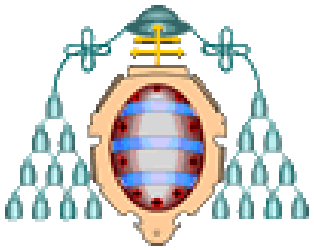


Universidad de Oviedo

Máster Universitario en Análisis de Datos para la Inteligencia de Negocios

TRABAJO FIN DE MÁSTER

Integración de Python en Power BI para analizar el *sharing* de vehículos en Barcelona.

Autores: Ayhan Hasanov Ahmedov
y Roberto Núñez Alvarez

Tutores: Carlos Enrique Carleos Artime y
Gonzalo Javier Pérez Fernández

Julio de 2021

Índice

Índice de gráficos e imágenes

1. Resumen:

En el presente informe se aborda la limpieza, transformación y extracción de conocimiento de los datos de un servicio de transportes eléctricos en Barcelona. A través de diferentes herramientas, y con ello diferentes filosofías de trabajo se busca lograr un aprovechamiento de la información para poder entender el comportamiento de los usuarios de estos servicios. Para ello se han utilizado dos herramientas diferentes, Python y Power BI. Python se ha utilizado para la limpieza y transformación de los datos, además del análisis mediante grafos del tráfico de estos vehículos por la ciudad mientras que Power BI se ha utilizado como almacén de datos y punto donde se pueden generar las relaciones entre tablas. El objetivo principal del trabajo es lograr la simbiosis entre Python y Power BI, ya que nos permitiría utilizar la versatilidad que tiene el código abierto, con la interfaz más sencilla para un usuario que no sabe programar. Al lograr la fusión entre ambos programas se sientan las bases para futuros proyectos de Machine Learning, llevando el paradigma de la programación un paso más allá.

Palabras clave: Grafo, Barcelona,Python,Power BI.

Abstract

This report deals with the cleaning, transformation and extraction of knowledge from the data of an electric transport service in Barcelona. Through different tools, and with it different work philosophies, it is sought to take advantage of the information in order to understand the behavior of the users of these services. For this purpose, two different tools have been used, Python and Power BI. Python has been used to clean and transform the data, in addition to graphing the traffic of these vehicles around the city, while Power BI has been used as a data warehouse and a point where relationships between tables can be generated. The main objective of the work is to achieve the symbiosis between Python and Power BI, since it would allow us to use the versatility that open source has, with the simplest interface for a user who does not know how to program. By achieving the fusion between both programs, the foundations are laid for future Machine Learning projects, taking the programming paradigm one step further.

Key words: graph, Barcelona,Python,Power BI.

2. Introducción

En el presente trabajo se realizará un análisis de los movimientos de los vehículos de un *sharing* de Barcelona. Los vehículos son motos eléctricas y bicicletas. El *sharing* de vehículos es un uso compartido de un vehículo. Esta es una alternativa al uso de vehículos tradicional, ya que solo pagarás por el tiempo que utilices el servicio. No importa si usas el vehículo durante dos minutos o dos semanas. Las empresas de *sharing* disponen de los datos de sus clientes sobre las ubicaciones por las que han pasado, la duración de sus viajes, el vehículo que han utilizado y, en el presente informe, se intentará dar una visión amplia de los movimientos de estos vehículos para lograr una representación de los trayectos resumida mediante grafos.

Este trabajo consta de varios apartados en los que poco a poco se irá acercando al objetivo del trabajo. Los apartados en los que trataremos temas más informativos ya fueran aquellos extraídos de diversas fuentes o los adquiridos por nosotros que puedan ser útiles para otras personas, son el de ETL y el del Marco Teórico, mientras que los 2 últimos de desarrollo tanto en Python como en Power BI entraremos más en detalle sobre cómo elaboramos el trabajo.

El primero de los últimos apartados anteriormente citados trata del trabajo que realizamos en Python con los datos desde los diferentes archivos *"txt"* donde cada uno tenía datos de un día hasta guardar estos mismo datos ya limpiados y tratados en archivos *"csv"*. El siguiente apartado principal sería el trabajo en Power BI, donde hablaremos de cómo exportamos esos datos en *"csv"*, la creación de tablas, la relación entre ellas para la construcción del modelo relacional y cómo creamos las distintas visualizaciones en Power BI.

Mientras, en el apartado de las visualizaciones en Power BI hablaremos en mayor profundidad de cómo conseguimos implementar gráficos elaborados y con cálculos en código Python en el programa, junto con los errores a los que tuvimos que hacer frente. Por tanto, este trabajo se puede considerar como una tarea de minería de datos o análisis exploratorio que ha requerido comunicación automatizada entre Power BI y Python.

3. Objetivos:

En un primer momento no teníamos objetivos concretos establecidos ya que desde la empresa no había ninguna meta especifica alcanzar, solo el simple hecho de tratar los datos y estudiar la información contenida en ellos. A medida que los datos iban siendo tratados y limpiados los objetivos fueron estableciéndose. Los datos facilitados por la empresa eran limitados ya que solo se disponían datos sobre el desplazamiento y ninguna información económica, por lo que los objetivos establecidos en este trabajo se pueden dividir en dos apartados, uno más enfocado a conceptos teóricos y otros puramente prácticos.

Objetivos teóricos

- Descripción de un proceso de ETL.
- Explicación de un grafo.

Objetivos prácticos

- Carga y limpieza de datos en formato JSON en Python.
- Carga y limpieza de coordenadas en formato GeoJson.
- Creación de tablas que recopilen la información extraída.
- Modelizar el tráfico de Barcelona mediante grafos.
- Relación entre tablas para la creación del modelo relacional.
- Creación de visualizaciones y manipulación de la información a través de filtros.
- Uso de Python en Power BI.
- Como último objetivo el dominio de ambas herramientas por separado y la unión de estas con el fin de lograr un mejor aprovechamiento de las ventajas individuales de cada una.

4. Metodología:

Para la elaboración de este informe se han utilizado fuentes como Dialnet, Google Scholar y diferentes blogs debido al carácter mayormente práctico que tiene la elaboración de este trabajo: Blogs relacionados con las comunidades de Python como *"Stack Overflow"* y Power BI como *"interactivechaos"*, *"Microsoft Docs"* y "*Support Microsoft"*.

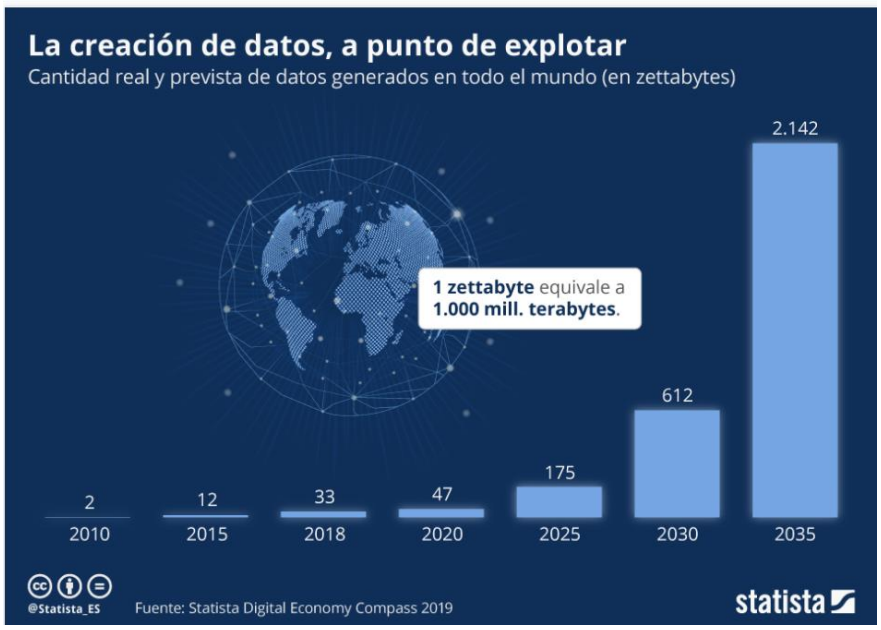
Para el desarrollo de la parte teórica de la ETL se han usado los trabajos de (Ruiz Borja, 2018) y (Sandoval Linares, 2018) como también la información obtenida en páginas webs como *"Cognodata"* y *"Agencia B12"* entre otras.

Para el desarrollo práctico del trabajo se ha usado la versión 3.9.4 de Python con el uso de paquetes estadísticos en su gran mayoría como *"pandas", numpy"*y "math" para el tratamineto de los datos, mientras que paquetes *"matplotlib"* y *"pylab"* fueron usado para las representaciones gráficas. Otros paquetes más especificos para el manejo de datos fueron *"json", "os", "bson.json_util"*, para la creación del MCA se usó el paquete *"prince"*, para el manejo de fechas se utilizó *"datetime"*, en la creación del grafo se uso *"networkx"*, finalmente se usó el paquete *"shapely"* con *"Polygon"* y*"Point*" para el tratamiento de los datos geográficos.

5. ETL:

El mundo se encuentra en la era de la información. La cantidad de datos de datos que se genera al día es enorme. Según (Statista, 2019) en el año 2018 se generaron 33 zettabytes de información (1 zettabyte equivale a 1000 millones de terabytes).

1. Gráfico cantidad de información



Fuente: Statista (Statista, 2019)

El ritmo de crecimiento es vertiginoso y las estimaciones son abrumadoras. Las organizaciones no son ajenas a estas circunstancias y desean explotar el recurso de la información porque para ellas esto puede suponer una ventaja competitiva frente a sus competidores (Sandoval Linares, 2018), además de ser un apoyo para la toma de decisiones empresariales (Ruiz Borja, 2018). Pero toda esta información está en crudo, hay que tratarla para poder extraer conocimiento de los datos. Para poder tratar esta información se sigue un proceso denominado ETL por sus siglas en inglés (Extract, transform, load). Este proceso consiste en la compilación de datos de diferentes fuentes, su tratamiento y finalmente su agregación en un único repositorio (B12, 2021). Pero el proceso de ETL no es algo novedoso, surge durante la década de los 70, aunque se popularizó durante la década de los 90. En esta década, la evolución de la ETL lo hizo de la mano de los Data Warehouses o almacenes de datos. Durante esta década empresas como SAS, Oracle o IBM lanzaron sus primeras herramientas dedicadas al proceso de ETL. Estas herramientas contaban con ciertas ventajas ya que no eran necesario programar en ellas exclusivamente en código (B12, 2021).

Fases del proceso ETL

El objetivo del proceso es producir un conjunto de datos limpio y fiable con el que poder extraer información útil o conocimiento. Para ello se debe pasar por 3 fases:

- Extracción: en esta fase se obtiene la "materia prima" con la que se trabajará en las fases posteriores. Muchas veces los datos provienen de orígenes diferentes y tienen formatos diferentes, por tanto, si contienen el mismo tipo de información deben ser compilados en las mismas estructuras (cognodata, 2019)

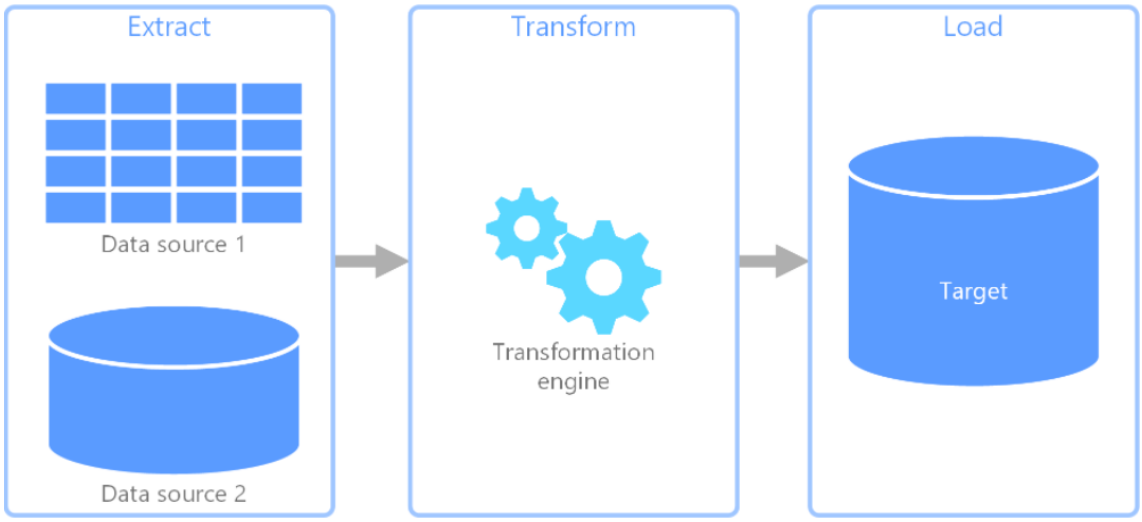
  El objetivo en esta fase es sintetizar toda la información en una o varias estructuras para que los datos normalizados puedan ser tratados (cognodata, 2019).

- Transformación: en esta etapa se requiere el manejo de los datos para poder homogeneizarlos mediante la aplicación de una serie de reglas (B12, 2021). Algunas de las transformaciones que se pueden llevar a cabo son:
  - Normalización: los datos que sean semejantes deberán aparecer con el mismo formato y estructura

- Eliminación de duplicados: esto evita sobrecargar las bases de datos con información redundante.
- Verificación: consiste en ejecutar comprobaciones automáticas para asegurar la veracidad de la información que se genera.
- Clasificación: organizar los diferentes elementos para que estos no se mezclen entre sí.

- Carga: es la fase final, es la fase en la que toda la información se almacena en único sistema de destino.

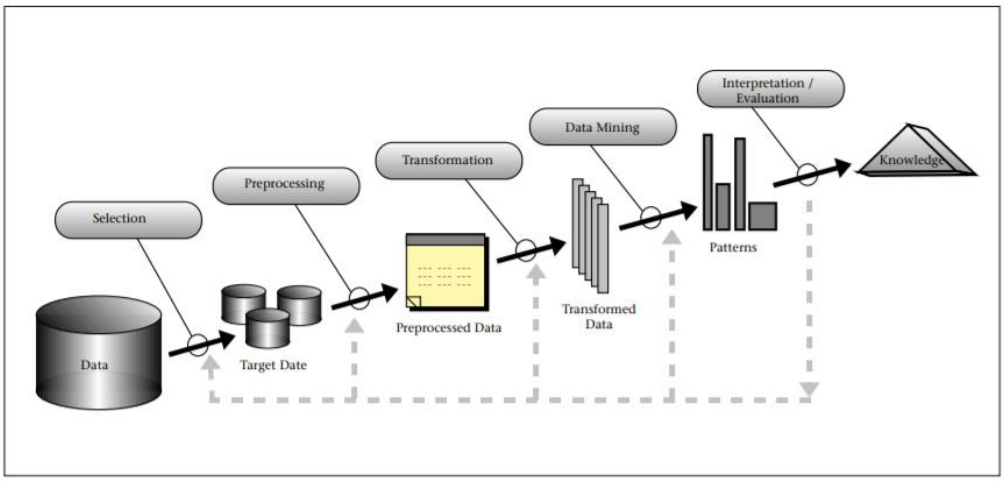
2. Gráfico ETL



Fuente: Micosoft Docs

El proceso de ETL es crucial para las empresas, ya que aporta la información necesaria para extraer conocimiento, pero para poder extraer conocimiento de los datos, previamente se tienen que tratar. *"El proceso de Descubrimiento de Conocimiento en Bases de Datos"* (KDD, Knowlegde Discovery in Database) combina conocimiento y descubrimiento (Ruiz Borja, 2018). En este proceso, se busca la extracción de patrones de los datos, tanto como para dar solución, como para prevenir de posibles problemas o pronosticar el futuro, en la toma de diferentes estrategias. En el proceso de KDD entra en juego la Minería de Datos.

3. Gráfico Etapas KDD



Fuente: (Fayyad, Piatetsky-Shapiro, & Smyth, 1997)

En el presente trabajo, los esfuerzos se han concentrado en el proceso de transformación de los datos, ya que la extracción ya se había realizado. Una vez se tenían los datos en crudo, se llevaron a cabo varias transformaciones de estos datos para normalizarlos y compilarlos en estructuras homogéneas entre ellas. Para la transformación de los datos en crudo se ha utilizado principalmente Python, debido a la versatilidad que ofrece. La posterior carga de los datos y la creación de las relaciones entre tablas se realizó en Power BI. Además, también se realizó un pequeño proceso de transformación de algunos elementos en Power BI.

6. Marco teórico:

6.1 Python:

Creado por Guido van Rossum en la década de los 80 y 90, inspirado por el desarrollo potencial del software libre y basado a su vez en otro lenguaje de programación, el ABC, el cual fue otro programa desarrollado a principios de los 80 orientado a la enseñanza. Su objetivo fue diseñar un lenguaje de programación que mantuviera esa capacidad de servir para la enseñanza, pero a la vez de ser usado para que programadores más avanzados trabajasen con él, enfocado en la reutilización de código y su adaptabilidad con el fin de en pocas líneas de código expresar conceptos que en otros lenguajes serían más arduos. (Chazallet, 2016) (Tulchak & Marchuk, 2016)

Python es un lenguaje multiparadigma, ya que permite varios estilos de programación, incluyendo la programación orientada a objetos, la cual se basa en clases y objetos, a los cuales se accede de una forma jerárquica y de alto nivel. Esto es, la sintaxis, las estructuras de programación y los algoritmos usados se adaptan de una forma sencilla a la forma de pensar humana, y no tanto a las de las máquinas. Seguidamente se expondrán de forma breve las principales características y puntos fuertes de este lenguaje:

- Contiene distintas estructuras, como listas, tuplas, listas de listas, diccionarios, etc. Esto permite al programador realizar distintas operaciones complejas de una forma sencilla, rápida e intuitiva.

- Limpieza y estructura, debido a que el lenguaje tiene que estar perfectamente delimitado, separado e indexado, presenta un aspecto visual muy agradable a la hora de revisar código por parte de otro usuario, lo que agiliza el trabajo en equipos. A esto hay que sumarle la gran variedad de opciones de personalización, que permiten adaptar los entornos de trabajo a los programadores en base a sus exigencias.

- La sintaxis de Python es una de las principales razones por las cuales es uno de los lenguajes más usados y demandados a nivel mundial, ya que es muy directo para programas que en otros lenguajes se tardaría más tiempo en realizar, y se necesitaría una mayor capacidad de conocimientos. La sintaxis de Python está denominada como de alto nivel, tal y como describe Sébastien Chazallet "*la referencia absoluta de los lenguajes de alto nivel por su extrema legibilidad y su flexibilidad que permite al desarrollador abordar una gran cantidad de casos de uso de manera muy elegante y natural*" (Chazallet, 2016).

  A continuación, se muestra un ejemplo de la sintaxis entre un lenguaje de programación a bajo nivel que es C++ (lenguaje también muy usado en el mundo) y Python:

