 4) **manhattan:** Igual que *l1*, mide la distancia entre dos puntos como la suma de sus diferencias absolutas.
    - 5) **cosine:** Calcula la similitud basada en el coseno del ángulo entre vectores.
    - 6) **cityblock:** Otra forma de describir la distancia Manhattan.
  - b) **Algorithm:** Selecciona el método para encontrar vecinos más cercanos:
    - 1) **auto:** Elige automáticamente el método más eficiente según los datos.
    - 2) **ball\_tree:** Usa una estructura de árbol espacial para organizar los puntos.
    - 3) **kd\_tree:** Similar a *ball\_tree*, pero optimizado para ciertas métricas.

- 4) **brute:** Calcula las distancias directamente, sin estructuras optimizadas.
  - c) **Max\_eps:** Establece el radio máximo para considerar puntos vecinos.
  - d) **Min\_samples:** Define el número mínimo de puntos necesarios para formar un cluster.
2. **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):** Este algoritmo agrupa puntos densos mientras ignora ruido. Comparte parámetros con OPTICS:
- a) **Metric:** Igual que en OPTICS.
  - b) **Algorithm:** Igual que en OPTICS.
  - c) **Eps:** Especifica el radio para considerar vecinos.
  - d) **Min\_samples:** Igual que en OPTICS.
3. **HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise):** Este algoritmo extiende DBSCAN para trabajar mejor con datos de densidad variable:
- a) **Metric:** Igual que en OPTICS.
  - b) **Algorithm:** Igual que en OPTICS.
  - c) **Min\_samples:** Igual que en OPTICS.
4. **Agglomerative Clustering:** Este algoritmo agrupa puntos en una jerarquía utilizando diferentes criterios de vinculación:
- a) **Metric:** Igual que en OPTICS.
  - b) **Linkage:** Determina cómo calcular la distancia entre clusters:
    - 1) **ward:** Minimiza la varianza dentro de los clusters.
    - 2) **complete:** Maximiza la distancia entre puntos más lejanos de clusters diferentes.
    - 3) **average:** Calcula la distancia media entre todos los puntos de dos clusters.
    - 4) **single:** Minimiza la distancia entre los puntos más cercanos de dos clusters.
  - c) **n\_clusters:** Especifica el número de clusters deseados en los resultados.

5. **Spectral Clustering:** Este algoritmo utiliza el espectro del grafo de afinidad para realizar clustering:
  - a) **Affinity:** Define cómo construir el grafo de afinidad:
    - 1) **nearest\_neighbors:** Usa vecinos más cercanos para definir las relaciones.
    - 2) **rbf:** Utiliza una función base radial para calcular similitudes.
    - 3) **precomputed:** Trabaja con una matriz de afinidad ya calculada.
    - 4) **precomputed\_nearest\_neighbors:** Usa vecinos más cercanos predefinidos.
  - b) **Assign\_labels:** Define el método para asignar etiquetas a clusters:
    - 1) **kmeans:** Usa el algoritmo K-Means para asignar etiquetas.
    - 2) **discretize:** Utiliza discretización para asignar etiquetas.
    - 3) **cluster\_qr:** Asigna etiquetas usando descomposición QR.
  - c) **n\_clusters:** Especifica el número de clusters deseados en los resultados.

Estas pruebas no solo validaron el funcionamiento de la aplicación, sino que también proporcionaron una comprensión más profunda de cómo los diferentes parámetros afectan el comportamiento de los algoritmos en diversos contextos.

#### **Criterios para las pruebas:**

1. **Límite de tiempo:** Cada prueba debe durar un máximo de dos horas, considerando tanto el tiempo de ejecución como el de representación de los datos. Este límite puede variar según el tamaño del conjunto de datos y el número de coordenadas procesadas.
2. **Selección de parámetros:** No se probarán todas las combinaciones posibles. Se seleccionarán aquellas configuraciones que se consideren más relevantes y que puedan producir cambios significativos en los resultados.
3. **Pruebas iniciales intensivas:** Se realizarán pruebas más pesadas al inicio, aunque en menor cantidad, para estimar mejor los tiempos de ejecución y obtener resultados más representativos.