4. Gráfico sintaxis C++.

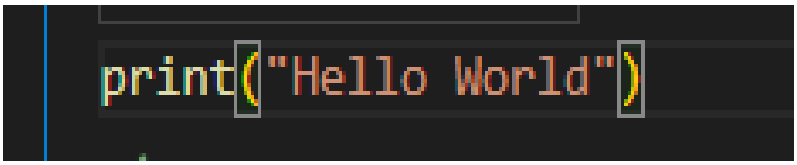
```cpp
// Your First C++ Program

#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Fuente: (Programiz, s.f.)

5. Gráfico sintaxis Python.



Fuente: Elaboración propia

Si comparamos ambas imágenes vemos a simple vista la diferencia entre ambos lenguajes, probablemente siendo Python algo más intuitivo en algunos aspectos..

- Los módulos y paquetes iniciales, otra de las ventajas de este lenguaje es la serie de librerías iniciales que permiten realizar una gran cantidad de operaciones en muchos ámbitos científicos, que abarcan desde análisis de datos hasta meteorología. A esto hay que sumarle que el propio programador puede añadir otros módulos creados por otros usuarios para facilitar el trabajo, incluso mejorarlo, lo que permite una gran capacidad de adaptación a cualquier tipo de trabajos. (Challenger-Pérez, Díaz-Ricardo, & Becerra-García, El lenguaje de programación Python , 2014)

- La comunidad es una de las más grandes a nivel global, lo que permite un acceso muy rápido a una posible solución de cualquier problema que esté experimentando el propio programador debido a que, seguramente antes alguien de la comunidad ya habrá experimentado ese mismo error, u otros muy similar; y por el uso de foros habrá recibido *"feedback"* para su solución, de la cual cualquier usuario podrá valerse. Si no siempre se puede acceder a la propia documentación de los paquetes y fórmulas de estos, con sus descripciones y funciones.
- La licencia es uno de los puntos fuertes, ya que tiene una propia, denominada Python Licence, pero que al estar certificada por Open Source permite generar programas

que luego se pueden implementar entregando o no el código fuente, lo que generará una posible mejora del código en el primer caso; y en el segundo que pueda usarse para uso empresarial y privado. (Challenger-Pérez, Díaz-Ricardo, & Becerra-García, El lenguaje de programación Python, 2014)

- Integración con otros programas que permiten importar código de otros lenguajes como C, C++, Java, Fortran, Scheme…, lo que amplia aún más el abanico de posibilidades de trabajo y mayor adaptabilidad. (Chazallet, 2016)

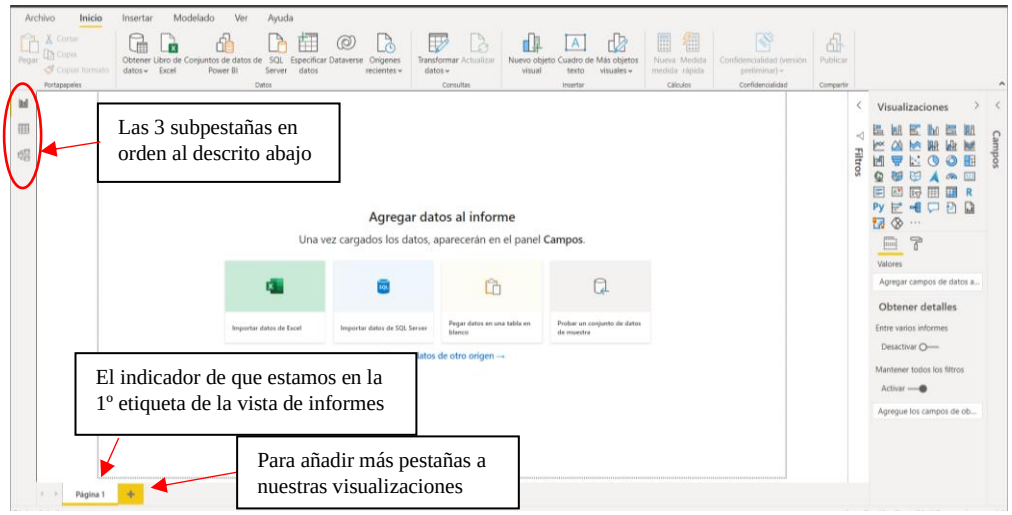
6.2 Power BI:

La creación de esta herramienta se diseñó en 2010 pero hasta el 2011 no estuvo disponible su descarga y utilización, pero no tenía una aplicación directa como tal, estaba integrada como complemento dentro de SQL server y de Excel, como Power Pivot originalmente, pero posteriormente se le añadió Power Query, Power View y Map. Cada uno tenía una función. (Burrueco, s.f.)

- Power Query: Enfocado en la transformación de los datos.
- Power View: Centrado en la visualización de datos en Excel.
- Power Map: Al contrario que el anterior este estaría enfocado a visualizaciones, pero en mapas.

No fue hasta el año 2014-2015 que se lanzó el Power BI en un formato independiente de los programas antes mencionados, tal y como lo conocemos hoy en día y desde ese día Microsoft no ha cesado en las continuas actualizaciones de manera mensual a fin de liderar el mercado de *"Business Intelligence"*. Dentro de esta herramienta podemos distinguir 2 áreas donde trabajaremos. La primera será el editor de consultas donde podremos generar visualizaciones, crear modelos de datos relacionales, cargar datos de distintos entornos… (Iseminger, 2020)

6. Imagen Inicio PBI



Fuente: Elaboración propia

En esta pestaña también podremos distinguir 3 subpestañas en la parte izquierda de la imagen, cada una estará centrada en un aspecto de los datos.

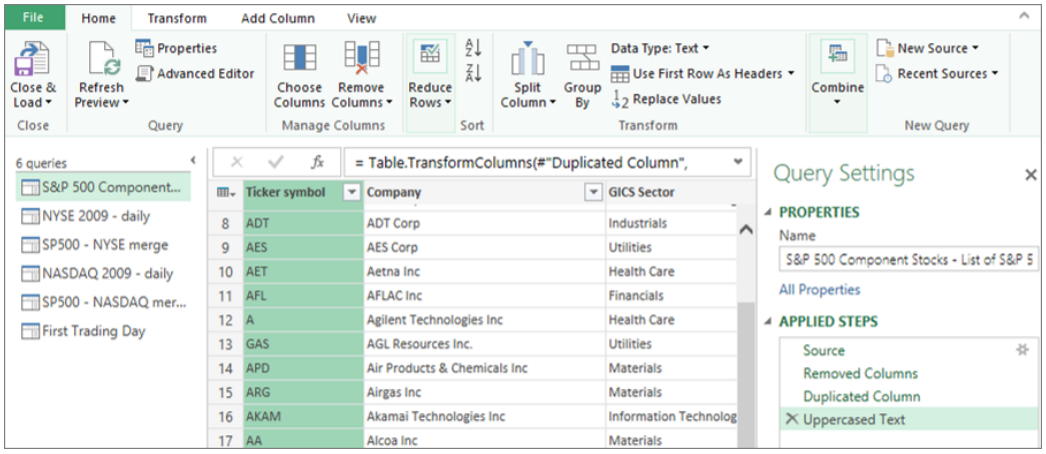
- Vista de informes: Se crean los indicadores relevantes para el Cuadro de Mando, segmentado por pestañas, tal y como podemos ver en la parte inferior de la imagen.
- Vista de datos: Apartado correspondiente al tratamiento de datos, los cuales se mostrarán mediante tablas.
- Vista de relaciones: Se corresponde con la relación de las tablas definidas en el párrafo anterior.

La segunda área será el Power Query, en el cual se desarrollará la parte de los procesos de extracción, transformación y carga de datos (ETL). Este apartado es el responsable de la transformación de los datos, puesto que permite trabajar con subconjuntos de datos y ver todos los pasos llevados a cabo por el usuario, al tener una lista de pestañas donde te indican todos los pasos dados desde la carga de los datos. También permite la actualización de los datos por si el conjunto de datos original sufrió algún cambio, esto se realizada de una manera continua y automática por parte del programa. (Llopis & olprod, 2020)

Para los usuarios más avanzados, Power BI también presenta dos lenguajes de programación para el tratamiento de los datos, esto son M y DAX. Ambos consisten en fórmulas que combinándolas se consigue la transformación de los datos. La diferencia

principal reside en la sintaxis de las funciones, y que DAX presenta una mayor cantidad de fórmulas para el tratamiento de los datos.

7. Imagen Power Query



Fuente: (Microsoft, s.f.)

6.3 Integración de Python y Power BI:

Power BI permite la implementación y ejecución de código Python dentro del propio programa para la creación y transformación de tablas. Esto ofrece una gran oportunidad de mejora para ambos programas, ya que ambos presentan puntos muy fuertes, pero otros aspectos más flojos, por lo que combinando lo mejor de ambos programas se puede ofrecer al usuario un mejor resultado.

Pros y contras de Power BI:

Como aspectos positivos y los cuales potenciarán a Python serán:

- Interfaz más amigable e intuitiva.
- No requiere tanto conocimiento para manejar el programa, al menos su parte más básica.
- Gran variedad de visualizaciones y opciones de personalización de estas.
- Implementación de modelos de datos de manera fácil y muy visual.
- Gran capacidad de creación de filtros para manejar las visualizaciones.

Mientras que los aspectos negativos de Power BI serían:

- Estas limitado a los gráficos que vienen en el propio Power BI o a los que crean los usuarios, pero para crear un gráfico necesitas un alto nivel de conocimiento, o en el caso de que haya un gráfico que necesites y esté creado, necesita un constante trabajo de actualización por parte del autor, que no siempre continúa con el trabajo.
- El lenguaje que usa Power BI para tratar y cambiar datos posee una sintaxis bastante limitada, que unido a una comunidad que ofrece su ayuda, no siempre puedes encontrar en foros las soluciones a los problemas.
- El lenguaje DAX puede ser tedioso a la hora de buscar información sobre su uso.

Pros y contras de Python:

De aspectos positivos de Python destacaríamos:

- Lenguaje más versátil.
- Tamaño de la comunidad, mejora la búsqueda y la solución de los problemas.
- Infinidad de gráficos que fácilmente puedes crear, mejorar y personalizar, al poder superponer gráficos.
- Lenguaje mejor optimizado.
- Gran capacidad de tratamiento de datos, creación de cualquier tipo de funciones en los que solo dependerá de la habilidad del programador.

Por contrapartida, los contras de Python serían:

- Una interfaz nada amigable para los usuarios más novatos.
- Se necesita una ligera idea de conocimientos de informática.
- La curva de aprendizaje si no has trabajado en otro lenguaje de programación es más dura que la de Power BI, aun siendo uno de los lenguajes de programación más sencillos.

Por tanto, el objetivo de combinar estos programas se realizará con el fin de poder tener una herramienta que sea agradable para el usuario final, de fácil manejo y muy intuitiva. La posibilidad de realizar relaciones en nuestro de modelos de datos con un simple "arrastrar y soltar" sin tener que realizar intersecciones y codificar para unir distintas tablas de datos, sobrepasar la limitación de tener que adaptarte a los gráficos existentes, y no poder crear uno tuyo propio con los requerisitos que tú necesites, o que haya solicitado el cliente. Si a esto le unimos la capacidad de superponer gráficos de Python,

obtendríamos en conjunto una gran herramienta de no solo de ETL, sino con una gran posibilidad de creación de contenido visual.

A todo esto, hay que sumarle la popularidad de Python junto a su amplia utilización en infinidad de organismos tanto públicos como privados junto con Power BI como mejor herramienta para Análisis de Datos e Inteligencia de Negocios, como bien se indica en el diagrama de Garner, más conocido como "El Cuadrante Mágico de Garner" publicado por la consultora Garner.

En dicho cuadrante podemos ver como Microsoft no solo es la líder, sino que presenta una clarísima ventaja ante el resto de sus competidores, como Tableau. Esto se debe a la gran apuesta que hizo Power BI por mejorar su herramienta de Business Intelligence ante la gran oferta de mercado que había.

8. Gráfico Garner

Fuente: (García, 2021)

7. Desarrollo en Python:

7.1 Carga y tratamiento de datos con Python

7.1.1 Qué son los archivos JSON y como se cargan

Uno de los primeros retos planteados fue cargar archivos en formato JSON en Python. El reto fue doble en estos primeros contactos con la herramienta. Por un lado, existía la dificultad de desconocer el propio lenguaje, ya que la única base de programación orientada a objetos que teníamos era el lenguaje R. Por otro lado, la dificultad que sumábamos era la de cargar y leer archivos en formato JSON, ya que estos no fueron un tipo de datos que se usaran, o se aprendieran a leer durante la duración del Máster. Teniendo en cuenta estas limitaciones, se tuvo que hacer en primer lugar un proceso de investigación sobre qué son los archivos en formato JSON y la estructura de estos. Los archivos en formato JSON, acrónimo de JavaScript Notation Object, es un formato de texto ligero que permite el intercambio de datos (ECMA, 2017).

JSON permite expresar lo que son listas de R, es decir, secuencias de objetos arbitrarios; hay secuencias de elementos anónimos encerradas en corchetes, secuencias de elementos con nombre encerradas en llaves y estás dos a su vez se pueden anidar:

- Corchetes: ["uno", "dos", 3.5]
- Llaves: {"x": 23, "y": 55}
- Anidados: [{"nombre": "lista1",
  "datos": [1, 2, 55]},
  {"nombre": "lista2", "datos": [10, 20]}]

En una lista anónima con dos elementos, cada elemento es una lista con dos elementos, el primero con nombre "nombre" y el segundo con nombre "datos".

Los archivos en formato JSON de los que se disponía tenían la siguiente forma:

9. Imagen Json datos desestructurados


```
[
{"_id":{"$oid":"60750a13cbb1260643d8e3b2"},"trip_id":"7674189","end_time":{"$date":"2021-04-12T21:52:
{"$numberLong":"1600"},"start_time":{"$date":"2021-04-12T21:45:06.000Z"},"route":{"features":[{"geome
["41.396355","2.142974","0"],["41.397884","2.140828","0"],["41.401077","2.137455","0"],["41.401750","
"2.137129","0"],["41.405370","2.137040","0"],["41.405390","2.137074","0"],["41.405495","2.137155","0"
["41.405376","2.137173","0"],["41.405403","2.137189","0"],["41.405437","2.137200","0"],["41.405468","
"2.137249","0"],["41.405550","2.137299","0"],["41.405610","2.137363","0"],["41.405533","2.137270","0"
["41.405430","2.137214","0"],["41.405350","2.137147","0"],["41.405334","2.137133","0"],["41.405270","
```


Fuente: Elaboración propia

Para que sea algo más comprensible la estructura de estos datos se ha hecho una tabulación para una mejor visualización.

10. Imagen Json datos estructurados


```
{
    "trip_id": "7674189",
    "end_time": {
        "$date": "2021-04-12T21:52:24.000Z"
    },
    "vehicle_type": "MBK125",
    "trip_distance": {
        "$numberLong": "1600"
    },
    "start_time": {
        "$date": "2021-04-12T21:45:06.000Z"
    },
    "route": {
        "features": [{
            "geometry": {
                "coordinates": [
                    ["41.396187", "2.143490", "0"],
                    ["41.396355", "2.142974", "0"],
                    ["41.397884", "2.140828", "0"],
                    ["41.401077", "2.137455", "0"],
                    ["41.401750", "2.135570", "0"],
                    ["41.404540", "2.138403", "0"],
```


Fuente: Elaboración propia

7.1.2 Cómo se cargan los datos y que información tenían

Después de adquirir algo más de comprensión sobre la estructura de este tipo de datos, se procede a la instalación de los paquetes necesarios en Python para la lectura de archivos en formato JSON, librería "*Json"* de Python, además de las librerías correspondientes para poder crear Data Frames, librería *"Pandas"*. Para que el primer contacto con este tipo de archivos fuera lo más liviano posible, se hicieron pruebas para la carga de archivos con datos correspondientes a un único día, este día sirvió de entrenamiento, para poder comprender los diferentes errores que podían surgir a la hora de cargar los archivos y

además cómo solventarlos, para que en el futuro se pudiera implementar un código que pueda leer varios días a la vez, haga la limpieza y tratamiento de todo el conjunto de datos de forma automática, solventando los posibles errores que puedan ocurrir.

Se partió de una idea inicial, en la cual se pensaba crear dos conjuntos de datos, se llamarían "*Tabla_ruta"* y "*Tabla_features",* con estos dos conjuntos de datos se pretendía tener información de: las coordenadas (latitud y longitud) por las que pasa un mismo "*TRIP_ID",* es decir, un mismo usuario, y el número de veces que se toman coordenadas de este mismo usuario, información que quedaría en "*Tabla_ruta".* Por otro lado, en "*Tabla_features"* se quería recoger información sobre el "TRIP_ID", la hora de inicio del viaje, la hora de fin, el tipo de propulsión (eléctrica, asistente eléctrico y humana), el tipo de vehículo, que podía ser "*MBK50",* motocicleta eléctrica de 50 centrimetros cúbicos, "*MBK125"* motocicleta eléctrica de 125 centímetros cúbicos y "*Bycycle"* bicicletas normales. Por último, también incluimos el nombre de los distintos proveedores que estaban suministrando el servicio que son: *'CITYSCOOT', 'YEGO', 'RIDEMOVI', 'ACCIONA', 'COOLTRA', 'MOVO', 'DONKEY', 'IBERSCOT', 'TUCYCLE', 'RESPIRO', 'RIDESHARINGS', 'AVANT'*. Esta es la información que en un principio se quería obtener con la lectura preliminar de los datos. Ambos conjuntos de datos tenían esta forma:

11. Imagen Tabla ruta

	TRIP_ID	orden	lat	lon
0	98380ae4b878f9459efbd5f2d0c949db	0	41.380142	2.174100
1	98380ae4b878f9459efbd5f2d0c949db	1	41.380142	2.174100
2	98380ae4b878f9459efbd5f2d0c949db	2	41.380142	2.174100
3	98380ae4b878f9459efbd5f2d0c949db	3	41.380188	2.174263
4	98380ae4b878f9459efbd5f2d0c949db	4	41.380250	2.174177

Fuente: Elaboración propia desde Python con datos desde JSON

12. Imagen Tabla Features

	TRIP_ID	hora_inicio	hora_fin	propulsion_type	tipo_vehiculo	empresa
0	98380ae4b878f9459efbd5f2d0c949db	2021-05-01 09:41:04	2021-05-01 10:02:37	ELECTRIC	MBK50	CITYSCOOT
1	48e2d4ec3420e06a14296a116273f393	2021-05-01 11:14:42	2021-05-01 11:18:56	ELECTRIC	MBK50	CITYSCOOT
2	bfb6bef3c7175876fce2d5df4f2f883f	2021-05-01 08:35:50	2021-05-01 08:46:54	ELECTRIC	MBK50	CITYSCOOT
3	0ad149da071a4f2c7df2c39ca289edb7	2021-05-01 06:48:11	2021-05-01 07:01:02	ELECTRIC	MBK50	CITYSCOOT
4	724dbad74903ef676b0edb67ca85925b	2021-05-01 07:56:35	2021-05-01 08:12:21	ELECTRIC	MBK50	CITYSCOOT

Fuente: Elaboración propia desde Python con datos desde JSON

Aquí vemos la carga preliminar, pero a la hora de hacer comprobaciones, se comenzaron a detectar errores que se habían producido a la hora de tomar los datos por parte de las empresas que los proporcionan.

7.1.3 Errores después de la carga de datos

1. Coordenadas de latitud y longitud intercambiadas, además de estar en alfanuméricos: en algunos casos, se intercambiaban los datos de longitud y latitud, es decir, donde debería aparecer latitud aparece longitud y viceversa, además de no ser detectados como números decimales, sino como valores alfanuméricos (cadena de texto no interpretada numéricamente). También hay coordenadas que no aparecen en Barcelona, sino que pertenecen a Zaragoza.
2. Coordenadas que son cero en latitud y longitud: algunas coordenadas para un mismo "*TRIP_ID"* no se habían recogido y aparecían ceros.
3. La *"hora fin*" no aparece como *"timestamp"*, es decir, una secuencia de caracteres que indican fecha y hora, sino que aparecen como una cadena de caracteres.
4. La hora de inicio en algunos casos es posterior a la hora final, es decir, se ha producido un intercambio a la hora de tomar los datos, y la hora de inicio está puesta como hora fin. Además, existen horas de inicio iguales a la hora final para un mismo viaje.
5. Las horas de inicio y final estaban mal tomadas, ya que aparecían con un desfase horario de dos horas, debido a la que una hora se suma por el huso horario, y otra por la hora añadida en el horario de verano. Esto provocaba que los primeros datos que se comienzan a tomar de un día concreto están adelantados dos horas, y, por

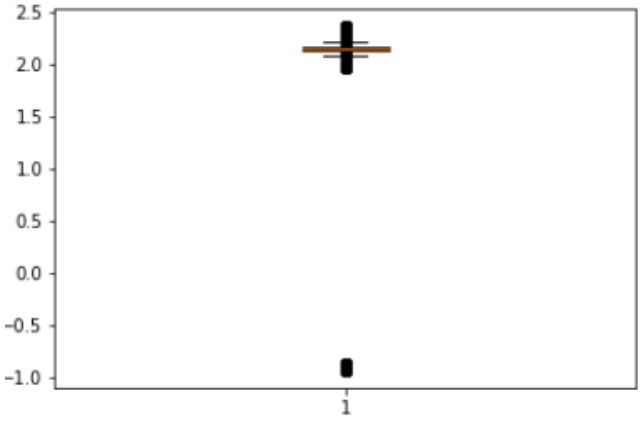
tanto, también los últimos días que se toman de un día en concreto salten al día siguiente.

6. Existen "*TRIP_ID"* con una sola coordenada.
7. Existen "*TRIP_ID"* duplicados, es decir, se repite la misma secuencia de datos varias veces, lo cual no aporta información adicional.

7.1.4 Soluciones de los errores después de la carga de datos

1. Como las coordenadas de latitud y longitud aparecen intercambiadas en algunas ocasiones, se comprobó cuáles son los valores de las coordenadas que delimitan Barcelona. Además, de este modo también se soluciona el punto 2, que las coordenadas aparecen en Zaragoza. Se llegó a la conclusión de que los valores de latitud que delimitan la ciudad de Barcelona están comprendidos entre 41 y 43, y los valores de la longitud se comprenden entre 2 y 3. Por tanto, para solucionar este problema se indicó que si el valor que se lee está entre 41 o 43 entonces estaríamos ante latitud y si está entre 2 y 3 estamos ante la longitud. Con esta solución logramos eliminar las coordenadas que aparecen en Zaragoza, ya que los valores de longitud para esta ciudad se encuentran por debajo de 0.

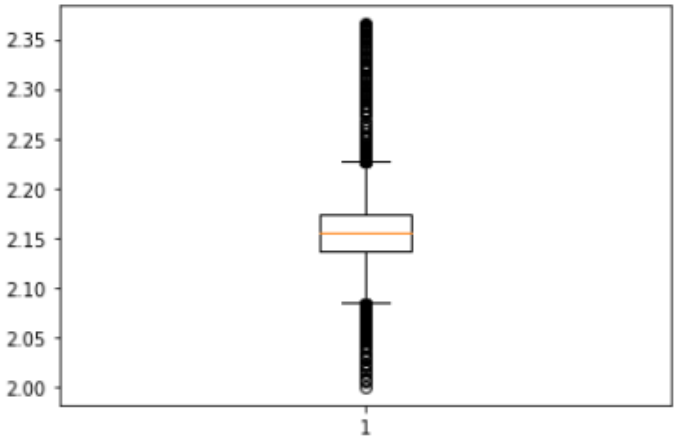
13. Gráfico de longitud con outliers



Fuente: Elaboración propia

Como podemos observar, existen outliers (valores atípicos) por debajo de 0. Al hacer la comprobación de varias coordenadas, se llegó a la conclusión de que no pertenecían a Barcelona, sino que era Zaragoza.

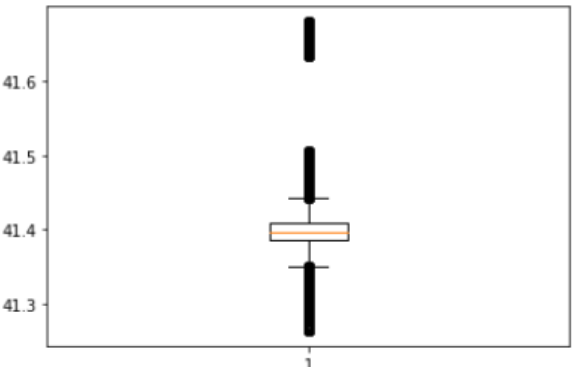
14. Gráfico de longitud sin valores atípicos



Fuente: Elaboración propia

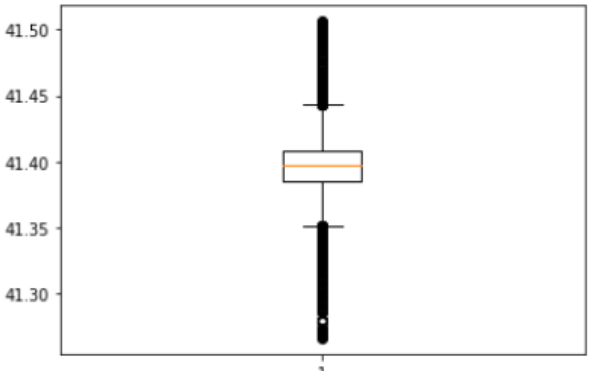
Con la solución propuesta se eliminan los valores atípicos y se delimitan las coordenadas a la región de Barcelona.

15. Gráfico de latitudes con valores atípicos



Fuente: Elaboración propia

16. Gráfico de latitudes sin valores atípicos



Fuente: Elaboración propia

Con la solución propuesta también se eliminaron esos pequeños valores atípicos en la latitud que corresponden a la zona de Zaragoza.

2. Para solucionar los problemas de coordenadas que son cero, se indicó que, si alguna de las latitudes se encuentra por debajo de 41 o por encima de 43 o alguna de las latitudes se encuentra por debajo de 2 o por encima de 3, se eliminarán del conjunto de datos.
3. Si la hora fin no aparece con el formato "*timestamp"* es suficiente con instalar el paquete "*datetime"* y ejecutar un módulo en el cual se le indica que la secuencia de caracteres que aparecen tiene forma de fecha y hora del siguiente modo (año, mes, día, horas, minutos, segundos).
4. Para solucionar el problema de la hora inicio mayor o igual que la hora final, basta con hacer una comprobación por cada dato que se carga, es decir, si la hora de inicio es igual a la hora final, se pasa a la siguiente ejecución y ese dato no se incluye dentro del conjunto de datos. Y si la hora inicial es mayor que la hora final, basta con intercambiar ambos valores.

   PseudoCódigo:

   SI (hora_inicio = hora_fin)
   pasar al siguiente dato
   SI (hora_inicio> hora_fin)
   Hora fin=temp
   Hora_inicio=hora_fin
   Hora_fin=temp
5. Después de comprobar la existencia del desfase horario al tomar los datos, lo cual provocaba que los tiempos iniciales y finales estuvieran dos horas adelantados, se optó por restar dos horas a la hora inicial y final de cada dato. De esta forma nos ajustaremos al intervalo horario real de los diferentes viajes.
6. Cuando existen "*TRIP_ID"* con una sola coordenada, es decir que se toman datos solo una vez, se elimina ese viaje del conjunto de datos.
7. Ante la existencia de "*TRIP_ID"* duplicados se opta por eliminar estos, ya que no aportan información adicional al conjunto de datos.

Una vez que se han detectado los errores que pueden aparecer en la carga inicial de los datos, y posteriormente corregido estos problemas, hay que automatizar la carga de los datos para un número de días indefinido, es decir, que se puedan cargar tantos días como

se quiera. Para ello tenemos que instalar primero un paquete en Python que se llama "*glob*", este paquete nos permite saber, dentro de la carpeta donde están nuestros archivos, cuáles se llaman de una forma en concreto, o tienen ciertos caracteres. En este caso, dentro de la carpeta en la que se trabaja, se buscan los archivos que terminan en "*.txt"* ya que cada uno de ellos contiene el fichero JSON sobre la información de los viajes correspondientes a ese día. Lo importante en este paso es tener, en nuestra carpeta, únicamente los archivos en formato "*txt"* correspondientes a los viajes, porque si no corremos el riesgo de que ocurra un error en caso de existir archivos diferentes a los viajes con ese mismo formato.

Una vez que se detectan cuáles son los archivos correspondientes a los viajes, se cargan todos esos datos en una lista de Python, con esto logramos cargar todos los días en un único objeto que Python puede leer. Y después solo tenemos que aplicar el mismo código que teníamos antes, limpiando los diferentes errores y generando "*tabla_features"* y "*tabla_ruta*" para todos los días que se encuentren en la carpeta. Con esto logramos automatizar el proceso y que el mismo código sea ejecutable para la cantidad de información que el usuario desee.

7.2 Centroides y formación de los barrios

7.2.1 Carga de barrios

Después de generar "tabla_ruta" y "tabla_features" tenemos que obtener información sobre el nombre de los barrios por los que pasan los distintos viajes, ya que tenemos las coordenadas, pero no sabemos exactamente a qué zona se corresponden. Para poder resolver este problema se dispone de un archivo *"Geojson"* con las coordenadas correspondientes a los distintos barrios de Barcelona. El formato *"Geojson"* es un "*formato de intercambio de datos geoespaciales de código abierto que representan entidades geográficas sencillas y sus atributos no espaciales*" (arcGIS, 2021). Una vez que cargamos el archivo, se vería del siguiente modo:

17. Imagen Tabla Barrios

	Barrio	coords
0	el Raval	[[2.164713785844972, 41.38593019854664], [2.16...
1	el Barri Gòtic	[[2.1770141884632963, 41.38524835715523], [2.1...
2	la Barceloneta	[[2.196228824385047, 41.3874542218253], [2.196...
3	la Dreta de l'Eixample	[[2.170908900255276, 41.40181726974143], [2.17...
4	l'Antiga Esquerra de l'Eixample	[[2.1573553158476955, 41.39330621402394], [2.1...
...	...	...
68	les Roquetes	[[2.1831474504317345, 41.4527899714784], [2.18...
69	la Trinitat Nova	[[2.188718069647769, 41.45504429166275], [2.18...
70	Torre Baró	[[2.1813687119424947, 41.46144296869927], [2.1...
71	Ciutat Meridiana	[[2.1795907562100414, 41.4640551364058], [2.17...
72	Vallbona	[[2.184065005667455, 41.46829759144268], [2.18...

Fuente: Elaboración propia

Como se puede observar, en la tabla anterior existen 73 barrios que delimitan la ciudad de Barcelona. Cada barrio este compuesto por una serie de coordenadas que lo delimitan en el espacio. Esta tabla nos permitirá hacer dos cosas:

- Dibujar el mapa de Barcelona segmentado por barrios
- Extraer por qué barrios pasan al inicio y final de cada trayecto. También se pueden saber los intermedios, pero en *"tabla features"* se incluirá solamente el inicio y el final de cada trayecto para optimizar el coste computacional, porque los primeros análisis tendrán en cuenta solamente las características básicas del recorrido; el análisis de la ruta completa se hará en un futuro.

7.2.2 Creación de Polígonos

Una vez que extraemos las coordenadas de los barrios debemos instalar un paquete en Python llamado "*Shapely"*, el cual tiene un módulo de geometría, y de él extraemos *"Polygon"* y *"Point"*. Con estas dos herramientas podemos representar los polígonos de cada barrio, es decir, su forma. También podemos crear puntos que serán las coordenadas de cada ruta en *"Tabla ruta"*. Esto nos permitirá saber a qué barrio pertenece cada coordenada de los trayectos.

18. Imagen Tabla Polígonos

	Barrios	coords
0	el Raval	POLYGON ((2.164713785844972 41.38593019854664,...
1	el Barri Gòtic	POLYGON ((2.177014188463296 41.38524835715523,...
2	la Barceloneta	POLYGON ((2.196228824385047 41.3874542218253, ...
3	la Dreta de l'Eixample	POLYGON ((2.170908900255276 41.40181726974143,...
4	l'Antiga Esquerra de l'Eixample	POLYGON ((2.157355315847695 41.39330621402394,...
...	...	...
68	les Roquetes	POLYGON ((2.183147450431735 41.4527899714784, ...
69	la Trinitat Nova	POLYGON ((2.188718069647769 41.45504429166275,...
70	Torre Baró	POLYGON ((2.181368711942495 41.46144296869927,...
71	Ciutat Meridiana	POLYGON ((2.179590756210041 41.4640551364058, ...
72	Vallbona	POLYGON ((2.184065005667455 41.46829759144268,...

Fuente: Elaboración propia

Los polígonos que se forman quedarían del siguiente modo:

19. Imagen del polígono de *"el Raval"*

Fuente: Elaboración propia

20. Imagen de *"el Raval"*

Fuente: Google Maps

Como se puede ver, se dibuja exactamente la forma de cada barrio con las coordenadas que aparecen en la tabla Polígono.

Después de generar los polígonos, debemos generar una tabla auxiliar. En esta tabla solo vamos a recoger los datos del inicio y fin de la ruta por cada identificador.

21. Imagen Tabla inicio y fin por cada identificador

	TRIP_ID	orden	lat	lon
0	98380ae4b878f9459efbd5f2d0c949db	0	41.380142	2.174100
1	98380ae4b878f9459efbd5f2d0c949db	23	41.411658	2.212152
2	48e2d4ec3420e06a14296a116273f393	0	41.379020	2.149672
3	48e2d4ec3420e06a14296a116273f393	6	41.382722	2.152275
4	bfb6bef3c7175876fce2d5df4f2f883f	0	41.364953	2.142380
...	...	...	...	...
62529	7929524	1557	41.426540	2.180084
62530	38b52eDB7520	0	41.389750	2.181186
62531	38b52eDB7520	1	41.398993	2.201827
62532	38af45AD6AD3	0	41.386097	2.170515
62533	38af45AD6AD3	1	41.371720	2.167531

Fuente: Elaboración propia

Con esta tabla auxiliar podemos agilizar el proceso de generar los puntos por cada coordenada, ya que ahora la longitud de la tabla es de 62534, y no es de más de 4 millones, como en "*tabla ruta",* la cual contiene todos los puntos por los que pasa cada identificador (TRIP_ID).

7.2.3 Creación de tabla Puntos

22. Imagen Tabla Puntos

	TRIP_ID	coords
0	98380ae4b878f9459efbd5f2d0c949db	POINT (2.1741 41.380142)
1	98380ae4b878f9459efbd5f2d0c949db	POINT (2.212152 41.411658)
2	48e2d4ec3420e06a14296a116273f393	POINT (2.149672 41.37902)
3	48e2d4ec3420e06a14296a116273f393	POINT (2.152275 41.382722)
4	bfb6bef3c7175876fce2d5df4f2f883f	POINT (2.14238 41.364953)
...	...	...
62529	7929524	POINT (2.180084 41.42654)
62530	38b52eDB7520	POINT (2.181186 41.38975)
62531	38b52eDB7520	POINT (2.201827 41.398993)
62532	38af45AD6AD3	POINT (2.170515 41.386097)
62533	38af45AD6AD3	POINT (2.167531 41.37172)

Fuente: Elaboración propia

Como se puede observar, ambas coordenadas se juntan y se crea un nuevo objeto llamado "*Point"*. Después de generar puntos y polígonos nos queda un último paso, comprobar si cada punto que se genera pertenece a alguno de los barrios. Esto se logra mediante un bucle *"for"* con el cual comprobamos si cada punto que hay en Tabla Puntos está dentro de alguno de los 73 polígonos de los barrios.

7.2.4 Creación de tabla Coordenadas

23. Imagen Tablas coordenadas

	TRIP_ID	Barrios	Posicion
0	98380ae4b878f9459efbd5f2d0c949db	el Raval	inicio
1	98380ae4b878f9459efbd5f2d0c949db	el Besòs i el Maresme	fin
2	48e2d4ec3420e06a14296a116273f393	la Nova Esquerra de l'Eixample	inicio
3	48e2d4ec3420e06a14296a116273f393	la Nova Esquerra de l'Eixample	fin
4	bfb6bef3c7175876fce2d5df4f2f883f	la Marina de Port	inicio
...	...	...	...
62529	7929524	el Congrés i els Indians	fin
62530	38b52eDB7520	Sant Pere, Santa Caterina i la Ribera	inicio
62531	38b52eDB7520	el Poblenou	fin
62532	38af45AD6AD3	el Barri Gòtic	inicio
62533	38af45AD6AD3	el Poble-sec	fin

Fuente: Elaboración propia

Como podemos observar por cada pareja de *"TRIP ID"* aparece el barrio de origen y destino. Pero a la hora de crear esta tabla, se detectó un fallo, la dimensión que debía tener era la misma que Tabla Puntos, pero al ejecutar el código, se observó que se perdían datos. Ante el problema se optó por identificar cuáles eran los *"TRIP ID"* que no aparecían en la Tabla Coordenadas y sí estaban en Tabla Puntos. Se descubrió finalmente que había un barrio que no estaba dentro de la Tabla Polígonos, por tanto, no se podía identificar cuál era, pero por este barrio habían pasado diferentes viajeros, finalmente se identificó que el barrio que faltaba en Polígonos era *"Hospitalet"*. Para solventar este problema y poder crear la Tabla Coordenadas correctamente, se incluyó dentro del bucle una opción, en la cual, si el barrio no estaba dentro de polígonos, directamente se asignará el nombre de Hospitalet, facilitando los resultados finales, ya que es un barrio bastante transitado y cercano a la ciudad de Barcelona. La Tabla Coordenadas es una tabla auxiliar que nos permitirá complementar la información que tenemos en tabla ruta, añadiendo el inicio y el final de cada viaje por cada *"TRIP ID"*. La creación de esta tabla es necesaria debido a que tabla features será utilizada como tabla *"madre"* dentro de Power BI, por lo que podemos concluir que se están preparando los datos en Python para que a la hora de usarlos en Power BI sea mucho más fácil.

Pseudo Código para crear tablas coordenadas

Bucle por cada coordenada:

Bucle por cada polígono:

Si (el polígono contiene la coordenada)

Incluye en el data frame (nombre del barrio y TRIP ID)

Si (el polígono no contiene la coordenada)

Incluye en el data frame (Hospitalet y TRIP ID)

7.2.5 Creación de tabla Centroides

Una vez generadas las tablas coordenadas y siguiendo con la parte más geométrica, se procede a extraer los centroides de cada polígono. Esto nos permitirá poder posicionar más adelante los nodos del grafo justo encima del mapa en el barrio que le corresponde. Dentro de los polígonos (POLYGON), al ser estos unos objetos de Python tienen ciertos

atributos o características, entre los cuales se encuentra el centroide, pero se tuvo que añadir manualmente el centroide de Hospitalet, para poder representar fidedignamente toda la información deseada.

24. Imagen Tabla centroides

	coords
el Raval	(2.170437430140355, 41.37899905292793)
el Barri Gòtic	(2.177311712246544, 41.38129203149088)
la Barceloneta	(2.1901483803779724, 41.377683632191356)
la Dreta de l'Eixample	(2.168203987222107, 41.393892295180514)
l'Antiga Esquerra de l'Eixample	(2.155172675512515, 41.38934720323793)
...	...
la Trinitat Nova	(2.184890450536423, 41.450607906002766)
Torre Baró	(2.1740652292036886, 41.454782460353194)
Ciutat Meridiana	(2.1748476502322522, 41.4612077364427)
Vallbona	(2.184082455782062, 41.46345746202674)
Hospitalet	(2.0997, 41.3596)

Fuente: Elaboración propia

7.2.6 Actualización de Tabla Features

Una vez extraída toda la información anterior sobre el origen y destino que pasa un "*TRIP ID"*, se puede sintetizar la información que aparece en Tabla Features.

Puesto que en Tabla Features aparece la hora de inicio y fin del trayecto, con esta información se puede saber en qué momento del día se realizan los viajes de un modo mucho más simple. Se generan 4 intervalos horarios que serán mañana, tarde, noche y madrugada del siguiente modo:

Pseudo código:

- *Si la hora se encuentra entre las 6 de la mañana y 14:00, es el periodo de mañana.*
- *Si la hora se encuentra entre las 14:00 y las 19:30, estamos en la tarde.*
- *Si la hora está entre las 19:30 y las 23:00 es el periodo de noche.*
- *Es madrugada para el resto de los casos.*

Toda esta información enriquece tabla features y permitirá hacer unos análisis más exhaustivos más adelante, con los que poder sacar información sobre el comportamiento de los usuarios de este tipo de transporte.

Para la actualización de tabla features, solo tendremos el periodo de mañana, tarde, noche y madrugada calculado para las horas de inicio.

25. Imagen Tabla Features

TRIP_ID	hora_inicio	hora_fin	propulsion_type	tipo_vehiculo	empresa	Barrios_ini	Barrios_fin	Periodo_inicio
98380ae4b878f9459efbd5f2d0c949db	2021-05-01 07:41:04	2021-05-01 08:02:37	ELECTRIC	MBK50	CITYSCOOT	el Raval	el Besòs i el Maresme	Mañana
48e2d4ec3420e06a14296a116273f393	2021-05-01 09:14:42	2021-05-01 09:18:56	ELECTRIC	MBK50	CITYSCOOT	la Nova Esquerra de l'Eixample	la Nova Esquerra de l'Eixample	Mañana
bfb6bef3c7175876fce2d5df4f2f883f	2021-05-01 06:35:50	2021-05-01 06:46:54	ELECTRIC	MBK50	CITYSCOOT	la Marina de Port	la Dreta de l'Eixample	Mañana
0ad149da071a4f2c7df2c39ca289edb7	2021-05-01 04:48:11	2021-05-01 05:01:02	ELECTRIC	MBK50	CITYSCOOT	el Clot	les Corts	Madrugada
724dbad74903ef676b0edb67ca85925b	2021-05-01 05:56:35	2021-05-01 06:12:21	ELECTRIC	MBK50	CITYSCOOT	Vallcarca i els Penitents	el Poblenou	Madrugada
...	...	...	...	...	...	...	...	...
7939882	2021-05-04 19:59:08	2021-05-05 00:00:25	ELECTRIC	MBK125	ACCIONA	la Font de la Guatlla	el Poble-sec	Noche
7932127	2021-05-04 12:00:18	2021-05-05 05:10:39	ELECTRIC	MBK125	ACCIONA	el Bon Pastor	el Fort Pienc	Mañana
7929524	2021-05-04 08:35:12	2021-05-05 05:10:15	ELECTRIC	MBK125	ACCIONA	la Sagrera	el Congrés i els Indians	Mañana
38b52eD87520	2021-05-04 20:28:34	2021-05-07 16:28:33	HUMAN	BICYCLE	DONKEY	Sant Pere, Santa Caterina i la Ribera	el Poblenou	Noche
38af45AD6AD3	2021-05-04 08:34:32	2021-05-12 06:57:45	HUMAN	BICYCLE	DONKEY	el Barri Gòtic	el Poble-sec	Mañana

Fuente: Elaboración propia

7.3 Matrices

7.3.1 Creación de la matriz final

En este apartado se procederá a la explicación de la creación de la matriz. En esta matriz se hace el recuento de viajes por barrio, es decir, las filas (origen) y columnas (destino) serán los barrios de Barcelona, y cada elemento de la matriz corresponderá al recuento del número de viajes que se han hecho desde el origen al destino. Se pretende crear una matriz cuadrada como la siguiente:

Matriz cuadrada

	el Raval	el Besòs i el Maresme	la Nova Esquerra de l'Eixample
el Raval	42	2	29
el Besòs i el Maresme	0	17	0
la Nova Esquerra de l'Eixample	39	1	189

Fuente: Elaboración propia

Para ello, en primer lugar, se hizo un recuento distintivo de los barrios que aparecen en tabla coordenadas y estos se guardaron en una lista. Se obtuvieron 67 barrios distintos, frente al total que aparecían en tabla barrios, que eran 73. Esto nos indicaba que existen barrios que estaban en la tabla barrios por los que la gente no transitaba con este tipo de

transporte. Para poder hacer la matriz cuadrada se tuvo que añadir a tabla coordenadas la posición que ocupa cada barrio dentro de la lista de los 67 barrios que se han obtenido de la tabla coordenadas.

26. Imagen Tablas coordenadas con la posición del barrio

	TRIP_ID	Barrios	Posicion	numero
0	98380ae4b878f9459efbd5f2d0c949db	el Raval	inicio	0
1	98380ae4b878f9459efbd5f2d0c949db	el Besòs i el Maresme	fin	1
2	48e2d4ec3420e06a14296a116273f393	la Nova Esquerra de l'Eixample	inicio	2
3	48e2d4ec3420e06a14296a116273f393	la Nova Esquerra de l'Eixample	fin	2
4	bfb6bef3c7175876fce2d5df4f2f883f	la Marina de Port	inicio	3
...	...	...	...	...
62529	7929524	el Congrés i els Indians	fin	31
62530	38b52eDB7520	Sant Pere, Santa Caterina i la Ribera	inicio	24
62531	38b52eDB7520	el Poblenou	fin	8
62532	38af45AD6AD3	el Barri Gòtic	inicio	40
62533	38af45AD6AD3	el Poble-sec	fin	25

Fuente: Elaboración propia

La columna posición nos permitirá saber la fila o la columna que ocupará en la matriz cada barrio, de este modo se creará una matriz cuadrada, pero para ello también hay que extraer una lista con los TRIP ID distintos que existen, en nuestro caso 31.267. Con toda esta información, se puede proceder a la creación de la matriz del siguiente modo:

- Se crea una matriz final de ceros de tamaño la longitud de la lista de barrios, en este caso 67 × 67. Esta matriz servirá para almacenar los resultados del recuento.
- Se crea un bucle que haga 31.267 iteraciones, una por cada TRIP ID distinto.
- Dentro del bucle se crea otra matriz intermedia de ceros de tamaño 67 × 67 para poder incluir un 1 en la posición que deseamos.
- Cuando entramos en el bucle se recorre cada TRIP ID. En este caso, entramos en el primer TRIP ID y se dispone de la siguiente información:

Tablas coordenadas con número de la posición

	TRIP_ID	Barrios	Posicion	numero
0	98380ae4b878f9459efbd5f2d0c949db	el Raval	inicio	0
1	98380ae4b878f9459efbd5f2d0c949db	el Besòs i el Maresme	fin	1

Fuente: Elaboración propia

- Una vez estamos dentro, se almacena la columna *"número"* en dos variables, inicio y fin respectivamente.
- Con las variables inicio y fin, que serán las posiciones de filas y columnas dentro de la matriz de ceros, incluimos un 1 en la posición que se indica en inicio y fin, del siguiente modo:

  27. Imagen de matriz de ejemplo

	0	*1 (fin)*	*2*
0 (inicio)	0	1	0
1	0	0	0
2	0	0	0

Fuente: Elaboración propia

- Después los resultados que se crean en esta matriz se van acumulando en la matriz final, ya que por cada iteración se suman, y esta matriz intermedia se renueva por cada iteración.

Tras ejecutar todo el proceso, la matriz final que se obtiene es la siguiente:

28. Imagen de la matriz final

	el Raval	el Besòs i el Maresme	la Nova Esquerra de l'Eixample	la Marina de Port	la Dreta de l'Eixample
el Raval	42.0	2.0	29.0	5.0	72.0
el Besòs i el Maresme	0.0	17.0	0.0	0.0	12.0
la Nova Esquerra de l'Eixample	39.0	1.0	189.0	12.0	188.0
la Marina de Port	3.0	0.0	15.0	34.0	15.0
la Dreta de l'Eixample	63.0	10.0	203.0	6.0	588.0
...	...	...	...	...	...

Fuente: Elaboración propia

La dimensión final de la matriz es 67 × 67, lo que nos permite saber el flujo total de viajes que se realizan intrabarrios e interbarrios. Se trata de una información interesante para poder determinar el comportamiento de los individuos y los hábitos que estos tienen. Esta matriz será la que nutra finalmente los grafos que se generen dentro de Python.

7.3.2 Creación de la matriz de flujos netos

Además de la matriz final, la cual tiene los flujos totales de los barrios, se calculó la matriz de flujos netos, es decir, los datos que aparecen son la diferencia entre origen menos destino. Número de viajes de "i" a "j" menos número de viajes de "j" a "i". Por tanto, un flujo neto positivo de "i" a "j" significa que, al final del periodo, "j" recibe más vehículos de "i" de los que le envía.

La forma de crear es la siguiente:

- Se crea una matriz de flujos netos de tamaño la dimensión de la matriz final, en este caso 67 × 67.
- Se ejecuta un doble bucle que recorra las filas y columnas de la matriz final

  Bucle por cada fila (i):

  Bucle por cada columna(k):

  Diferencia $= a_{ik} - a_{ki}$

  Matriz flujos netos (i,k) = Diferencia

29. Imagen Matriz flujos netos

	el Raval	el Besòs i el Maresme	la Nova Esquerra de l'Eixample	la Marina de Port	la Dreta de l'Eixample
el Raval	0.0	2.0	-10.0	2.0	9.0
el Besòs i el Maresme	-2.0	0.0	-1.0	0.0	2.0
la Nova Esquerra de l'Eixample	10.0	1.0	0.0	-3.0	-15.0
la Marina de Port	-2.0	0.0	3.0	0.0	9.0
la Dreta de l'Eixample	-9.0	-2.0	15.0	-9.0	0.0
...	...	...	...	...	...

Fuente: Elaboración propia

Se ha creado una matriz antisimétrica la cual contiene el flujo neto de origen – destino y ceros en la diagonal principal. La lectura sería la siguiente, de *"el Raval"* se sale más veces hacia *"el Besòs i el Maresme"* que al revés, en concreto hay un flujo neto de +2 desde *"el Raval"*. Esta información es útil para saber si hay alguna relación entre barrios más frecuentados desde otro, y cómo se puede distribuir el tráfico de vehículos eléctricos.

Como la matriz es antisimétrica y la lectura de los datos se puede hacer desde ambos sentidos, se decide optar por la elección de dejar únicamente los datos de la triangular superior, dejando los datos que están por debajo con ceros.

30. Imagen Matriz flujos netos triangular superior

	el Raval	el Besòs i el Maresme	la Nova Esquerra de l'Eixample	la Marina de Port	la Dreta de l'Eixample
el Raval	0.0	2.0	-10.0	2.0	9.0
el Besòs i el Maresme	0.0	0.0	-1.0	0.0	2.0
la Nova Esquerra de l'Eixample	0.0	0.0	0.0	-3.0	-15.0
la Marina de Port	0.0	0.0	0.0	0.0	9.0
la Dreta de l'Eixample	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...

Fuente: Elaboración propia

7.3.3 Creación de la matriz de distancias

Con la información que tenemos también podemos calcular la matriz de distancias entre barrios, aprovechando que tenemos el centroide de los barrios. Para calcular la distancia de los barrios se puede utilizar la distancia euclídea o la distancia de "*Haversine"*. La fórmula de "*Haversine"* sirve para calcular la aproximación esférica de la distancia entre dos puntos de la superficie de la Tierra (Guerrero, 2012). Como estamos en la ciudad de Barcelona la curvatura geodésica o esférica, teniendo en cuenta que la Tierra no es plana, es prácticamente despreciable, por tanto, puede ser más útil tomar la distancia euclídea (Diaz, 2012). Pero con el deseo de realizar una implementación lo más generalizable posible, se usará la fórmula de "*Haversine"*.

Para poder calcular la distancia euclídea *"debemos transformar primero las coordenadas cartesianas en coordenadas ortonormales en el espacio (x, y, z) con origen en el centro de la Tierra"* (Diaz, 2012). Si tomamos la Tierra como una esfera perfecta podemos tomar como radio 6371 km además de convertir las coordenadas de latitud y longitud en radianes.

- Transformación previa de las coordenadas para calcular la distancia Euclídea

$$x = 6371 * \cos\varphi * \cos\lambda$$
$$y = 6371 * \cos\varphi * \sin\lambda$$

$$z = 6371 * \sin\varphi$$

$$\lambda = valor\ que\ resulta\ de\ la\ Longitud$$

$$\varphi = distancia\ que\ resulta\ de\ la\ Latitud$$

- Distancia euclídea

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

La fórmula de "*Haversine"*.

$$d = 2 * r * \sin^{-1}\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos\phi_1 * \cos\phi_2 * \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}$$

$$\phi_1 = latitud\ 1$$

$$\phi_2 = latitud\ 2$$

$$\lambda_1 = longitud\ 1$$

$$\lambda_2 = longitud\ 2$$

$$r = radio\ de\ la\ Tierra, 6371\ km$$

Teniendo en cuenta estas consideraciones, se procede a calcular la distancia entre los centroides de cada barrio. La idea es poder obtener una matriz de tamaño 67 × 67 en la cual se indique la distancia que hay entre los distintos barrios. En este caso utilizaremos la distancia de "*Haversine"*. La forma de programarlo es la siguiente:

Primero se crea la función de *"H*aversine" para calcular distancias