## Resultados

Las pruebas iniciaron con un análisis de rendimiento. Primero, se evaluó cómo funcionaría la aplicación al ejecutarse en remoto utilizando Render. Los resultados fueron poco satisfactorios debido a varias limitaciones inherentes a la versión gratuita de Render, tales como un rendimiento bajo y restricciones significativas de memoria. Dado que el programa requiere recursos considerables para procesar correctamente grandes cantidades de datos o ejecutar múltiples algoritmos de clustering de forma simultánea, se produjeron errores al superar la memoria disponible o por tiempos de ejecución excesivos.

Debido a estas limitaciones económicas, se optó por realizar todas las pruebas de manera local.

### Configuración inicial

Se comenzaron las pruebas utilizando diferentes conjuntos de datos y ajustando los parámetros a valores iniciales básicos. Los datos de prueba consistieron en 10 filas de cada conjunto, y se utilizaron los cinco algoritmos de clustering con los siguientes parámetros iniciales:

- **Metric:** euclidean
- **Algorithm:** auto
- **Min\_samples:** 5
- **Max\_eps:** 1
- **Eps:** 0.1
- **Linkage:** ward
- **Affinity:** nearest\_neighbors
- **Assign\_labels:** kmeans
- **n\_clusters:** 7

### Análisis de los resultados

#### Rendimiento por conjunto de datos

- **Trayectorias Taxis [?]**  
Tiempo de ejecución aproximado: **45 segundos.**
- **Geolife [?]** (ordenado por horas)  
Tiempo de ejecución aproximado: **2 minutos y 50 segundos.**
- **MoveBank [?]** (ordenado por día y id)  
Tiempo de ejecución aproximado: **1 minutos y 50 segundos.**

## Hallazgos preliminares

### 1. OPTICS y HDBSCAN producen resultados similares

En los tres conjuntos de datos, los resultados de OPTICS y HDBSCAN fueron idénticos cuando el parámetro `Max_eps` se fijó en 1. Esto sugiere que, bajo los parámetros configurados en esta aplicación, ambos algoritmos interactúan de manera muy similar. Esto es consistente con la naturaleza de ambos, ya que son derivados de DBSCAN.

### 2. Efecto del tamaño de los datos

El análisis reveló cómo el tamaño de los datos iniciales afecta los resultados. A continuación, se presentan los resultados específicos para cada conjunto de datos:

**Geolife (10 filas)** Los resultados iniciales de Geolife mostraron trayectorias extensas y separadas, lo que influyó en los resultados de cada algoritmo:

- **OPTICS y DBSCAN:** Dado que los datos estaban dispersos, estos algoritmos identificaron puntos medios de distancia con poca coherencia.
- **HDBSCAN:** Generó dos trayectorias principales, correspondientes a las zonas donde los movimientos del sujeto se concentraban.
- **Agglomerative Clustering y Spectral Clustering:** Fragmentaron los datos en zonas de interés similar a HDBSCAN, debido a la configuración de `n_clusters` en 7.

**MoveBank (10 filas)** Este conjunto presentó movimientos concentrados en un área específica, con un desplazamiento esporádico a una gran distancia. Esto produjo que los algoritmos no fueran capaces de mostrar ningún resultado visible al igual que OPTICS Y DBSCAN en el conjunto anterior.

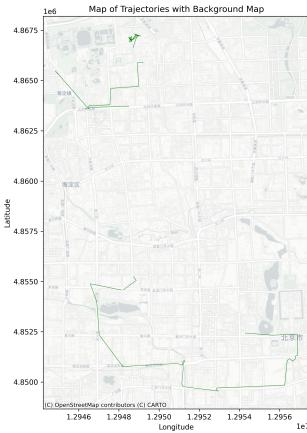


Figura 5.17: Mapa de trayectorias Geolife, 10 filas.