```
def haversine():
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
```

dlat = lat2 - lat1

*a = sin(dlat/2)^2 + cos(lat1) * cos(lat2) * sin(dlon/2)^2*

*c = 2 * asin(sqrt(a))*

*km = 6371 * c*

Se crea un doble bucle para poder crear la matriz

Se crea una matriz_distancias de ceros de tamaño 67 x 67

Bucle por cada fila de la matriz final(i):

Bucle por cada columna de la matriz final(k):

Lat 1= extrae la latitud del centroide correspondiente a la posición i

Lon 1 = extrae la longitud del centroide correspondiente a la posición i

Lat 2= extrae la latitud del centroide correspondiente a la posición k

Lon 2= extrae la longitud del centroide correspondiente a la posición k

Distancia=haversine(lon 1,lat 1,lon 2,lat 2)

Matriz_distancias[i,k]= Distancia

31. Imagen de la matriz de distancias de Haversine entre barrios

	el Raval	el Besòs i el Maresme	la Nova Esquerra de l'Eixample	la Marina de Port	la Dreta de l'Eixample
el Raval	0.000000	3.867005	1.790052	2.552372	0.186231
el Besòs i el Maresme	3.864964	0.000000	5.654071	6.415989	4.051097
la Nova Esquerra de l'Eixample	1.789940	5.656704	0.000000	0.762273	1.603720
la Marina de Port	2.553117	6.421251	0.762543	0.000000	2.366831
la Dreta de l'Eixample	0.186189	4.052308	1.603453	2.365599	0.000000
...	...	...	...	...	...

Fuente: Elaboración propia

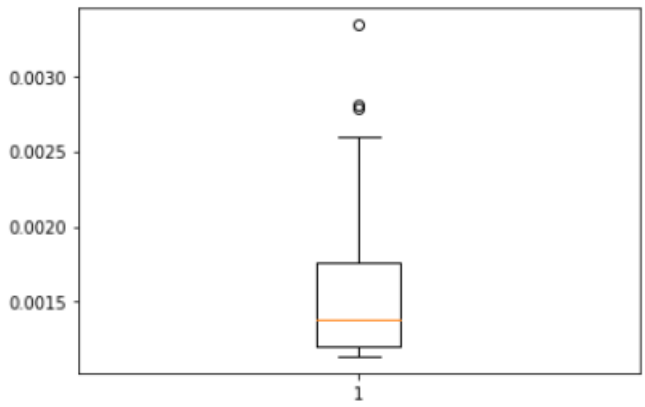
32. Imagen de la matriz de distancias euclídeas entre barrios

	el Raval	el Besòs i el Maresme	la Nova Esquerra de l'Eixample	la Marina de Port	la Dreta de l'Eixample	
el Raval	0.000000	3.869434	1.791176	2.553976	0.186348	1
el Besòs i el Maresme	3.867392	0.000000	5.657623	6.420019	4.053642	2
la Nova Esquerra de l'Eixample	1.791064	5.660257	0.000000	0.762752	1.604728	3
la Marina de Port	2.554721	6.425285	0.763022	0.000000	2.368318	4
la Dreta de l'Eixample	0.186306	4.054854	1.604460	2.367085	0.000000	1
...	...	...	...	...	...	

Fuente: Elaboración propia

Las diferencias a simple vista parecen despreciables, pero para poder tener una imagen más fidedigna entre ambas matrices, se calcula la diferencia de los resultados entre ambas, y se toma la media de las diferencias por fila. Una vez hecho esto se genera un boxplot para observar la variabilidad de la media de la diferencia de distancias entre ambas medidas.

33. Gráfico Boxplot



Fuente: Elaboración propia

Como podemos observar, la variabilidad se sitúa entre 1 y 4 metros aproximadamente. Una diferencia prácticamente despreciable.

7.4 Creación de grafo en Python

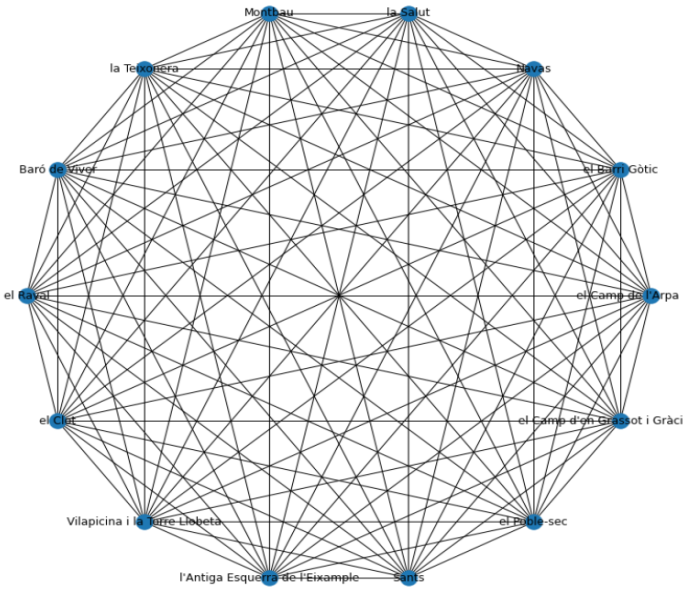
Una vez que creamos las tablas y matrices necesarias, el siguiente paso en el trabajo es modelar el tráfico de vehículos de Barcelona mediante un grafo, ya que esto permitirá al usuario entender la evolución del tráfico de vehículos de manera mucho más intuitiva.

Un grafo no deja de ser un conjunto de relaciones entre diversos objetos y relacionados entre sí por una determinada característica. En los grafos podemos distinguir 2 conceptos básicos:

- Nodos: Es el objeto que estará relacionado con otros objetos dentro del grafo, en nuestro caso serían los barrios.
- Aristas: Asocia cada uno de los nodos del grafo, en nuestra representación serán los viajes entre barrios, almacenados dentro de una matriz.

Una vez conocida la teoría de los grafos y sabiendo dónde se quería llegar, nos propusimos crear un grafo. En el primer intento se seleccionaron 14 barrios de la matriz final. El grafo creado era un grafo simple, donde cada par de vértices se conectaba por una arista.

34. Gráfico grafo simple



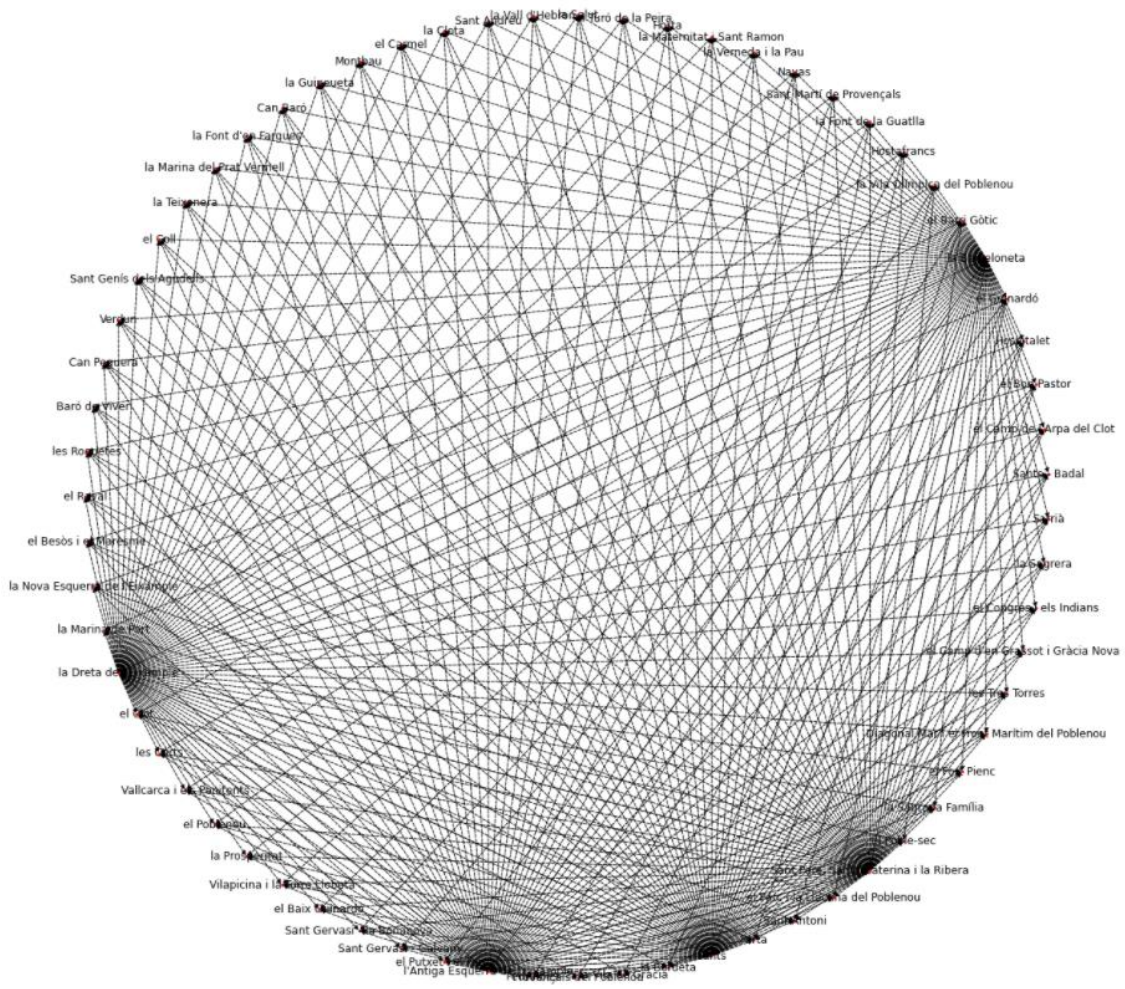
Fuente: Elaboración propia

Una vez creado este primer grafo simple, se continuó con la labor investigadora, ya que la información que aportaba en este momento no era muy abundante. Se intentó crear un

grafo dirigido, que es un tipo de grafo en el cual las aristas tienen un sentido definido. Con este tipo de grafo se podría saber cómo se desplazaban los viajeros entre los distintos barrios.

Para la creación del grafo dirigido se decidió utilizar la matriz final completa. El motivo por el que se eligió este tipo de grafo es que al ser dirigido nos permitió poder seleccionar desde que barrios (nodos) hasta que otros, existía un flujo, y más adelante ponderarlo para así poder resaltar más los flujos principales de modo que los viajes con una mayor afluencia quedarían con flechas más pronunciadas.

35. Gráfico nodo dirigido



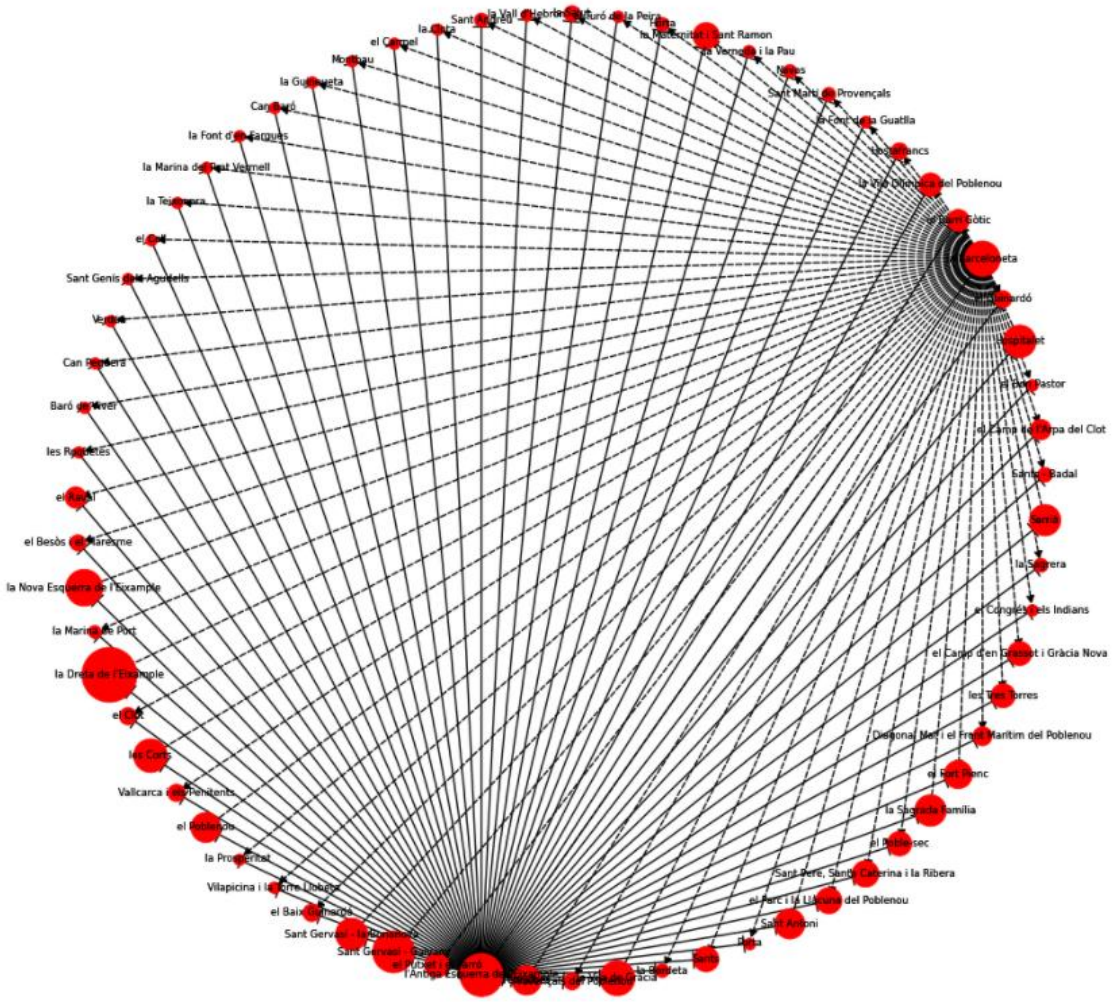
Fuente: Elaboración propia

Con este grafo se observó que había ciertos núcleos desde los cuales se recibía más flujo de tráfico, era el caso de *"La Dreta de l'Eixample" "La Barceloneta" "Sants" e*tc. Se

puede observar cómo estos barrios tienen más afluencia de flechas y por eso se terminan sombreando, esto indica mayor circulación por esa zona.

El siguiente paso era intentar añadir el tamaño de los nodos, que estos aumentaran o disminuyeran en función de la afluencia de viajes que recibían o cualquier otra función, por ejemplo, una función de coste que este definida como el producto de la distancia por el número de viajes. Finalmente se obtuvieron los siguientes resultados.

36. Gráfico grafo Python

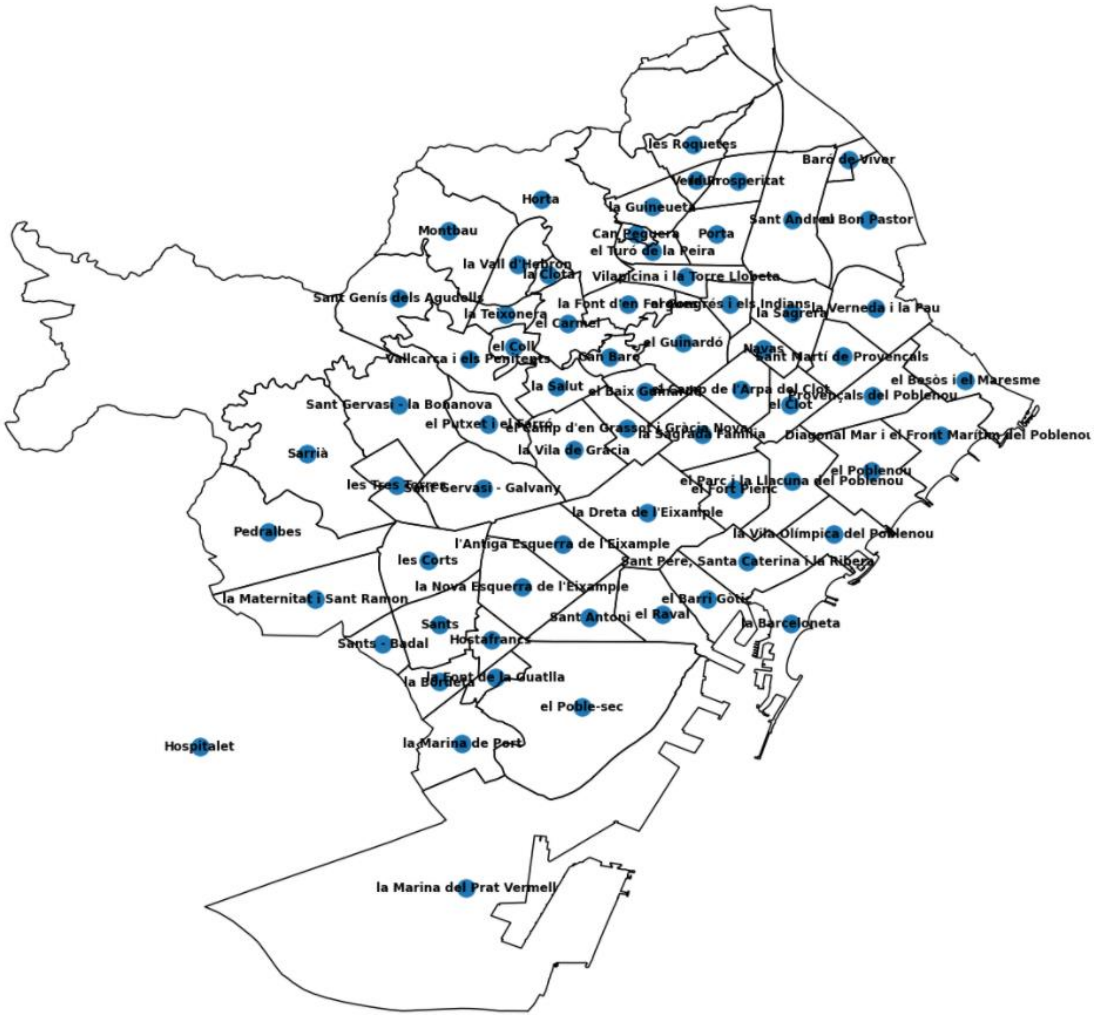


Fuente: elaboración propia

De este modo se logró ponderar el peso de los barrios en función del coste que tenían. Aunque la función de ponderación puede ser cualquiera, por ejemplo, la suma de tráfico

origen – destino, o el tráfico neto entre los barrios. Con esta información se seguía apoyando la teoría de la existencia de ciertos barrios que tienen una mayor afluencia de viajeros, como era de esperar. El siguiente paso fue intentar dibujar los nodos del grafo, en unas posiciones indicadas, que en este caso serían los centroides de los barrios. Con esto se podría situar cada nodo justo encima del barrio correspondiente y que a su vez indicara el flujo que tenía con el resto de los barrios. En primera instancia se dibujó el mapa y se incluyeron los nodos de cada barrio.

37. Gráfico con nodos en Barcelona



Fuente: elaboración propia

El siguiente paso fue introducir la ponderación de cada nodo en función del flujo de cada barrio. Como ejemplo, se crea una matriz auxiliar en la cual se toman los 8 barrios con más flujo (se tomó el flujo de origen a destino)

38. Gráfico con nodos unidos en Barcelona

Fuente: elaboración propia

Con este grafo se puede comprobar como la zona centro es una de las más transitadas por este tipo de vehículos. A este grafo se le decide incluir la ponderación de las aristas. Mediante esto se intenta que el flujo total que se produce entre los barrios quede plasmado de una forma más visual. Se pretende que las aristas que unen los nodos aumenten su grosor y la intensidad de su color en función de la cantidad de tráfico que tengan.

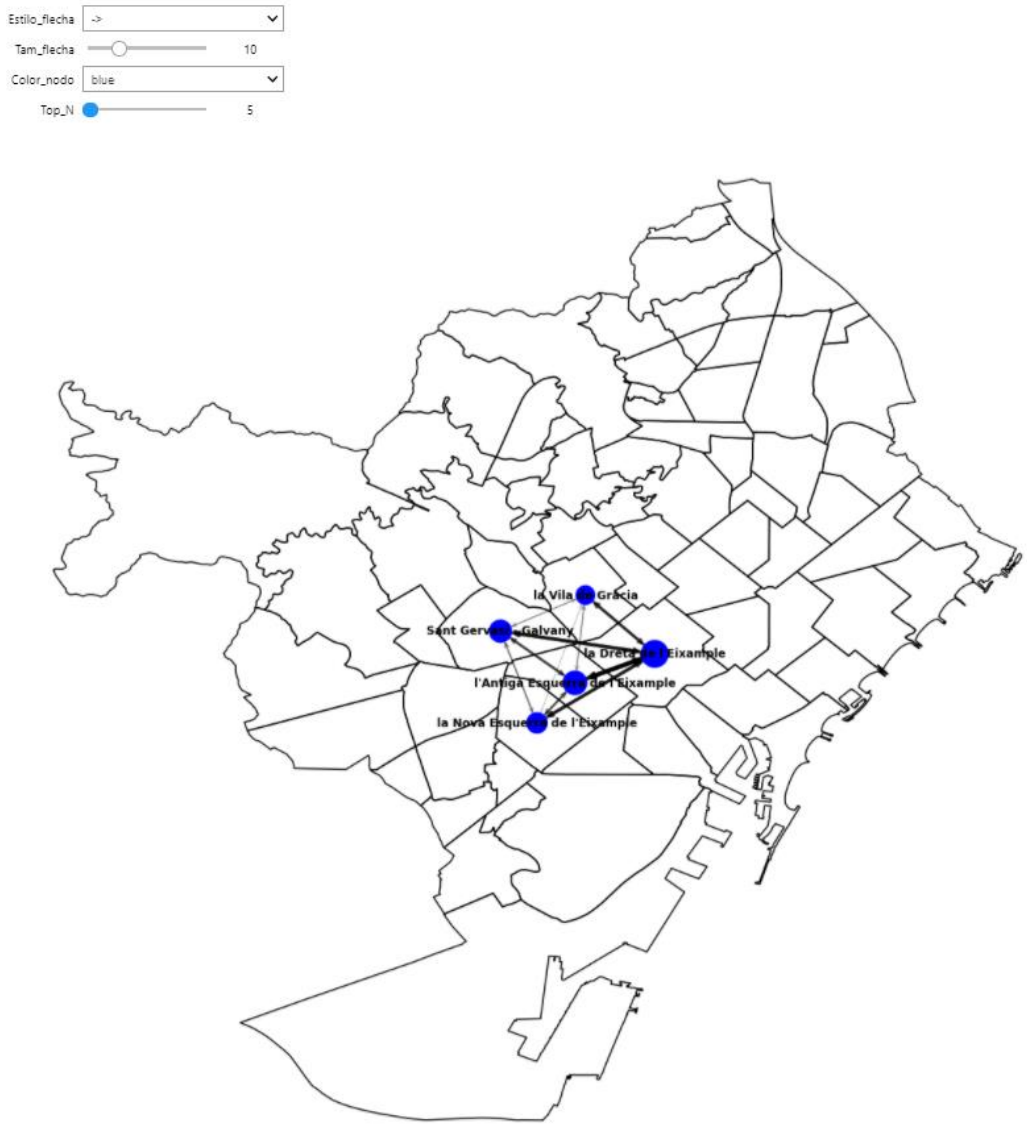
39. Gráfico con nodos unidos y diferenciados en Barcelona

Fuente: elaboración propia

En este último grafico se han seleccionado solamente 5 barrios con el objetivo de ver el efecto final que tienen las líneas. Como se puede observar *"La Dreta de l'Eixample"* es uno de los barrios con más flujo, pero cabe destacar que el uso de estos vehículos sobre todo se realiza en el centro de Barcelona.

Por último, se creó un grafo dinámico. El objetivo es intentar replicar lo máximo posible la salida en Power BI con sus diferentes filtros.

40. Gráfico dinámico



Fuente: elaboración propia

8. Desarrollo en Power BI

En este apartado trataremos todo lo que tiene que ver con la carga, la limpieza, el tratamiento y la visualización de los datos en Python, ya que, aunque los datos fueron extraídos, manipulados y limpiados a través de Python para poder representarlos en visualizaciones de Power BI hace falta realizar tratamientos adicionales.

8.1 Carga de datos

Para la carga de los datos en primer lugar se guardaron en *"csv"* desde Python para su posterior carga, esto se debe a que este formato presenta una forma sencilla de transferir la información y de separar las columnas por comas a fin de que quede legible tanto para el usuario como para el programa.

La carga desde un *"csv"* se realiza de una forma muy sencilla, ya que con unos pocos clics se puede cargar la información y separar correctamente las columnas, además hay que añadir que Power BI es capaz de actualizar de forma automática un conjunto de datos. Por tanto, el usuario solo tendrá que preocuparse de que la estructura del nuevo *"csv"* no tenga nombres cambiados en las variables que se tratarán en el Power BI y de esperar a que ese proceso de actualización se realice, que normalmente no tarda mucho.

8.2 Tratamiento y limpieza de los datos

Este paso consistió en el reemplazo de valores que se importaban como tipo texto desde el *"csv"*, por ejemplo, las latitudes y longitudes, el cambio del separador de decimales o ajustar los nombres para presentar tablas más agradables a los visuales.

41. Imagen cambio de formato

Antes de los ajustes

lat	lon
41.380142	2.1741
41.380142	2.1741
41.380142	2.1741

Después de los ajustes

Latitudes	Longitudes
41,380142	*2,1741*
41,380142	*2,1741*
41,380142	*2,1741*

Fuente: Elaboración propia

En esta imagen se puede ver a simple vista como las coordenadas están separadas por puntos y Power BI las declara en formato texto lo que imposibilita la creación de objetos visuales. A la derecha se cambian los puntos por comas y es posible guardar las altitudes y longitudes como números decimales

En este apartado también se realizó la creación de la *"tabla frecuencias"* a partir de una de las tablas principales, la *"tabla features"*, esto se hizo a raíz de la necesidad de un conjunto de datos diferente para la representación de una tabla que contase el número total de viajes entre e intra-barrios.

42. Imagen Tabla features

TRIP_ID	Hora detall	Hora detall fin	propulsion_type	tipo_vehiculo	empresa	Barrios	Barrios_fin
18868160	*01/05/2021 16:04:31*	*01/05/2021 16:12:35*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	el Baix Guinardó
18868226	*01/05/2021 16:10:00*	*01/05/2021 16:26:58*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	la Dreta de l'Eixample
18868195	*01/05/2021 16:07:20*	*01/05/2021 16:48:59*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	la Dreta de l'Eixample
18868451	*01/05/2021 16:25:32*	*01/05/2021 16:32:19*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	la Vila de Gràcia
18868264	*01/05/2021 16:13:04*	*01/05/2021 16:21:43*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	el Camp d'en Grassot i Gràcia Nova

Fuente: Elaboración propia

43. Imagen Tabla frecuencias

Barrios	Barrios_fin	Recuento	Unión
el Parc i la Llacuna del Poblenou	el Bon Pastor	*1*	el Parc i la Llacuna del Poblenou -> el Bon Pastor
Sant Gervasi - la Bonanova	el Clot	*1*	Sant Gervasi - la Bonanova -> el Clot
Hostafrancs	el Putxet i el Farró	*1*	Hostafrancs -> el Putxet i el Farró
la Vila Olímpica del Poblenou	Navas	*1*	la Vila Olímpica del Poblenou -> Navas
Porta	la Barceloneta	*1*	Porta -> la Barceloneta
Vallcarca i els Penitents	el Turó de la Peira	*1*	Vallcarca i els Penitents -> el Turó de la Peira
la Prosperitat	el Clot	*1*	la Prosperitat -> el Clot

Fuente: Elaboración propia

En la primera imagen vemos unas pequeñas filas de la *"tabla features"* de la cual solo se seleccionaron los *"Barrios"* y los *"Barrios_fin"*, y se formó otra tabla aparte, donde tendríamos por columnas, el barrio inicial, el final y una columna que sumará el número de viajes que se realizaba entre esos 2 barrios, con el nombre de *"Recuento"*, a fin de que en la tabla que se explicará más adelante, pueda ordenar por el top de la columna de *"Recuento"*.

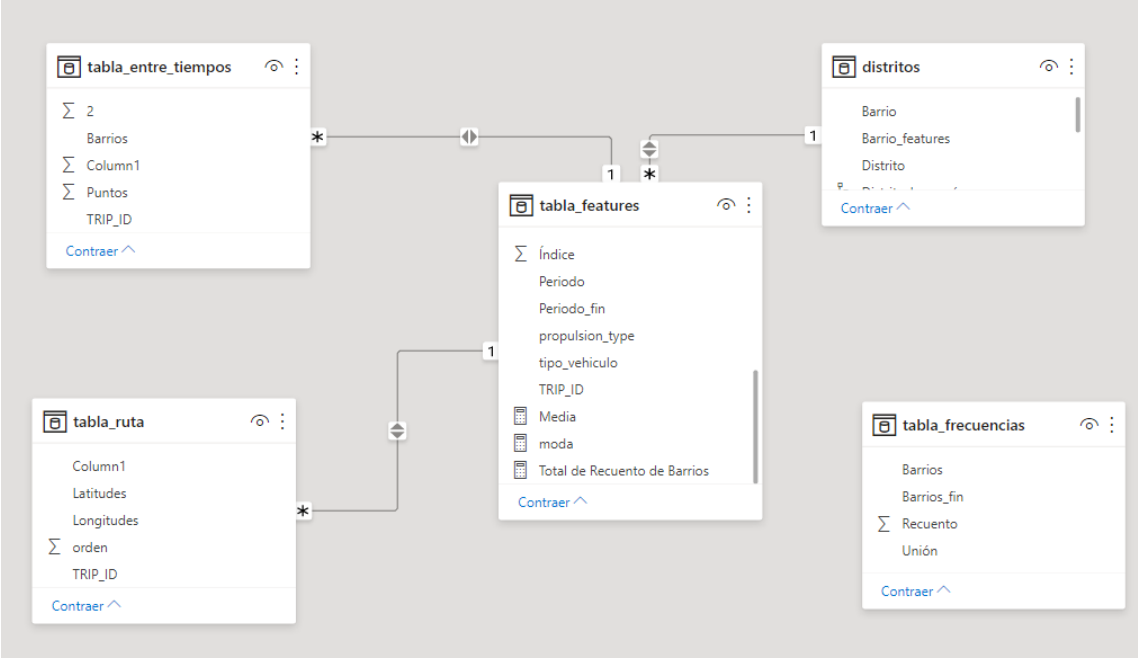
Por último, se creó una última columna con el fin de ver en la visualización, de una forma más estética, el top de los viajes entre esos barrios, solo se procedió a concatenar los nombres de las otras dos columnas y a unirlas por el símbolo "->".

8.3 Relación entre tablas y creación del modelo relacional

Para el correcto funcionamiento de los filtros para las visualizaciones creadas en Power BI como para las elaboradas en código Python es necesaria la correcta relación entre las tablas, a fin de que el cliente que esté manejando el Cuadro de Mando (herramienta de gestión de información que ofrece un conjunto de indicadores numéricos, estadísticos, etc… para una visión general) vea como los filtros funcionan adecuadamente, y sobre todo los filtros que actúan de forma directa sobre las gráficas realizadas en Python.

En la siguiente imagen vemos las relaciones entre las tres tablas originales, a saber, *"tabla features"*, *"tabla_entre tiempos"* y *"tabla_ruta"*. Las otras serían *"tabla_frecuencias"* cuyo propósito se explicó anteriormente, y la tabla de *"distritos"*, en la cual están relacionados cada uno de los barrios con el distrito al que pertenecen, para así poder generar filtros por distritos, ver qué barrios están dentro de cada distrito y observar por qué distritos se mueven las motos y bicis.

44. Imagen modelo relacional



Fuente: Elaboración propia

Entrando en un poco de mayor detalle sobre el modelo representado justo arriba, decir que las tablas fueron relacionadas en base al *"TRIP_ID"* a excepción de la tabla de

"distritos" que se relacionó por el nombre de los barrios. La tabla *"tabla_frecuencias"* no está relacionada debido a que solo se usará para la visualización del top de viajes de los barrios.

8.4 Creación de visuales en PBI

Los objetos visuales en Power BI son, sin duda, uno de los puntos fuertes de este programa y de los más usados, ya que este programa no solo es uno de los mejores en el análisis de datos por su gran capacidad de tratar y transformar datos, sino por sus visualizaciones y la amplia variedad de personalización, ya sea desde un simple fondo o un marco, pasando por diferentes tipos de filtros de búsqueda como el caso de un mero selector, hasta la aplicación de filtros en los propios gráficos con una simple visualización.

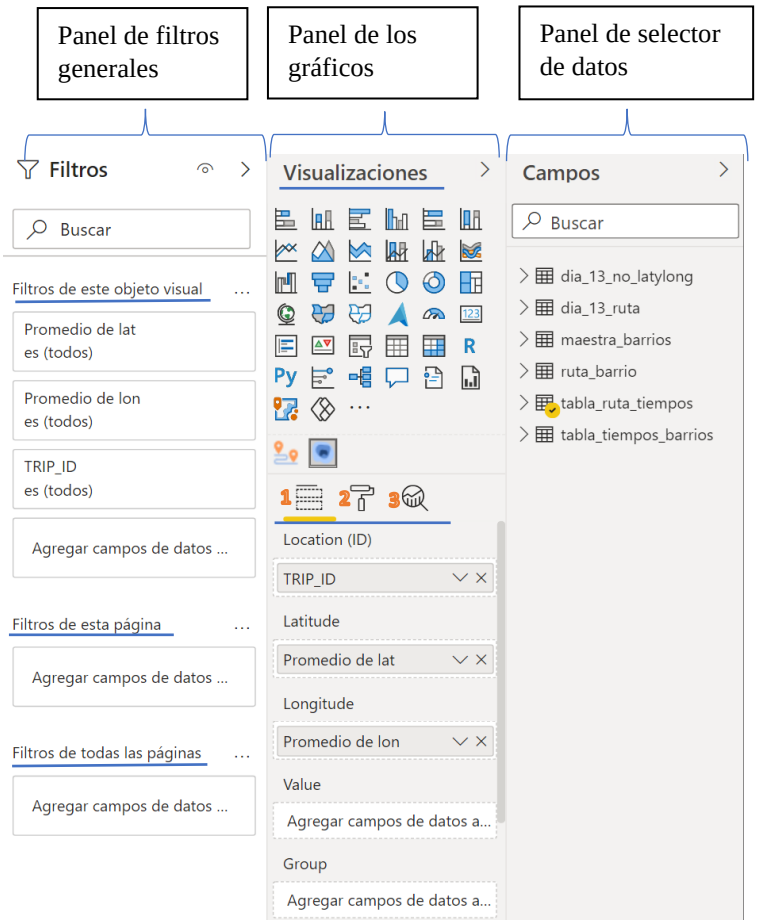
45. Imagen de visualización con filtro

Fuente: Elaboración propio

En esta imagen podemos ver un mapa de calor donde la intensidad del color indica una mayor presencia de viajes en esas zonas, pero filtrado por horas, en este caso entre las 10:00 y las 12:00. Estos filtros son facilitados por el modelo relacional y a que tenemos datos por un lado que nos dan la información del número de viajes, que en el caso que nos ocupa están adjuntados y representados en el mapa, y, por otro lado, los datos sobre la hora que se encontraban esos individuos y el lugar.

Siguiendo con el ejemplo de las visualizaciones anteriormente expuestas, nos centraremos ahora en el panel de personalización, en el cual pueden distinguirse 3 áreas subdivididas en apartados.

46. Imagen panel de personalización



Fuente: Elaboración propia

En la anterior imagen vemos claramente las tres áreas para personalizar nuestro Cuadro de Mando en Power BI, ahora profundizaremos un poco más en cada una para dar una imagen de la capacidad del programa sólo en materia de dibujos.

- Panel de filtros generales: En la imagen podemos ver 3 tipos distintos de filtros y cada uno afecta a un ámbito distinto, por orden serían:

  - Filtros de este objeto visual: Sólo repercutiría sobre la visualización seleccionada y sobre los datos que en ella se han incluido, como filtros sobre un valor máximo o mínimo.
  - Filtros de esta página: Afectaría al conjunto total de visualizaciones y tarjetas de filtros de la propia pestaña, los tipos de filtros serían los mismos que en el anterior ámbito.
  - Filtros de todas las páginas: Como su propio nombre indica el ámbito que le ocupa serían todas las páginas, por lo que afectaría a todo el Cuadro de Mando.

- Panel de los gráficos: Tarjeta desde la cual podemos retocar el gráfico seleccionado, está subdividida en dos partes: la primera sería donde se presentan todas las posibles visualizaciones que puede elegir el usuario; mientras que la segunda está centrada en la personalización de la visualización. En la imagen vemos la primera subpestaña, donde arrastramos los datos que queremos visualizar, la segunda sería personalizar nuestro dibujo, y la última es desde donde se pueden aplicar pequeños análisis a los datos del gráfico y representarlos, como líneas de tendencia o marcadores de media, moda, varianza….

- Panel de selector de datos: En este apartado vemos todos los conjuntos de datos de los que disponemos en nuestro Power BI, la función de esta pestaña es la de seleccionar los datos que queremos para nuestra visualización. Una vez seleccionada la tabla que queremos, el propio programa nos mostrará cada una de las columnas que tiene ese conjunto para que seamos más específicos en nuestra elección de los datos.

47. Imagen del selector de datos

Fuente: Elaboración propia

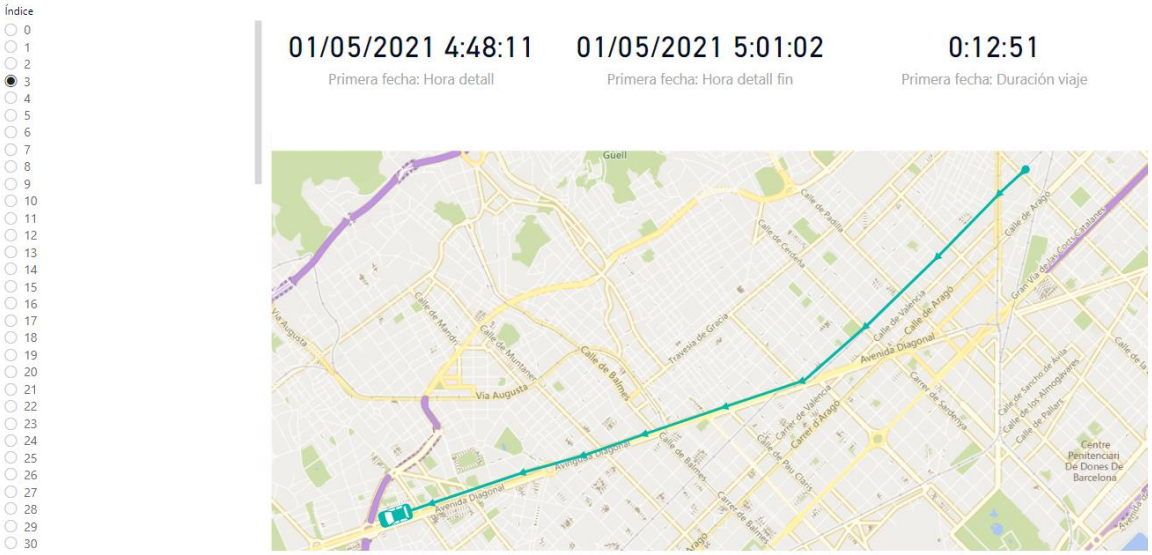
8.5 Visualizaciones

En este apartado nos centraremos en repasar cada una de las representaciones gráficas expuestas a lo largo del Power BI, realizadas con los paquetes de visualizaciones de este programa. No se tratarán aquellas realizadas en código Python, ya que estás últimas serán comentadas en otro apartado.

8.5.1 Route Map.

Con esta visualización queríamos mostrar a un nivel más detallado la ruta o el trayecto por el que circula cada individuo, junto con la fecha a la que inicia y termina el viaje con todo detalle e incluso la duración de este. Por lo que el cliente solo tendrá que seleccionar en el panel de *"Índice"* situado a la izquierda y todos los datos, incluido el mapa, cambiarán automáticamente. Para la elaboración de este mapa se usó la tabla *"tabla_ruta"*.

48. Imagen Route Map



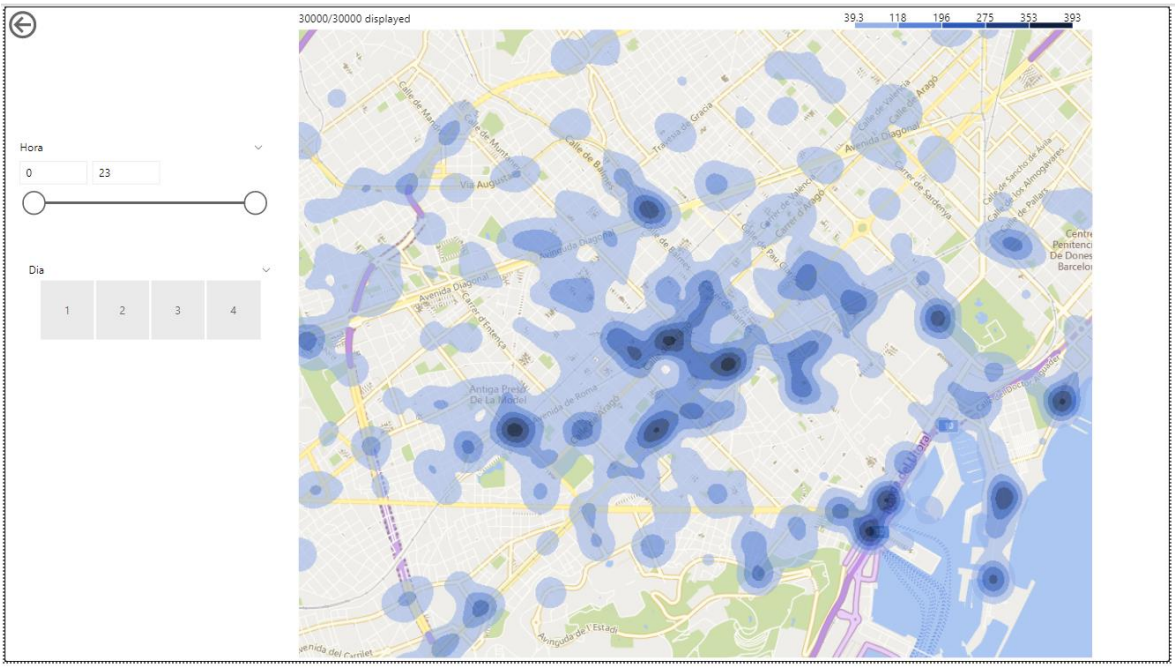
Fuente: Elaboración propia

Hay que destacar la complejidad de la elaboración de esta visualización debido a que el autor que la creó abandono el proyecto y al no haber actualizaciones por su parte el gráfico no funcionaba bien en un comienzo. Al final pudo encontrase una versión actualizada por la comunidad que funcionaba. Esto podría haber sido una razón para que en el caso de que se quisiera haber usado este gráfico y de no haber podido encontrar una versión actualizada haber creado con Python un mapa de este estilo.

8.5.2 Heat Map.

Al igual que en el anterior, se buscaba un punto visual con un mapa, en este se realizó con la misma finalidad, representar por donde pasan los individuos, pero usando otro enfoque. En este caso se usó un mapa de calor, a través del cual, el cliente puede filtrar por horas o por días, y ver en qué localizaciones de Barcelona hay una mayor concentración de motos y bicicletas en todo momento. Para este mapa se usó la tabla *"tabla_ruta"*.

49. Imagen Heat Map

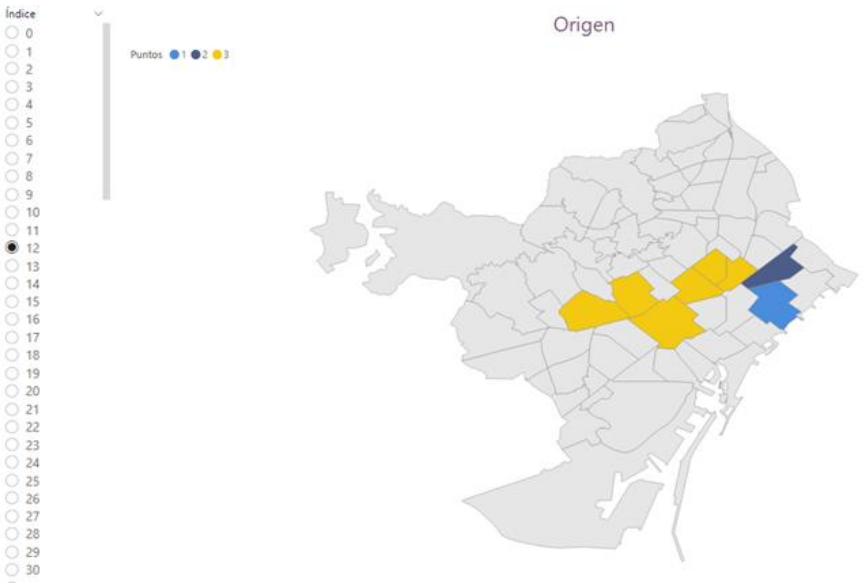


Fuente: Elaboración propia

8.5.3 Form Map.

Continuando con los gráficos de mapas, elaboramos uno enfocado en el que se vean los barrios en los que está situado el individuo en todo momento. Para este mapa se usó la tabla *"tabla_entre_tiempos"*.

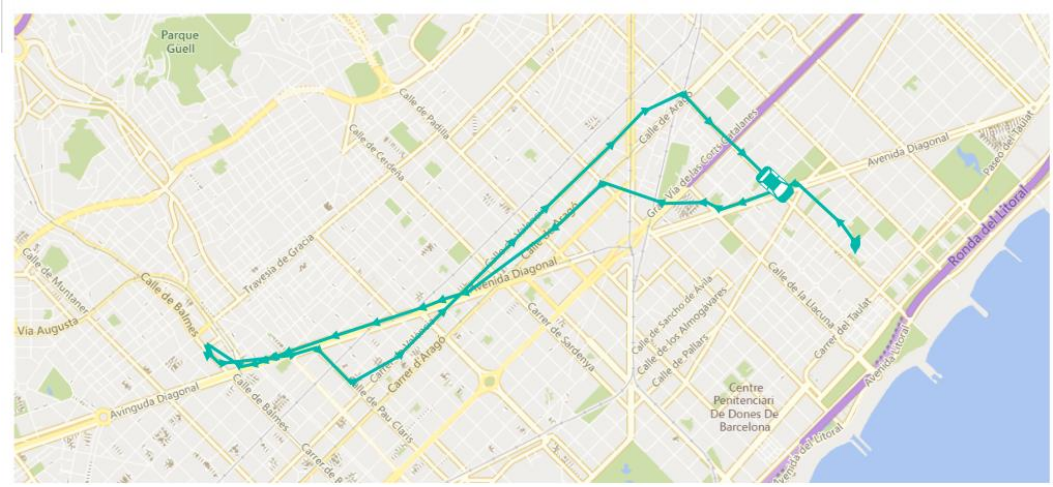
50. Imagen Form Map



Fuente: Elaboración propia

Resaltar que la imagen anterior está puesta con la intención de visibilizar trayectos "extraños" que a simple vista podrían confundir, pero si lo combinamos con el Route Map podremos saber qué es lo que pasa, en este caso se trata de una persona que salió de su barrio llego a otro y dio la vuelta hasta el mismo punto. Como se puede ver a continuación:

51. Imagen dato extraño con Route Map



Fuente: Elaboración propia

8.5.4 Top_N.

Aquí, cambiamos un poco el estilo de la visualización para poder analizar los trayectos más comunes, usando la tabla *"tabla_frecuencias"* de la cual se habló anteriormente sobre su creación.

52. Imagen Tabla Top_N

Unión	Recuento
la Dreta de l'Eixample->la Dreta de l'Eixample	588
Sant Gervasi - Galvany->Sant Gervasi - Galvany	331
Hospitalet->Hospitalet	278
la Dreta de l'Eixample->l'Antiga Esquerra de l'Eixample	273
l'Antiga Esquerra de l'Eixample->la Dreta de l'Eixample	246

Fuente: Elaboración propia

Aquí podemos ver de mejor manera en comparación con la Imagen Tabla frecuencias, donde se puede observar cuáles son las principales conexiones entre barrios, destacando que las dos primeras posiciones son movimientos intra-barrios. La tercera, en cambio, son datos de los cuales en su mayoría van por Hospitalet. Finalmente, las dos últimas posiciones las ocupan los mismos barrios, pero existiendo más viajes entre la Dreta de l'Eixample a l'Antiga Esquerra de l'Example, que en sentido contrario.

Tabla resumen tiempos y periodos. En esta pestaña se puede ver de una mejor manera la gran cantidad de personalización y de información que se puede transmitir con Power BI. Ya que aquí tenemos 3 filtros subdivididos por tipos de empresa, el primero sería el selector del día inicial de viaje de los trayectos y los otros dos se tratará a más a fondo ya que se procedió a la creación de 2 medidas para el dinamismo de la información.

- Duración media del servicio: Como su nombre indica nos dice cuánto dura cada viaje de media en base a la empresa a la que hace referencia y por día, en el caso de que el propio cliente o usuario decida aplicar el filtro. Este indicador se creó de la siguiente forma.