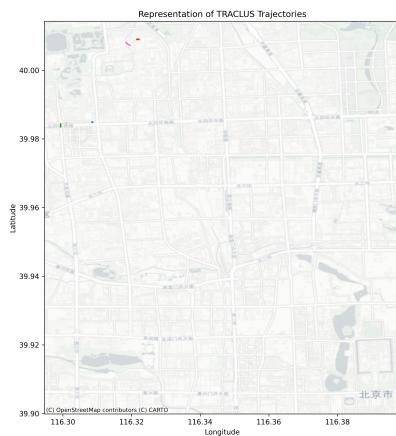


Figura 5.18: Mapa trayectorias resultantes OPTICS y DBSCAN, 10 filas.

**Trayectorias Taxis (10 filas)** Este conjunto mostró trayectorias más homogéneas, con menor dispersión:

- **OPTICS y DBSCAN:** Produjeron rutas consistentes.
- **HDBSCAN:** Generó tres rutas principales, siendo una de ellas significativamente más extensa y errática.

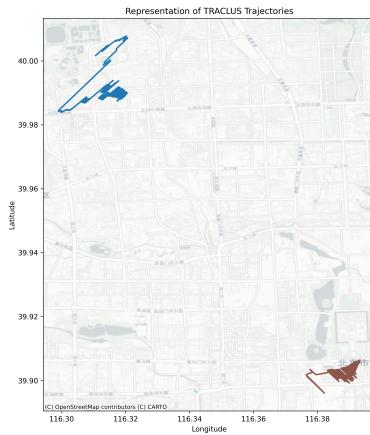


Figura 5.19: Mapa trayectorias resultantes HDBSCAN, 10 filas.

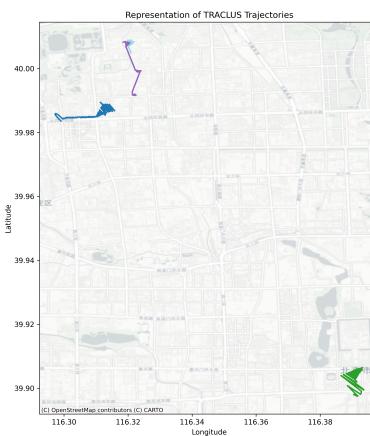


Figura 5.20: Mapa trayectorias resultantes Agglomerative, 10 filas.

- **Agglomerative y Spectral Clustering:** Igual que los anteriores algoritmos representaron bien las zonas mas concurrentes pero su resultados restaron más erráticos debido al número de clusters predefinido.

Estas pruebas iniciales revelaron diferencias significativas en el comportamiento de los algoritmos bajo diferentes configuraciones y conjuntos de datos, como el humor de culters que se generaron en todos los conjuntos,

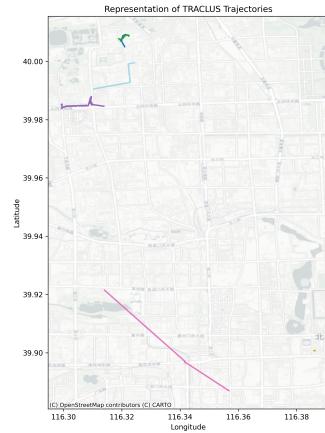


Figura 5.21: Mapa trayectorias resultantes Spectral, 10 filas.