```
Media = FORMAT(AVERAGE(tabla_features[Duración viaje]),"HH:MM: SS")
```

  En la anterior fórmula indicamos a Power BI que haga la media de la columna *"Duración viaje"* de la tabla *"tabla_features"* y con la función *"format"* le decimos que nos exprese el resultado en horas, minutos y segundos.

- Inicio del servicio de media: En este caso, se realizó una moda da la variable *"periodo"* ya que al tratarse de variables categóricas no podía realizarse una media. Se llevo a cabo de la siguiente manera.

```
moda =
var grupos = SUMMARIZE(tabla_features, tabla_features[Periodo], "Total"
, COUNTROWS(tabla_features) )
var maximo = MAXX(grupos , [total])
var modas = filter(grupos, [total] = maximo)
return
CONCATENATEX(modas, tabla_features[Periodo] , "/" )
```

  En la anterior fórmula se procedió a declarar primero la variable "grupos", la cual como su nombre indica, agrupaba cada tipo de periodo en la columna "Periodo". Seguidamente, se creó la variable "máximo" que evaluaba el número de veces que se repetía cada periodo en sus grupos y por último "modas" la cual devolvía el periodo máximo de los distintos máximos por grupos. Por último, se indicó que devolviera el valor de la variable "modas".

53. Imagen Tabla resumen tiempos y periodos

Dia de inicio de viaje

	1		2		3		4	
	Duración media del servicio	Cuando inician el servicio de media		Duración media del servicio	Cuando inician el servicio de media		Duración media del servicio	Cuando inician el servicio de media
Cityscoot	00:12: 04	Tarde	avant.	00:11: 30	Tarde	Donkey Cycles	02:35: 17	Tarde
ridemovi	00:17: 31	Mañana/T...	Tucycle	00:12: 24	Mañana	acciona	00:13: 52	Tarde
IBERSCOT	00:11: 06	Tarde	OIZ	00:11: 33	Tarde	MOVO	00:12: 59	Tarde
YEGO	00:16: 24	Tarde	cooltra	00:12: 24	Tarde	respiro	00:13: 17	Mañana

Fuente: Elaboración propia

8.6 Integración de gráficos de Python en PBI

A lo largo de este apartado nos centraremos en cómo ejecutar gráficos con código Python en Power BI, como debe ser su estructura y la evolución de nuestras visualizaciones, ya que hay que tener en cuenta la novedad que supuso para nosotros ambos programas y el breve tiempo del que dispusimos. En este apartado, trataremos la creación de lo gráficos de Python integrados en Power BI, los errores que se tuvieron y como se relacionan los datos para poder ejecutar los filtros.

Para comenzar a integrar código Python en Power BI primero se debe descargar Python en el equipo local, y posteriormente se deberá habilitar la creación de scripts de Python en el propio Power BI a través de la pestaña de *"Archivos"," Opciones y configuración"*, *"Opciones"* y *"Creación de scripts de Python"*. Seguidamente, se deberá instalar desde el símbolo del sistema *"cmd"* los paquetes esenciales para el manejo y tratamientos de datos *"Pandas"* y visualización *"Matplotlib"*. (Bichiashvili & Olprod, 2020)

Una vez tengamos instalado y habilitado Python se deberá crear una visualización especial en Power BI denominada *"Objeto visual de Python"* a través de la cual, podremos representar la salida gráfica de nuestro código, resaltar lo de salida gráfica

debido a que, si no el propio script no funcionará, ya que las únicas salidas que permite son aquellas en las que el código tiene como resultado un gráfico.

A esta dificultad añadirle otras propias limitaciones dentro de Power BI que el mismo programa tiene implícitas y que el propio usuario deberá tener en cuenta en todo momento para el correcto funcionamiento de las salidas con Python, algunas son:

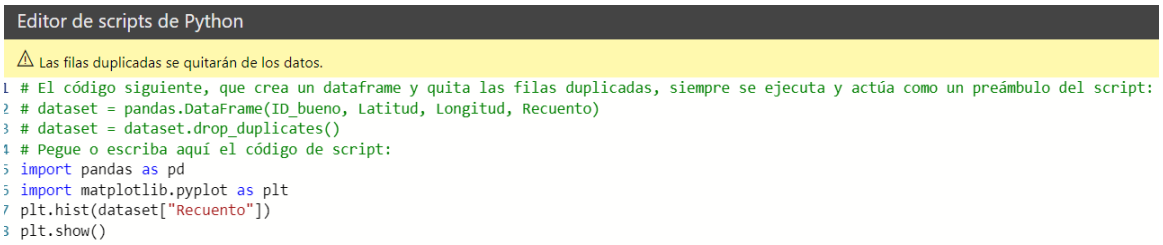
- Duración de ejecución: Los scripts cuyo tiempo de duración sea mayor de 30 minutos no podrán ejecutarse, ya que se agotará el tiempo de espera del propio programa. (Bichiashvili & Olprod, 2020)
- No es posible trabajar con tablas anidadas, es decir, unas dentro de otras. (Bichiashvili & Olprod, 2020)
- La ruta de acceso al directorio de trabajo debe de ser la ruta completa en lugar de la ruta relativa por lo que si tenemos dos carpetas distintas de Python y cada una con librerías distintas, deberíamos tenerlo en cuenta ya que la ruta completa solo elegirá la carpeta seleccionada, únicamente, quitándole flexibilidad al código. (Bichiashvili & Olprod, 2020)

54. Imagen visualización con Python

Fuente: Elaboración propia

La mecánica inicial es la misma que en la de un gráfico de Power BI, seleccionamos el objeto visual de Python y se nos abrirá un cuadro en la pestaña de visualizaciones, después tendremos que arrastrar los datos con los que queramos trabajar. Seguidamente, se nos incorporará en una pestaña que tendremos justamente debajo de la imagen anterior con el siguiente código:

55. Imagen script de Python en PBI



Fuente: Elaboración propia

A continuación, podemos tener un primer vistazo de los cambios con respecto a un script de Python normal, donde todos los datos incorporados se almacenarán en un mismo conjunto de datos. Luego se eliminarán las filas duplicadas por lo que hay que tener cuidado con el tipo de información que seleccionaremos, ya que todo se juntará en un mismo conjunto de datos y esto nos dificultará a la hora de trabajar. No solo estaremos limitados con los datos que incorporaremos, sino que, si en nuestras filas tenemos valores duplicados porque así requerimos, Power BI las eliminará automáticamente, otra dificultad añadida.

```
# El código siguiente, que crea un dataframe y quita las filas duplicadas, siempre se
ejecuta y actúa como un preámbulo del script:
# dataset = pandas.DataFrame(ID_bueno, Latitud, Longitud, Recuento)
# dataset = dataset.drop_duplicates()
# Pegue o escriba aquí el código de script:
import pandas as pd
import matplotlib.pyplot as plt
plt.hist(dataset["Recuento"])
plt.show()
```

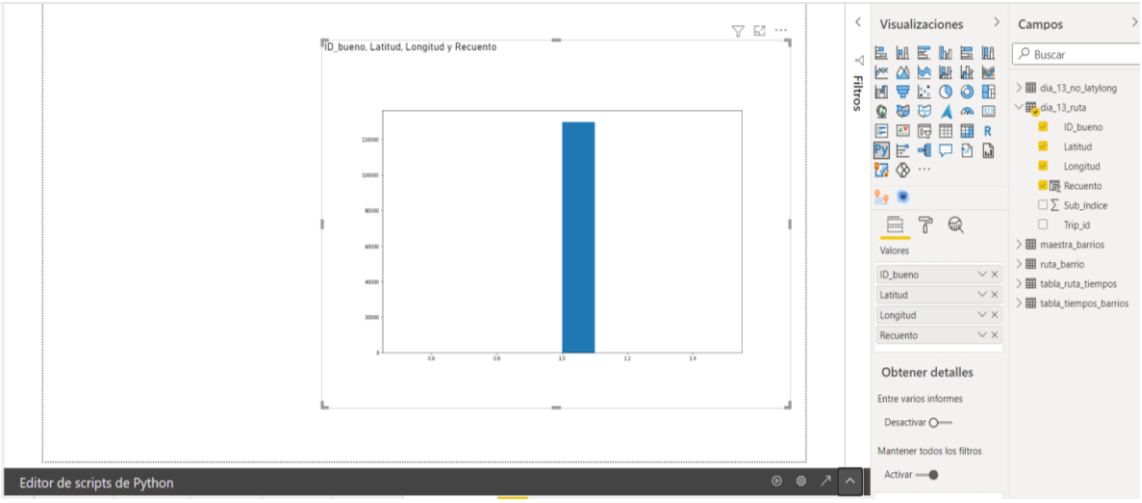
Seguidamente, se deberán importar todos los paquetes que requeriremos al principio, como en un script normal, tras ello podremos ejecutar el código que generará la salida del gráfico que necesitemos, seleccionando los valores de nuestro conjunto previamente creado, ahora con el nombre de "*dataset*". Hay que aclarar que siempre que importemos los datos el nombre de la tabla en la que estaban no se tendrá en cuenta y a partir de ahora se pasará a denominar con el nombre anteriormente indicado. Por último e indispensable, para que se ejecute el código este debe acabar con un *"plt.show()"*, ya que solo aceptará

visualizaciones y gráficos, por lo que salidas como tablas, números o listas quedarán descartados al no ser gráficos como tal.

Esto siempre supondrá una de las mayores dificultades para el usuario, ya que no sabrá si los datos que está manejando son números enteros, decimales, tuplas, listas o sublistas, por lo que sería aconsejable que cualquier script de Python en Power BI fuera acompañado de su homónimo en un notebook de Python. De este modo, se podrá controlar con qué tipo de datos se está trabajando, y aun así no librará al propio usuario de este tipo de errores, ya que hay estructuras con su código correspondiente que en Power BI o no existen o no se pueden trabajar con ese mismo código, como por ejemplo listas anidadas, como veremos más delante.

A continuación, mostramos la salida del anterior código como un ejemplo y también para ilustrar como empezamos a desarrollar código y su salida correspondiente en Power BI, con el fin de que sea visible la evolución del código, su nivel de complejidad y las salidas que este ocasionará.

56. Imagen del 1º gráfico con Python en PBI



Fuente: Elaboración propia

Como puede verse el gráfico es un simple histograma, sin ningún tipo de información relevante que extraer, solo resaltar de esta imagen el resultado de la potencia de combinar ambos programas, ya que hemos construido un gráfico a nuestro gusto y mantenemos el aspecto visual agradable de Power BI, junto con su interfaz y las opciones de filtros con las que podremos cambiar y darles dinamismo a nuestros propios gráficos.

8.7 MCA

8.7.1 Explicación y objetivos:

Con este método queríamos encontrar alguna forma de poder diferenciar los distintos barrios de donde partían los usuarios de las bicis y motos de Barcelona, y así encontrar algún patrón, ya fuera por el periodo (mañana, tarde, noche…) en el que iniciaban el viaje o por el tipo de propulsión.

El método de Análisis de Correspondencias Múltiples fue elegido con ese propósito, ya que permite introducir datos categóricos, en nuestro caso los periodos y el tipo de propulsión, además los datos que los componían eran los barrios asociados a estos.

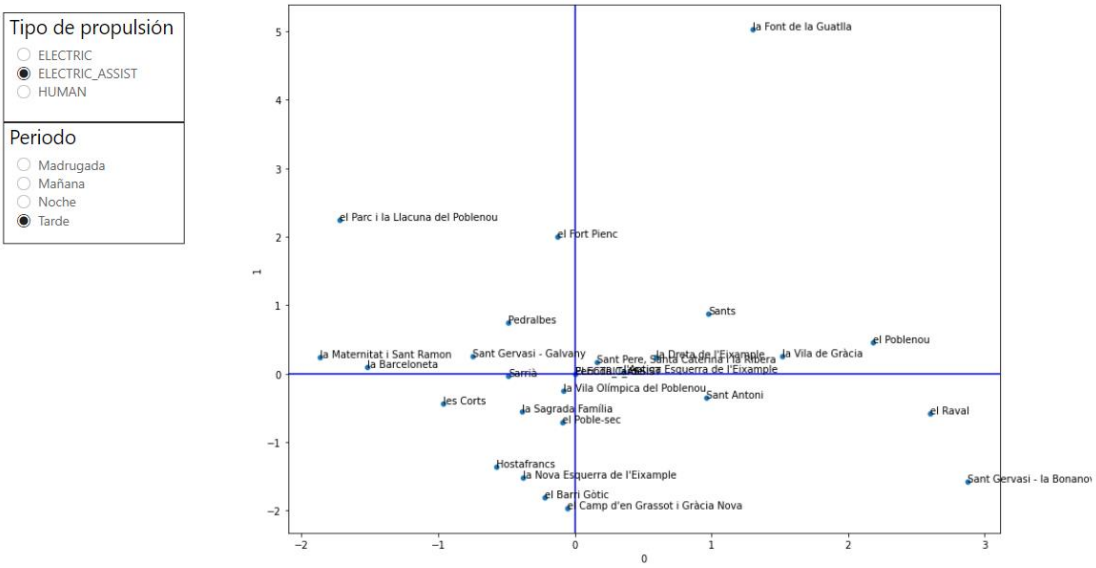
"El Análisis de Correspondencias Múltiples utiliza unos cálculos de ajuste qué recurren esencialmente al álgebra lineal y produce unas representaciones gráficas donde los objetos a describir se transforman en puntos sobre un eje o en un plano". (Enrique Parra Olivares, 1996)

8.7.2 Implementación:

La disposición del gráfico MCA en Power BI no fue del todo complejo, debido a que la creación y la salida gráfica en Python no era muy complicada y a la hora de implementarlo en Power BI no hubo que cambiar muchos aspectos.

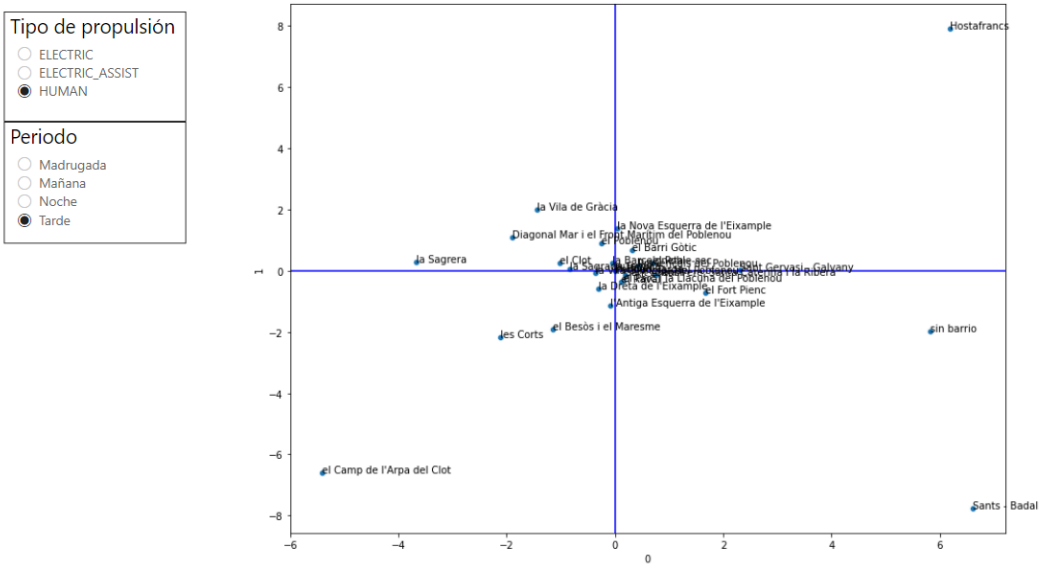
El proceso se dividió en varias etapas, la primera, como siempre, es la importación de todos los paquetes requeridos, entre los que se incluyó *"prince"* y *"seaborn"*, el primero se usaría para la creación del propio MCA, y el segundo para la representación gráfica. Con esto último, nos dimos cuenta que para el funcionamiento de la salida en Power BI no hacía falta que el gráfico estuviera creado con una función del mismo paquete que la función que generaba la salida, es decir, podían mezclarse módulos sin problemas. Seguidamente se generó el modelo usando *"prince.MCA"*, al tener más de dos variables, tres para ser exactos, y todas ellas categóricas, como se indicó en el apartado de explicación y objetivos. El modelo se hizo con el objetivo de reducir la dimensionalidad y de intentar agrupar, o ver alguna relación entre los barrios, y las otras variables categóricas. Para poder discernir mejor las diferencias se optó por aplicar los filtros de Power BI usando tanto el tipo de propulsión y los periodos, obteniendo como resultado final estas salidas.

57. Gráfico MCA en PBI con ELECTRIC_ASSIT



Fuente: Elaboración propia

58. Gráfico MCA en PBI con HUMAN



Fuente: Elaboración propia

En las anteriores imágenes podemos ver claramente un comportamiento distinto entre barrios dependiendo de los filtros aplicados. En el primero, cuando el tipo de propulsión es *"asistido"* y periodo es la *"tarde"* vemos más separación entre barrios, mientras que si cambiamos a un tipo de propulsión "*humano"* hay una mayor agrupación de los barrios.

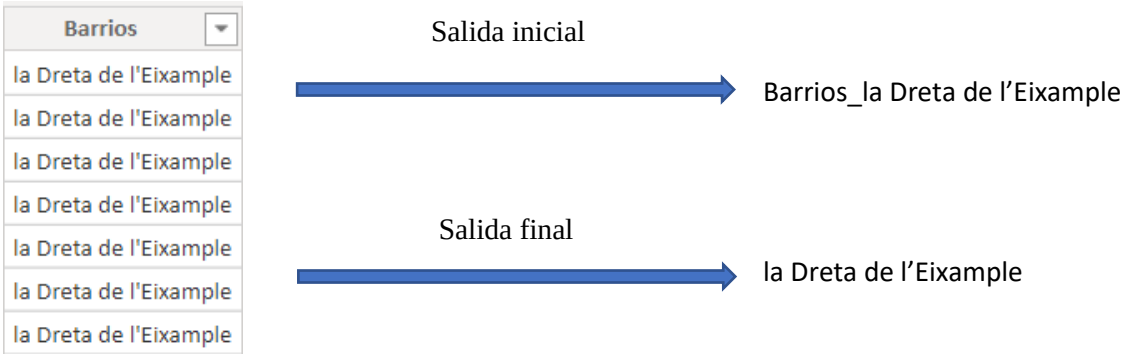
Con el fin de ilustrar mejor lo que podría hacerse con el MCA y a lo que podría deberse, pondremos un ejemplo de un posible comportamiento, esto podría deberse a que la cantidad de viajes que hay entre los puntos es diferente. En la última imagen, cuando se usan bicis por la tarde, hay una mayor comunicación entre esos barrios, mientras que en la otra imagen sería la suposición contraria. Estas conclusiones no pudieron ser afirmadas ya que los ejes explicaban un porcentaje muy bajo de la variación, debido a que este método al reducir la dimensionalidad de tres variables a dos pierde información, que en algunos casos es asumible, pero en este caso no.

Por lo que este estudio podría ser interesante para futuros trabajos en los que se tuviera una mayor cantidad de datos a través de los cuales sacar alguna característica diferencial, que pueda adaptarse mejor la oferta a la demanda de los viajes.

8.7.3 Errores:

En este gráfico apenas tuvimos errores, solo nombrar uno que sí podrían ser interesante a nivel visual. Este sería la limpieza de los nombres de los barrios con el fin de que a la hora de representar el gráfico los nombres de los barrios incorporaban el nombre de la columna, por lo que la longitud de cada título de los puntos era muy grande, teniendo como resultado una peor visualización y la superposición de los nombres.

59. Imagen error del nombre



Fuente: Elaboración propia

8.8 GRAFO

8.8.1 Objetivos:

El objetivo principal es poder replicar el grafo que se ha conseguido obtener en Python utilizando la extensión que tiene Power BI de este lenguaje. Con esta extensión se permite obtener distintas visualizaciones dentro de la propia herramienta Power BI, pero utilizando exclusivamente Python. El grafico que queremos replicar sería este:

60. Gráfico grafo diferenciado

Fuente: Elaboración propia

8.8.2 Errores

Como se comentó más arriba, ejecutar código Python en Power BI no siempre funciona igual debido a que Power BI no interpreta los tipos de datos igual que Python, dificultando la tarea. Entre los errores a los que tuvimos que enfrentarnos están:

- Pintar grafico de Barcelona. En primer lugar, para poder hacer el dibujo necesitamos los datos de las coordenadas de los barrios de Barcelona, unos datos de los que disponemos en formato CSV. Los datos fueron cargados correctamente.

  61. Imagen Tabla barrios antes del tratamiento

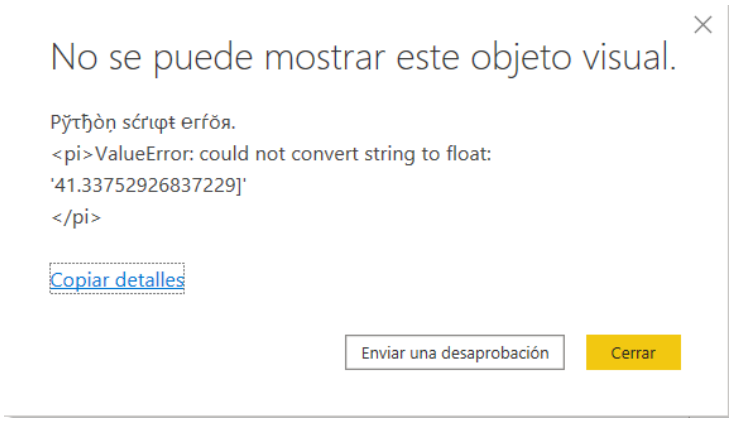
Barrio	coords
el Raval	[[2.164713785844972, 41.38593019854664], [2.1693583444407
el Barri Gòtic	[[2.1770141884632963, 41.38524835715523], [2.178734943517
la Barceloneta	[[2.196228824385047, 41.3874542218253], [2.19631396013653
la Dreta de l'Eixample	[[2.170908900255276, 41.40181726974143], [2.1722060725489
l'Antiga Esquerra de l'Eixample	[[2.1573553158476955, 41.39330621402394], [2.158470643683
la Nova Esquerra de l'Eixample	[[2.1498381792247425, 41.37538785628039], [2.149766712937

  Fuente: Elaboración propia

  Como se puede observar en la columna *"coords"* aparecen las coordenadas de los barrios las cuales deben ser tratadas, ya que aparecen dos corchetes, y ambos programas entienden estos datos como cadenas de texto, no como listas. Este problema se tuvo que resolver en Python primero, programando un código que fuera capaz de separar las diferentes coordenadas por pares, y que finalmente nos creara una lista para poder trabajar con ella.

  A la hora de replicar el código en Power BI comenzaron a surgir diferentes errores, uno de los más llamativos era que la extensión de Python en Power BI no era capaz de leer las coordenadas de cada barrio. El error que aparecía nos indicaba que no era capaz de convertir una cadena de texto en número.

62. Imagen salida de error en PBI 1



Fuente: Elaboración propia

Para intentar solucionar el problema se volvieron a cargar los datos y se hicieron varias transformaciones de estos desde el ejecutable de Python que tiene Power Query. Con estas transformaciones pudimos obtener un conjunto de datos que se pareciera al deseado y el cual pudiera ser ejecutable.

63. Imagen ejecutable Python en Power Query

Fuente: Elaboración propia

Mediante esta transformación se logra convertir la cadena de texto en una lista, ya que, para Python, los datos que se encuentran ahora dentro de la columna *"coords"*

son una lista, y dentro de la lista cada coordenada será tomada como un número decimal, esto es lo que ocurre en Python.

64. Imagen Tabla barrios después del tratamiento

Barrio	coords
el Raval	[(2.164713785844972, 41.38593019854664), (2.16935834444077, 41....
el Barri Gòtic	[(2.1770141884632963, 41.38524835715523), (2.178734943517667, ...
la Barceloneta	[(2.196228824385047, 41.3874542218253), (2.196313960136539, 41....
la Dreta de l'Eixample	[(2.170908900255276, 41.40181726974143), (2.1722060725489283, ...
l'Antiga Esquerra de l'Eixample	[(2.1573553158476955, 41.39330621402394), (2.1584706436839887,...
la Nova Esquerra de l'Eixample	[(2.1498381792247425, 41.37538785628039), (2.1497667129372506,...

Fuente: elaboración propia datos de Power BI.

Como podemos observar, después del tratamiento, desaparece uno de los corchetes que existía en el conjunto de datos. Al volver a intentar ejecutar el código para generar el mapa el error persistía, el programa no era capaz de convertir las cadenas de texto a número decimal, con lo cual se decidió volver a tratar los datos desde el ejecutable de Python en Power BI. Con este tratamiento se volvieron a separar las coordenadas para poder volver a guardarlas con una estructura algo más simple, para que se pudiera ejecutar en Power BI, ya que Power BI sigue tomando estos datos como cadenas de texto, por eso, aunque usemos Python y este pueda leer los datos como una lista, los datos en Power BI siguen sin ser decimales, y no se puede dibujar el mapa.

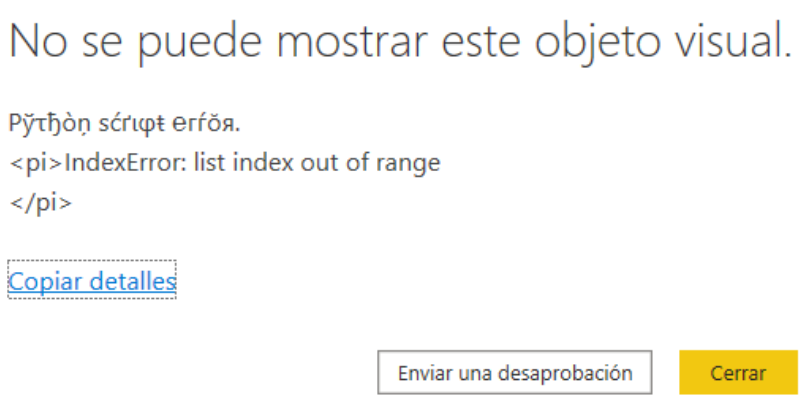
65. Imagen código para el tratamiento en PBI

```
lon=list()
for k in range(len(barrios)):
    lat=list()
    for i in barrios["coords"][k].split("),"):
        coor=i.replace("[","").replace(" ","").replace("]","").replace("(","").replace(")","").replace("\n","").split(",")
        lat.append((float(coor[0]),float(coor[1])))
    lon.append((barrios["Barrio"][k],lat))
```

Fuente: elaboración propia datos de Power BI.

Tras realizar un nuevo tratamiento apareció un nuevo error *"list index out of range"*, ante este problema, no se tenía muy claro qué hacer, ya que el código ejecutaba perfectamente en Python, pero no cesaba de dar fallos en Power BI.

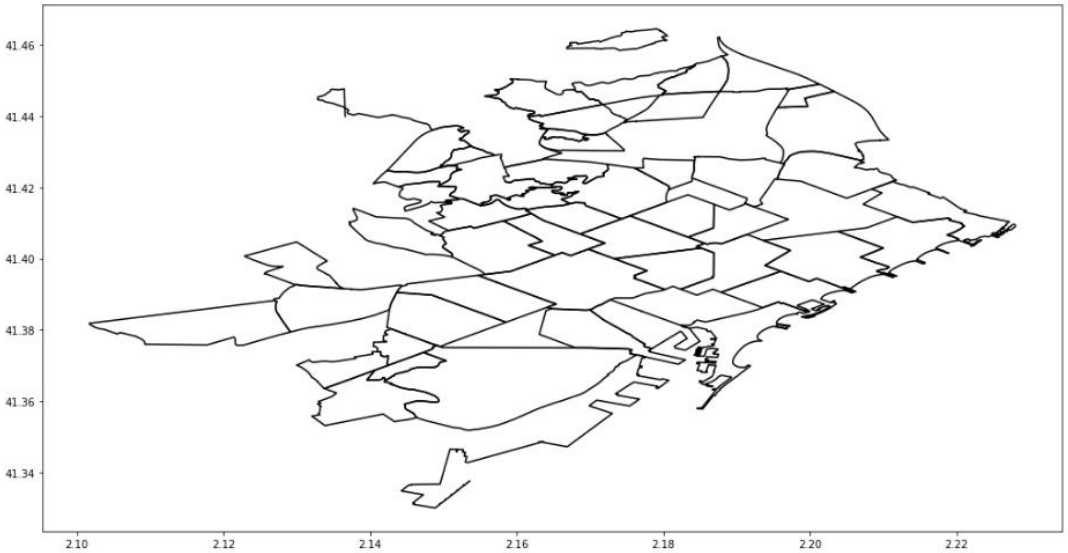
66. Imagen salida de error en PBI 2



Fuente: elaboración propia datos de Power BI.

Se decidió no coger todos los datos, sino las 10 primeras filas, para así poder detectar y entender de una forma más sencilla el por qué fallaba. Al hacer esto y ejecutar el código, de repente se pintaron los 10 primeros barrios. Se hizo una prueba con los 20 siguientes y con los 30 hasta llegar al barrio numero 54 (“el Camp de l'Arpa del Clot”). A partir de ese punto se dejaba de pintar el mapa y saltaba el error. El mapa que obtuvimos era este:

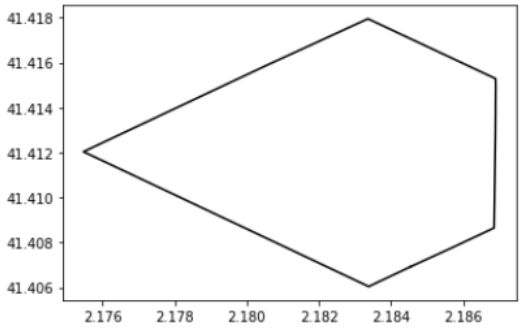
67. Imagen grafo incompleto



Fuente: elaboración propia

Los barrios situados más al norte no se podían dibujar. Ante este fallo se decidió extraer los datos del primer barrio que fallaba, para intentar dibujarlo en Python. Y en Python, se dibujó sin problema.

68. Imagen barrio el "Camp de l'Arpa del Clot" en Python



Fuente: elaboración propia

69. Gráfico barrio el "Camp de l'Arpa del Clot"

Fuente: Google Maps

Ante la imposibilidad de resolver este error se tuvo que cambiar de estrategia. Para ello se dividieron las coordenadas de los barrios 2 a 2, tanto en latitud y longitud con el fin de delimitarlo, y se asoció cada coordenada a su barrio. El objetivo es poder acceder a los datos para así dibujarlos evitando el problema anterior. Una vez creado el conjunto de datos, este quedó del siguiente modo.

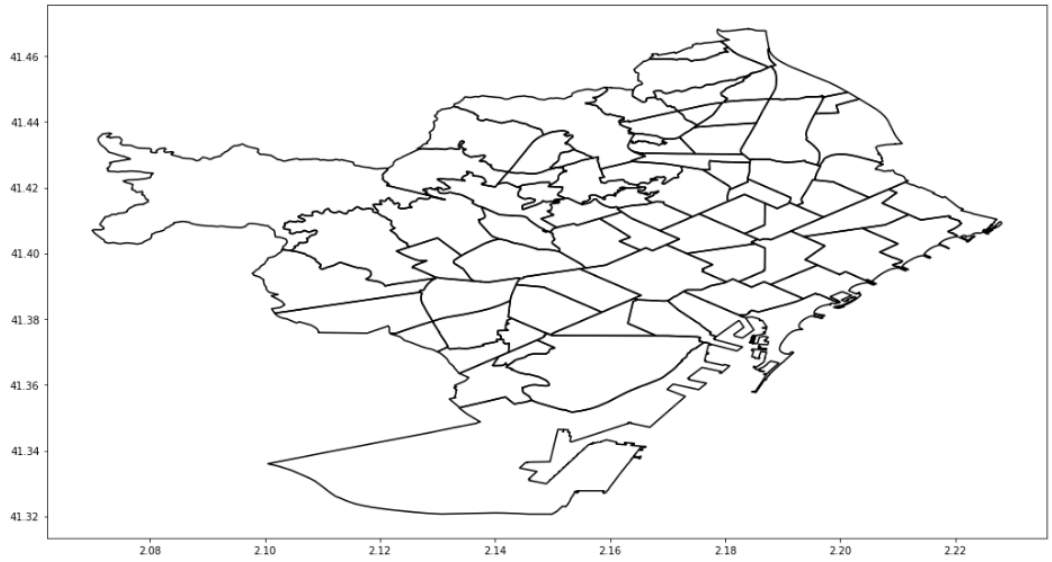
70. Imagen Tabla latitud, longitud y orden

1.2 lat	1.2 lon	1²3 Orden	ABC Barrio
41,32068288	2,149857306	3763	la Marina del Prat Vermell
41,32074056	2,149519866	3766	la Marina del Prat Vermell
41,32074361	2,149620003	3765	la Marina del Prat Vermell
41,32075092	2,145741528	3771	la Marina del Prat Vermell
41,32075425	2,146859509	3768	la Marina del Prat Vermell
41,32075759	2,149728407	3764	la Marina del Prat Vermell
41,32075768	2,14771162	3767	la Marina del Prat Vermell

Fuente: elaboración propia

Con estos datos finalmente se pudo crear el mapa de Barcelona, uniendo las diferentes coordenadas de cada barrio, quedando el grafico del siguiente modo:

71. Imagen mapa Barcelona completo



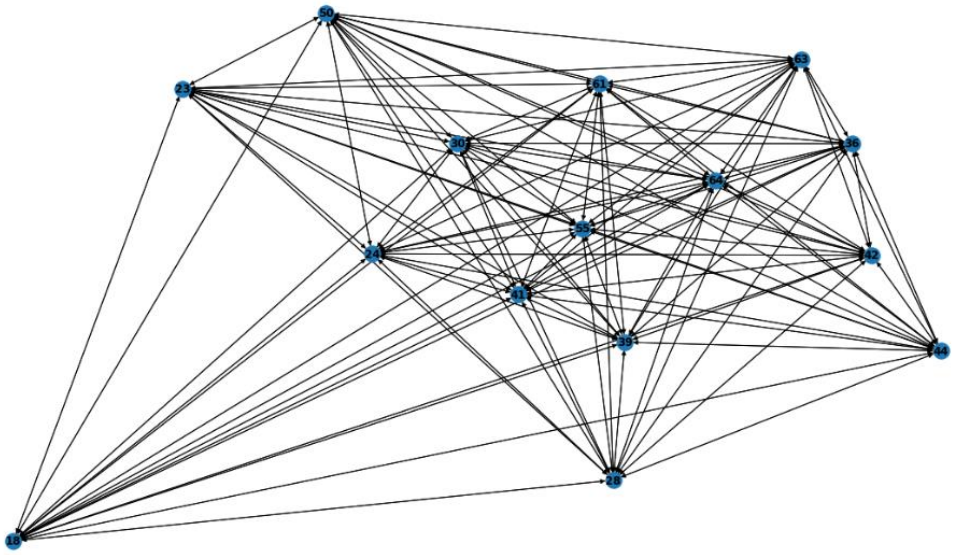
Fuente: elaboración propia desde Power BI.

- Problema de dimensionalidad hasta posteriormente importar los "*csv*". Al resolver este problema, se planteó cómo crear un grafo y poder dibujar los diferentes nodos sobre el mapa, pero en primer lugar había que probar a crear un grafo y comprobar si el código, que estaba escrito en Python, era capaz de ejecutarse en Power BI. Para ello, usamos la matriz creada en Python, la cual hace un recuento del número de viajes que se realiza de barrio a barrio y aparte, se cargó otro conjunto de datos en el cual están los centroides de cada barrio.

Este último conjunto de datos se usaría para poder representar cada nodo del grafo en el mapa. Aquí tenemos que añadir que, la dimensión por filas de la matriz con el número de viajes coincidía con la dimensión por filas de los centroides, y en Power BI es importante, ya que crea un único conjunto de datos con toda la información, aunque provengan de diferentes orígenes.

Si tienen la misma dimensión, se puede evitar un fallo a la hora de ejecutar el código desde Python. Ambos conjuntos de datos se relacionan por el nombre del barrio, este es un apunte esencial, debido a que, si no, no sería posible relacionar ambas tablas. Finalmente se pudo crear el grafo, quedando del siguiente modo:

72. Imagen grafo en PBI



Fuente: elaboración propia desde Power BI

La finalidad de la obtención de cada grafico por separado es poder aprender cómo funciona el programa internamente cuando se usa el lenguaje Python con elementos más simples, para así poder detectar los posibles fallos de manera más rápida y eficiente, sin complicar el entendimiento cuando se junten ambas partes.

Una vez creados ambos gráficos la tarea era juntarlos para poder representar la misma figura que teníamos en Python, representar el mapa de Barcelona y encima

poder representar el grafo que se obtiene, para que el usuario pueda extraer información de manera inmediata. Pero a la hora de juntar ambos elementos comienzan a surgir multitud de problemas, el primero es la dimensión de cada conjunto de datos utilizado. La matriz para crear el grafo tiene una dimensión de 67×67, el conjunto de datos para poder representar los datos de Barcelona tiene una dimensión de 27478 filas, y la tabla con los centroides tiene una dimensión de 67 filas.

El mayor reto supuso entender cómo solventar el problema de la dimensionalidad, ya que, al agrupar todos los conjuntos de datos, estos se relacionan mediante barrio y se crea un conjunto de datos de tamaño 27.478 filas, lo cual no se puede ejecutar, porque comienza a fallar a la hora de generar el grafo, ya que este tiene 67 filas. Se intento segmentar todo el conjunto de datos en tres diferentes, pero la extensión de Python en PBI tiene serias limitaciones, ya que únicamente te muestra objetos gráficos, no es posible observar que tipo de datos se están creando o que es lo que se puede ejecutar y lo que no. Es una programación a ciegas, con una ligera intuición del mensaje de error cuando salta. Con todas estas limitaciones no se podía saber muy bien si el código estaba generando tres conjuntos de datos diferentes, porque al ejecutarlo en Python, si daba resultado, pero en Power BI no.

Finalmente se decidió intentar leer archivos *"csv"* desde la extensión de Python en Power BI, lo cual dio resultados positivos y funcionó. La forma más sencilla en la que se pudo ejecutar el código fue creando un único conjunto de datos utilizando la matriz con el número de viajes y los centroides de cada barrio. Por otro lado, se incorporó un archivo *"csv"* con las coordenadas de los barrios para poder dibujar el mapa de Barcelona. Con el conjunto de datos que tenía la matriz y los centroides se logró hacer una separación de estos, por un lado, se creó la matriz eliminando del *"dataset"* completo las latitudes y longitudes de los centroides y por otro se logró crear una lista con las latitudes y longitudes de cada barrio, para poder dibujar la posición en la que estaría cada nodo del grafo. El resultado final fue el siguiente:

73. Gráfico grafo Barcelona PBI

Fuente: elaboración propia usando Power BI.

Como se puede observar el resultado en cuanto a datos es el mismo. Las relaciones entre nodos se representan del mismo modo que en Python, dibujando unas líneas más anchas en función de si el número de viajes es mayor entre cada nodo. También se ha podido plasmar el tamaño del nodo, este es más grande si la suma de viajes por filas (origen-destino) es mayor. Quedaría mejorar ciertos aspectos formales, aunque es complicado ya que surgen errores y no es posible representar el grafico a la hora de incluir los nombres de cada barrio.

- Matriz para el grafo, diagonal principal. En la construcción del grafo tuvimos que generar una matriz con los barrios, desde donde iniciaban el viaje cada usuario, en las filas y los barrios donde terminaban los viajes, por columnas. Con eso lograr todas y cada una de las posibles combinaciones entre barrios para así poder crear la visualización y que uniera los distintos nodos (uno por barrio) con los demás, pero por temas de visualización, a fin de no sobrecargar el propio grafo, decidimos quitar los movimientos intra-barrios, ya que esa información la representaríamos a través de Power BI.

74. Imagen matriz final

	el Raval	el Besòs i el Maresme	la Nova Esquerra de l'Eixample	la Marina de Port	la Dreta de l'Eixample	el Clot	les Corts	Vallcarca i Penite
el Raval	42.0	2.0	29.0	5.0	72.0	5.0	9.0	
el Besòs i el Maresme	0.0	17.0	0.0	0.0	12.0	5.0	3.0	
la Nova Esquerra de l'Eixample	39.0	1.0	189.0	12.0	188.0	8.0	82.0	
la Marina de Port	3.0	0.0	15.0	34.0	15.0	1.0	6.0	
la Dreta de l'Eixample	63.0	10.0	203.0	6.0	588.0	25.0	73.0	
...	...	...	...	...	...	...	...	
Sant Genís dels Agudells	0.0	0.0	1.0	0.0	1.0	0.0	0.0	
Verdun	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
Cap Pequera	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

Fuente: Elaboración propia

Para solucionarlo, decidimos realizar un "bucle for" en el cual, en cada iteración, revisaría si la fila y la columna coincidían, de ser así le asignaría un "0" a esa posición correspondiente en el caso contrario dejaría el valor que ocupase esa posición. El código usado fue el siguiente:

```
> Para cada fila "i":
>       Para cada columna "k":
>             Si i = k:
>                   mejores(i,k) = 0
>             Sino:
>                   mejores(i,k) = matrices(i,k)
```

Para el desarrollo de este bucle la principal dificultad fue la de seleccionar los valores dentro de "matrices". En este código vemos como primero se delimitan el número de iteraciones del bucle con "selección", que será una cantidad que se haya elegido previamente, luego el bucle detectará si la fila y la columna son iguales, pero si no es así, tendrá que asignarle el valor correspondiente. Esto se hizo creando la lista "position", con las posiciones de los "x" mejores barrios seleccionados.

8.8.3 Grafo en Power BI utilizando Tabla Features

En este apartado se va a volver a crear el mismo grafo que en el apartado anterior, pero esta vez no se usara la matriz final, ya que con esta matriz no se mantiene la granularidad de los datos. Es decir, los datos que aparecen en la matriz no están dentro del modelo relacional y no se pueden aplicar directamente filtros sobre el grafo. Al utilizar tabla features, en el modelo relacional como "tabla madre" la cual conecta con el resto de las tablas, esto permite poder hacer un filtrado de datos más exhaustivo, en función de variables que están relacionadas con ella como tipo de propulsión, tipo de vehículo, empresa, periodo del día en que se usan los vehículos, barrios… Para poder crear una matriz igual que la matriz final hay que pivotar la tabla features, pero en primer lugar hay que agrupar los datos de tabla features en función de los barrios de inicio y fin.

75. Imagen Tabla features en Power BI

TRIP_ID	Hora detall	Hora detall fin	propulsion_type	tipo_vehiculo	empresa	Barrios	Barrios_fin
18868160	*01/05/2021 18:04:31*	*01/05/2021 18:12:35*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	el Baix Guinardó
18868226	*01/05/2021 18:10:00*	*01/05/2021 18:26:58*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	la Dreta de l'Eixample
18868195	*01/05/2021 18:07:20*	*01/05/2021 18:48:59*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	la Dreta de l'Eixample
18868451	*01/05/2021 18:25:32*	*01/05/2021 18:32:19*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	la Vila de Gràcia
18868264	*01/05/2021 18:13:04*	*01/05/2021 18:21:43*	ELECTRIC	MBK50	COOLTRA	la Dreta de l'Eixample	el Camp d'en Grassot i Gràcia Nova

Fuente: Elaboración propia

76. Imagen Tabla features agrupada en función de los barrios iniciales y finales

	Barrio_ini	Barrio_fin	Viajes
0	Baró de Viver	Vallcarca i els Penitents	1
1	Can Baró	Can Baró	1
2	Can Baró	Pedralbes	2
3	Can Baró	Sant Gervasi - la Bonanova	1
4	Can Baró	Sarrià	2
...	...	...	...
2544	les Tres Torres	la Salut	5
2545	les Tres Torres	la Vall d'Hebron	3
2546	les Tres Torres	la Vila de Gràcia	18
2547	les Tres Torres	les Corts	17
2548	les Tres Torres	les Tres Torres	41

Fuente: Elaboración propia

Una vez que se elabora la tabla agrupada toca pivotar esta tabla, hay que poner los valores de barrio inicial en función de los barrios y que se agreguen los valores de los viajes a cada elemento de la matriz que se genere.

77. Imagen Matriz final pivotada

Barrio_fin	Can Baró	Can Peguera	Diagonal Mar i el Front Marítim del Poblenou	Horta	Hospitalet
Baró de Viver	0.0	0.0	0.0	0.0	0.0
Can Baró	1.0	0.0	0.0	0.0	0.0
Can Peguera	0.0	1.0	0.0	0.0	0.0
Diagonal Mar i el Front Marítim del Poblenou	0.0	0.0	39.0	1.0	1.0
Horta	1.0	0.0	0.0	29.0	1.0
...	...	...	...	...	...

Fuente: Elaboración propia

La dimensión de la matriz final pivotada que se obtiene es de 66 × 66, es decir, se pierde una fila y una columna. Para entender mejor que elementos se habían eliminado, se ejecutó el mismo código en el Notebook de Python, y se comprobó que en las filas de la matriz desaparece el barrio "les Roquetes", mientras que en las columnas desaparece "Baró de Viver". Ante este problema, se comprobó que estos barrios no creaban ninguna combinación. Es decir, al hacer una suma de los elementos de la matriz final creada en Python que tenía 67 barrios, "les Roquetes" da cero, y al hacer la suma por columnas ocurre lo mismo con "Baró de Viver". Esto ocurre porque estos barrios no tienen ninguna combinación con otros en origen y en destino y por eso la función *"groupby"* los omite.

78. Imagen Tabla Suma por filas de la matriz final sin pivotar

les Roquetes	0.0
Baró de Viver	1.0
Verdun	2.0
Can Peguera	3.0
la Teixonera	4.0

Fuente: elaboración propia

79. Imagen Tabla Suma por columnas de la matriz sin pivotar

Baró de Viver	0.0
les Roquetes	1.0
Verdun	2.0
la Teixonera	2.0

Fuente: elaboración propia

Una vez comprobado que los datos estaban bien, tanto en la matriz final pivotada como sin pivotar, había que cargar los datos de las coordenadas que delimitan los barrios y también los centroides de cada barrio. Estas dos tablas se importaron directamente como un *"csv"* desde el script de Python en Power BI. Después de cargar ambos *"csv"* como archivos independientes simplemente se hicieron unas transformaciones en las coordenadas de los centroides ya que estos no se leían como números decimales sino como una cadena de texto. Además, dentro del propio código se ejecutó una opción para que se mostrara el Top N de barrios que un usuario desease, esto evitaría saturar el grafo de información y que la ejecución sea más rápida. Para lograr esto se recalculaba la matriz final en función del número de barrios especificado. El Top N de barrios se obtiene de la suma por filas de la matriz final y su ordenación es de mayor a menor, es decir, en función de los barrios que más flujo tienen a los que menos. En esta matriz que se genera se elimina la diagonal principal, ya que es una información que no quedará representada en el grafo y se puede representar de otros modos como un Heat Map.

80. Imagen Matriz pivotada con el Top 4 y con ceros en la diagonal

	la Dreta de l'Eixample	l'Antiga Esquerra de l'Eixample	Sant Gervasi - Galvany	la Nova Esquerra de l'Eixample
la Dreta de l'Eixample	0.0	273.0	211.0	203.0
l'Antiga Esquerra de l'Eixample	246.0	0.0	150.0	137.0
Sant Gervasi - Galvany	200.0	148.0	0.0	87.0
la Nova Esquerra de l'Eixample	188.0	128.0	106.0	0.0

Fuente: elaboración propia

Finalmente, el grafo que se obtiene es el siguiente:

81. Imagen grafo Barcelona en PBI final

Fuente: elaboración propia

A este grafo, se le pueden aplicar diferentes filtros como, por ejemplo, tipo de vehículo=MBK150 Y distrito= L`Eixample quedando así:

82. Imagen grafo Barcelona en PBI final filtrada

Fuente: elaboración propia

8.9 HEATMAP

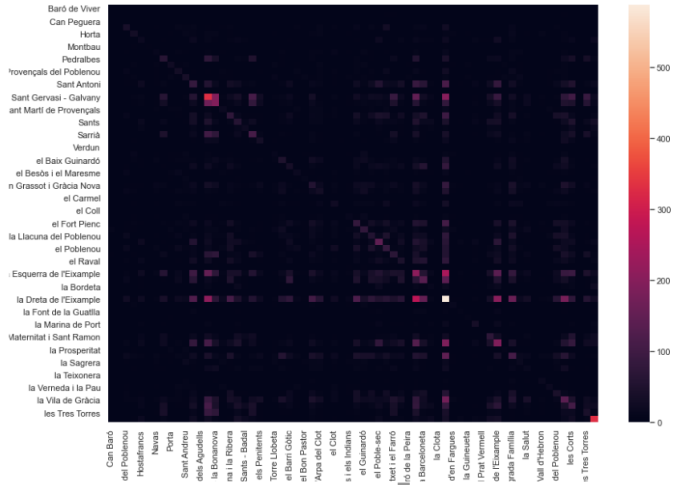
8.9.1 Explicación y objetivos:

Un mapa de calor es una técnica de visualización que permite representar la magnitud de los datos en función de una escala de colores. Con esta técnica se puede saber en este caso las interacciones más fuertes que se producen entre los diferentes barrios. Además, es una forma de medir el tráfico intra-barrios, es decir, de un barrio consigo mismo, ya que en el grafo esta información se ha eliminado.

8.9.2 Implementación:

El mapa de calor de Power BI se genera a través del script de Python. Para poder programarlo y que además los filtros de Power BI se apliquen al mapa se procede a la creación de la matriz final, pero esta vez utilizando una función *"groupby"* la cual permite agrupar los elementos de la tabla features y posteriormente se pivotaron los datos para poder ordenarlos en forma de matriz.

83. Imagen Heat Map en PBI



Fuente: elaboración propia

Aunque ha desaparecido un barrio diferente por cada fila y columna, se puede observar como la diagonal del gráfico aparece bastante bien marcada. Esta no es la diagonal principal, porque está desplazada, pero sí que refleja los movimientos más frecuentes, que son los movimientos entre los mismos barrios, es decir, la gente suele empezar su viaje en un barrio y el destino más frecuente suele ser el mismo barrio.

9. Conclusiones

Atendiendo a los objetivos teóricos del trabajo se ha logrado en primer lugar la explicación de lo que es un proceso de ETL, con sus diferentes fases que son carga, extracción y limpieza de datos. Además, se ha puesto de manifiesto la importancia de este proceso a la hora de extraer conocimiento a través de los datos.

En cuanto al desarrollo teórico de Python, se ha hecho una breve introducción de su creación junto con su objetivo original, también se ha conseguido explicar la importancia y la versatilidad que tiene la programación orientada a objetos.

Respecto a los objetivos prácticos, se ha logrado entender el funcionamiento y la estructura de los datos en formato Json y GeoJson, con la posterior carga de estos en Python, con su respectiva limpieza. Se logró crear todas las tablas para recopilar la información de forma ordenada y estructurada, información que va a nutrir nuestro modelo relacional en Power BI.

Se logró analizar, mediante un grafo, el tráfico de Barcelona, lo cual permitió extraer información sobre cuáles son los barrios más transitados, como *La Dreta de l'Eixample* o la zona centro de la ciudad. Además. se logró incorporar dinamismo al grafo en base a ciertos filtros como el horario o periodo del día, el tipo de vehículos que se usan, tipo de propulsión…. Cabe añadir que se consiguió una fusión entre la salida del grafo en código Python junto con los filtros de distintas tablas del modelo relacional de Power BI.

Finalmente, y como objetivo a destacar, se ha logrado en poco tiempo dominar ambas herramientas, logrando una sinergia entre ellas con el fin de explotar los puntos fuertes de cada una. Se ha utilizado un formato de datos que nunca habíamos usado en el máster, que además eran reales, sin simplificar para uso didáctico, con todos los problemas que eso conlleva y que hemos explicado en el presente informe. Esto supuso un reto de aprendizaje ya que, a pesar de que desde la propia empresa esta fusión entre programas

era un objetivo perseguido desde años atrás, no se había abordado aún y no se disponía de conocimiento previo, por lo que queremos destacar el aprendizaje autónomo, como equipo, que hemos tenido que llevar a cabo. Esto asienta las bases para futuros trabajos o proyectos que lleven un paso más allá el trabajo que aquí se ha realizado.

10. Bibliografía

arcGIS. (2021). *ArcGIS Enterprise*. Obtenido de GeoJSON: https://enterprise.arcgis.com/es/portal/10.8/use/geojson.htm

B12, A. (9 de julio de 2021). *Agencia B12*. Obtenido de ¿Qué son los procesos ETL?: https://agenciab12.com/noticia/que-son-procesos-etl

Bichiashvili, O., & Olprod. (02 de 06 de 2020). *docs.microsoft.com*. Obtenido de https://docs.microsoft.com/es-es/power-bi/connect-data/desktop-python-scripts

Burrueco, D. (s.f.). *interactivechaos.com*. Obtenido de https://interactivechaos.com/es/manual/tutorial-de-power-bi/historia-de-power-bi

Challenger-Pérez, I., Díaz-Ricardo, Y., & Becerra-García, R. A. (2014). El lenguaje de programación Python. *Ciencias Holguín, XX (2)* , 7.

Challenger-Pérez, I., Díaz-Ricardo, Y., & Becerra-García, R. A. (2014). El lenguaje de programación Python . *Ciencias Holguín, XX (2)*, 6.

Chazallet, S. (2016). *Python 3. Los fundamentos del lenguaje - 2ª edición.* Eni.

cognodata. (7 de mayo de 2019). *cognodata*. Obtenido de Procesos ETL: cómo obtener valor de los datos: https://www.cognodata.com/blog/procesos-etl/

Diaz, A. I. (2012). *Calculadora Geoespacial.* Buenos Aires: Universidad Tecnológica Nacional Facultad Regional Bahía Blanca.

ECMA, 4. (2017). The JSON Data Interchange Syntax. *ecma International*, 1-5.

Enrique Parra Olivares, J. (1996). Modelo de análisis de correspondencias múltiples. *Revista de Ciencias Sociales*, 183-196.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1997). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 41.

García, D. (25 de 02 de 2021). *https://www.bitec.es/*. Obtenido de https://www.bitec.es/herramientas-bi/power-bi-otro-ano-como-lider-en-el-cuadrante-magico-de-gartner/

Guerrero, J. (13 de noviembre de 2012). *El Blog de José Guerrero*. Obtenido de Distancia entre dos puntos de la superficie terrestre mediante la fórmula de Haversine con Python: https://joseguerreroa.wordpress.com/2012/11/13/distancia-entre-dos-puntos-de-la-superficie-terrestre-mediante-la-formula-de-haversine-con-python/

Iseminger, D. (11 de 01 de 2020). *docs.microsoft.com*. Obtenido de https://docs.microsoft.com/es-es/power-bi/transform-model/desktop-query-overview

Llopis, M., & olprod. (27 de 07 de 2020). *microsoft.com*. Obtenido de https://docs.microsoft.com/es-es/power-query/power-query-what-is-power-query

Microsoft. (s.f.). *support.microsoft.com*. Obtenido de https://support.microsoft.com/es-es/office/acerca-de-power-query-en-excel-7104fbee-9e62-4cb9-a02e-5bfb1a6c536a

Programiz. (s.f.). *Programiz.com*. Obtenido de https://www.programiz.com/cpp-programming/examples/print-sentence

Ruiz Borja, J. E. (2018). *Comparación de herramientas ETL de.* Medellin: Facultad de Minas.

Sandoval Linares, A. G. (2018). *Análisis de métodos y técnicas de Limpieza de Datos existentes y aplicación en un Sistema CRM para una institución educativa.* Lima.

Statista. (17 de abril de 2019). *https://es.statista.com/grafico/17734/cantidad-real-y-prevista-de-datos-generados-en-todo-el-mundo/*. Obtenido de https://es.statista.com/grafico/17734/cantidad-real-y-prevista-de-datos-generados-en-todo-el-mundo/

Tulchak, L. V., & Marchuk, A. O. (2016). *History of Python.*

Anexo

Partes realizadas por cada miembro, siendo:

- A=Ayhan
- R=Roberto

ETL: A

Marco teórico:

- Python: R
- Power BI: R
- Integración de Python y Power BI: R

Desarrollo en Python:

- Carga y tratamiento de datos con Python: A
- Qué son los archivos JSON y como se cargan: A
- Cómo se cargan los datos y que información tenían : A
- Errores después de la carga de datos: A
- Soluciones de los errores después de la carga de datos: A
- Centroides y formación de los barrios: A
- Carga de barrios: A
- Creación de Polígonos: A
- Creación de tabla Puntos: A
- Creación de tabla Coordenadas: A
- Creación de tabla Centroides : A
- Actualización de tabla Features: A
- Matrices: A
- Creación de la matriz final: A
- Creación de la matriz de flujos netos : A
- Creación de la matriz de distancias: A
- Creación de grafo en Python: R

Desarrollo en Power BI

- Carga de datos: R
- Tratamiento y limpieza de los datos: R
- Relación entre tablas y creación del modelo relacional: R
- Creación de visuales en PBI: R
- Visualizaciones: R
- Route Map: R
- Heat Map: R
- Form Map.: R
- Top_N: R
- Integración de gráficos de Python en PBI: R
- MCA: R
- Explicación y objetivos:R
- Implementación: R
- Errores: R
- GRAFO
- Objetivos: A
- Errores A
- Grafo en Power BI utilizando Tabla Features: A
- HEATMAP: A
- Explicación y objetivos: A
- Implementación: A