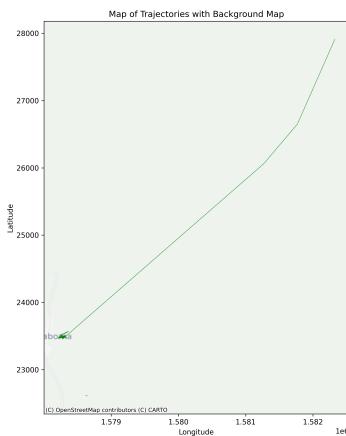


Figura 5.22: Mapa de trayectorias MoveBank, 10 filas.

cuatro en HDBSCAN y siete en el resto. Ademas consumió una cantidad de tiempo considerable lo que predijo lo extenso que podría ser comprobar y analizar resultados de diferentes configuraciones en diferentes conjuntos de datos.

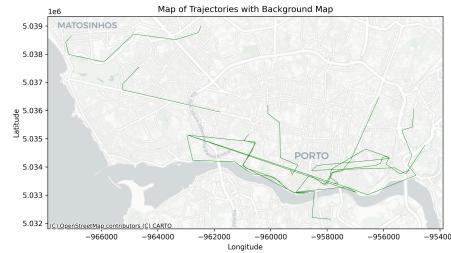


Figura 5.23: Mapa de trayectorias Taxis, 10 filas.

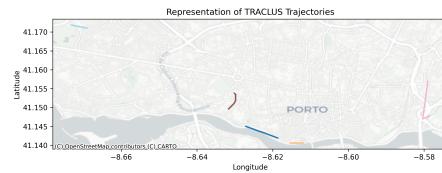


Figura 5.24: Mapa trayectorias resultantes OPTICS y DBSCAN, 10 filas.

## 5.6. Testing

Para que un código sea verdaderamente funcional, seguro y mantenible, debe someterse a pruebas exhaustivas. Durante el desarrollo del proyecto, se implementaron varios tipos de pruebas para evaluar el rendimiento y la calidad del código, así como para ampliar el conocimiento en el campo del testing.

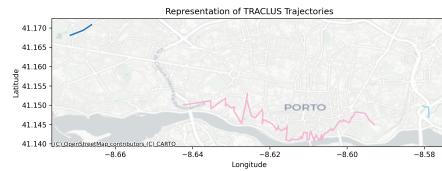


Figura 5.25: Mapa trayectorias resultantes HDBSCAN, 10 filas.

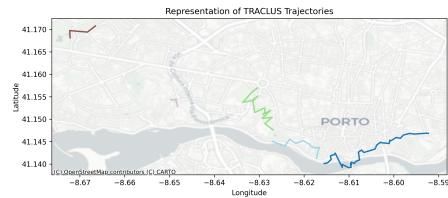


Figura 5.26: Mapa trayectorias resultantes Agglomerative, 10 filas.

## Análisis de código estático

El análisis de código estático es una técnica utilizada para identificar posibles errores en el código fuente sin necesidad de ejecutarlo. Este método es particularmente útil para evitar fallos humanos como bucles infinitos, errores de formato o problemas de nomenclatura.

Se evaluaron varias herramientas de análisis estático teniendo en cuenta las siguientes limitaciones:

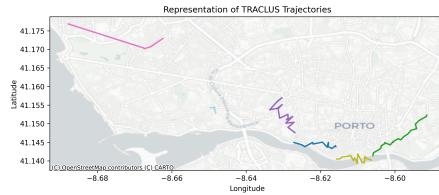


Figura 5.27: Mapa trayectorias resultantes Spectral, 10 filas.

- Debían ser compatibles con Python y CSS, los lenguajes utilizados en el proyecto.
- No debían implicar costos adicionales.
- Era deseable la integración directa con GitHub para facilitar el flujo de trabajo.

Con estas características, se seleccionaron dos herramientas principales: **SonarQube** y **Code Climate**. De estas, **SonarQube** fue la más utilizada, ya que ofrecía soporte para analizar IPython Notebooks, un formato clave en los experimentos realizados durante el proyecto. Aunque los notebooks no formaban parte directa de la aplicación web, su análisis fue crucial para garantizar la calidad de los experimentos previos.

Lo ideal en este tipo de análisis es aplicarlo de manera continua durante las diversas fases del proyecto, ya que esto reduce significativamente la carga de trabajo y mejora la calidad del código a medida que se desarrolla. Sin embargo, en este proyecto, el análisis estático se realizó al final, lo que no fue óptimo pero permitió identificar una serie de problemas, entre ellos:

- **Mejoras en nomenclatura:** Se ajustaron nombres de variables para facilitar su comprensión y evitar confusiones con otros datos.

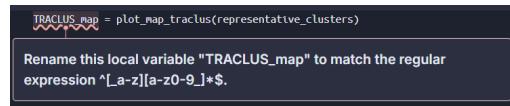


Figura 5.28: Error nomenclatura.

- **Variables no utilizadas:** Se detectaron y eliminaron variables que ya no eran relevantes, ya fuera por errores en su definición o por haber quedado obsoletas durante el desarrollo.

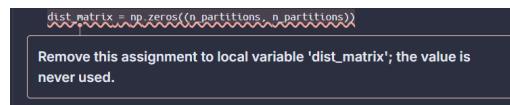


Figura 5.29: Código innecesario.



Figura 5.30: Variable sin usar necesaria.

- **Importaciones innecesarias:** Se eliminaron módulos y librerías no utilizadas, lo que ayudó a reducir la carga del programa y mejorar su legibilidad.

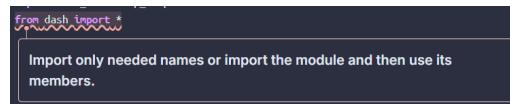


Figura 5.31: Importación excesiva.

- **Complejidad en funciones:** Se identificaron funciones cuya complejidad excedía los límites recomendados. Aunque algunas de estas funciones no pudieron simplificarse debido a las características del algoritmo, el análisis ayudó a priorizar futuras mejoras.

Aunque este tipo de análisis no corrige directamente errores en la funcionalidad del algoritmo, resulta muy útil para evitar problemas menores, mejorar la mantenibilidad del código y garantizar un nivel básico de seguridad.

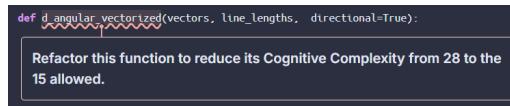


Figura 5.32: Complejidad grande.

## Pruebas unitarias y su implementación

Las pruebas unitarias son fundamentales para garantizar la funcionalidad de cada componente del código y asegurar que los resultados generados sean consistentes y correctos. Durante el desarrollo de la aplicación, se planteó inicialmente realizar pruebas unitarias utilizando `pytest` que evaluaran el comportamiento de cada botón y función de la aplicación mientras esta se ejecutaba. El objetivo era cubrir todo el flujo de la web en tiempo real, desde la interacción del usuario con los botones hasta la ejecución de los `callbacks` de Dash.

Sin embargo, esta aproximación presentó múltiples problemas. Dash, como framework basado en componentes interactivos, no es directamente compatible con las herramientas de testing tradicionales. Intentar realizar pruebas unitarias mientras la aplicación se encontraba en ejecución resultó inviable, ya que las pruebas no se ejecutaban correctamente, y la interacción con los componentes de la interfaz gráfica no podía ser simulada adecuadamente. Como resultado, las pruebas terminaban siendo llamadas directas a las funciones asociadas a los botones, sin representar el comportamiento real del flujo de la aplicación ni garantizar que los `callbacks` funcionaran en contexto.

Ante esta limitación, se decidió cambiar el enfoque hacia pruebas más efectivas y relevantes. La nueva estrategia consistió en lo siguiente:

- **Pruebas de funciones críticas:** Se probaron de manera individual las funciones más importantes de los modelos, como la implementación del algoritmo TRACLUS y su integración con diferentes algoritmos de clustering. Estas pruebas se realizaron utilizando conjuntos de datos generados de manera aleatoria para garantizar que el comportamiento del código fuese robusto ante diferentes escenarios.
- **Representación de mapas:** Se verificó que las funciones encargadas de generar mapas, tanto de segmentos como de clústeres, produjeran respuestas correctas.

---

## 6. Trabajos relacionados

---

En el desarrollo de este proyecto, fue crucial estudiar tanto los fundamentos teóricos como los trabajos previos relacionados con el algoritmo TRA-CLUS, ya que estos sirvieron como base para implementar y adaptar la solución propuesta. Este capítulo describe dos elementos clave: el artículo original que introduce TRA-CLUS y la biblioteca de código en GitHub que proporcionó una referencia práctica para la implementación del algoritmo.

### 6.1. Estudio del algoritmo TRA-CLUS

El algoritmo TRA-CLUS, introducido en el artículo *Trajectory Clustering: A Partition-and-Group Framework* [?], establece un marco novedoso para la agrupación de trayectorias basado en dos fases principales: segmentación y agrupamiento. Este enfoque busca identificar patrones significativos en conjuntos de datos espaciales y temporales, como los recorridos de taxis, rutas de navegación o trayectorias de animales.

Entre los conceptos fundamentales del algoritmo destacan:

- **Segmentación:** Las trayectorias se dividen en segmentos lineales, utilizando un enfoque de optimización que minimiza el error de representación.
- **Agrupamiento:** Los segmentos resultantes se agrupan utilizando un algoritmo de clustering basado en densidad, como DBSCAN, para identificar patrones comunes en las trayectorias.

- **Representación:** Cada grupo de segmentos se representa mediante una trayectoria representativa", que captura la esencia del grupo y permite una interpretación más clara de los datos.

Este marco de partición y agrupamiento demostró ser efectivo para manejar grandes volúmenes de datos geoespaciales, ofreciendo una solución escalable y precisa para el análisis de trayectorias. Este trabajo sirvió como base teórica para este proyecto, orientando el desarrollo y la implementación del algoritmo.

## 6.2. Biblioteca TRA-CLUS

Para complementar el estudio teórico del algoritmo, se utilizó como referencia práctica la biblioteca de código *TRA-CLUS* disponible en GitHub [?]. Esta implementación, desarrollada por Adriel Amoguis, ofrece una base funcional del algoritmo TRA-CLUS con una estructura modular y clara, ideal para ser reutilizada y adaptada en proyectos personalizados.

### Características principales de la biblioteca

La biblioteca *TRA-CLUS* destaca por:

- Una implementación directa de las dos fases principales del algoritmo: segmentación y agrupamiento.
- Código bien documentado que facilita su comprensión y modificación.
- Ejemplos prácticos que demuestran su aplicación en datasets sencillos, lo que reduce la curva de aprendizaje para los usuarios.

### Adaptaciones y mejoras realizadas en este proyecto

Aunque la biblioteca original ofrecía una base sólida, fue necesario realizar una serie de adaptaciones para cumplir con los requisitos específicos de este proyecto:

- **Optimización:** Se mejoraron ciertos aspectos del código para garantizar su rendimiento en conjuntos de datos más grandes, como los datos de trayectorias de taxis utilizados en este trabajo.

- **Integración con visualización:** Se desarrollaron herramientas adicionales para vincular los resultados del algoritmo con representaciones gráficas interactivas, facilitando la interpretación de los datos.
- **Extensibilidad:** Se modularizó aún más el código para permitir su integración en la aplicación web desarrollada.
- **Comparativas:** Se añadieron funciones para realizar comparaciones con otros algoritmos de clustering, como DBSCAN, OPTICS y HDBSCAN.

### 6.3. Impacto en el proyecto

El uso combinado del artículo original de TRA-CLUS y la biblioteca de código permitió acelerar el desarrollo y centrar los esfuerzos en aspectos diferenciadores del proyecto, como la visualización y el análisis comparativo. Además, este enfoque demostró cómo los trabajos previos pueden ser un recurso invaluable en la investigación y el desarrollo, ofreciendo tanto una base teórica sólida como herramientas prácticas para la implementación.



---

## **7. Conclusiones y Líneas de trabajo futuras**

---

### **7.1. Conclusiones**

El desarrollo de este proyecto ha permitido abordar diversas problemáticas relacionadas con el análisis y la visualización de datos geoespaciales mediante técnicas de clustering. A continuación, se resumen las principales conclusiones obtenidas durante su realización:

- Se logró implementar con éxito el algoritmo TRA-CLUS, permitiendo segmentar y agrupar trayectorias geográficas de manera eficiente, y generando resultados útiles para la interpretación de grandes volúmenes de datos.
- La aplicación web desarrollada ha demostrado ser una herramienta valiosa para visualizar, comparar y analizar los resultados de diferentes algoritmos de clustering. La posibilidad de interactuar con gráficos, mapas y tablas ha facilitado la interpretación de los datos.
- Durante las pruebas comparativas, se evidenció que cada algoritmo de clustering tiene ventajas y limitaciones dependiendo de las características de los datos y los parámetros configurados. Esto refuerza la importancia de disponer de herramientas flexibles que permitan explorar múltiples opciones.
- La adopción de la arquitectura modelo-vista-controlador (MVC) mejoró significativamente la organización y mantenibilidad del código,

facilitando la incorporación de nuevas funcionalidades y la resolución de problemas técnicos.

- El despliegue de la aplicación en un entorno de producción a través de la plataforma Render permitió validar la funcionalidad del sistema en condiciones reales, garantizando su accesibilidad a otros usuarios.

Técnicamente, el proyecto ha permitido afianzar conocimientos en áreas como el análisis de trayectorias, la programación en Python, el uso de bibliotecas de visualización y frameworks web, además de explorar técnicas de optimización y manejo de datos masivos.

## 7.2. Líneas de Trabajo Futuras

A pesar de los logros alcanzados, existen múltiples aspectos que podrían mejorarse o extenderse en futuros trabajos relacionados con este proyecto. Algunas de las líneas de trabajo futuras incluyen:

- **Optimización del rendimiento:** Implementar técnicas de optimización tanto en el procesamiento de datos como en la generación de gráficos interactivos para reducir los tiempos de carga y procesamiento en datasets más grandes.
- **Ampliación de funcionalidades:** Incluir soporte para nuevos algoritmos de clustering, como Birch o K-Means, y permitir configuraciones más avanzadas de los parámetros por parte del usuario.
- **Análisis en tiempo real:** Adaptar la aplicación para procesar datos en tiempo real, lo que sería especialmente útil en aplicaciones relacionadas con la gestión del tráfico o la monitorización de flotas de vehículos.
- **Mejoras en la interfaz de usuario:** Refinar la experiencia del usuario mediante el uso de bibliotecas avanzadas de diseño y optimización de la navegación, además de añadir herramientas de análisis más intuitivas.
- **Integración con otras fuentes de datos:** Permitir la carga y análisis de datos provenientes de diversas fuentes, como APIs de mapas en tiempo real, sensores o bases de datos externas.
- **Validación con expertos:** Realizar evaluaciones cualitativas con expertos en análisis geoespacial para identificar posibles mejoras en los resultados generados y la interfaz de la aplicación.

- **Publicación científica:** Documentar los resultados obtenidos y presentarlos en conferencias o revistas científicas relacionadas con el análisis de datos y Big Data.

En conclusión, este proyecto ha sentado las bases para el desarrollo de sistemas de análisis de trayectorias avanzados, demostrando la viabilidad y utilidad de combinar algoritmos de clustering con herramientas de visualización. Los resultados obtenidos abren un abanico de posibilidades para futuras investigaciones y desarrollos en el ámbito del Big Data aplicado a datos geoespaciales.