UNIVERSIDAD COMPLUTENSE DE MADRID

ELECTRIC ROUTE

Trabajo Fin de Máster
Universidad Complutense de Madrid

Febrero 2021

Blanca Alonso González
Marta García Palomo
Irene Robles Cabezas
Lucía Tomaino De la Cruz

Contenido

1. Introducción y Objetivos

Durante los últimos años, la movilidad se ha convertido en uno de los principales problemas que siguen generando altos índices de contaminación. Alcanzar, en este aspecto, espacios más sostenibles y limpios es el gran reto para todos los países comprometidos en la lucha contra el cambio climático. Una de las medidas para lograr este objetivo ha sido impulsar la movilidad sostenible. En noviembre 2020, Sara Aagesen, secretaria de Estado de Energía, informaba, en una comparecencia en la Comisión de Transición Ecológica y Reto Demográfico del Congreso de los Diputados, que el Gobierno ha incluido dentro del plan de recuperación que ha remitido a la Unión Europea una partida de 1.100 millones destinados a la movilidad eléctrica y que podría activarse ya en 2021.

La principal motivación de este trabajo persigue sumarse a ese cambio y fomentar el uso de los coches eléctricos. En los últimos años se han incrementado las ventas de estos vehículos, aunque aún existe el hándicap de las rutas de largo recorrido, producido por la autonomía de los coches. De esta necesidad, surge el principal objetivo del trabajo, desarrollar un recomendador para trayectos de larga distancia según la autonomía de los coches eléctricos, *Electric Route*.

Electric Route es una aplicación web que, dados dos puntos de la Península Ibérica, origen y destino, y, seleccionado un tipo de coche eléctrico, recomendará una ruta óptima en tiempo para ese recorrido. A la hora de determinar la ruta se tendrá en cuenta la autonomía del coche, de manera que si el viaje está limitado por la batería del coche se le recomendará al usuario un punto de recarga en medio del recorrido.

Además, en caso de no existir un punto de recarga conveniente para el coche eléctrico en cuestión, debido a tener un tipo de conector no compatible, se utilizará una red que combina los puntos de recarga reales con puntos de recarga ficticios, situados en estaciones de servicio. Esta recomendación proporciona localizaciones donde sería interesante instalar nuevos puntos de recarga.

1.1. Planificación y estructura del proyecto

En el desarrollo del trabajo se utiliza un repositorio para control de versiones dónde se puede consultar la documentación, los diferentes conjuntos de datos utilizados, el código de todo el desarrollo del proyecto, el despliegue de la aplicación web Electric Route, y la productivización de la misma con la herramienta Docker. Se puede acceder a toda esta información en el link [GitHub].

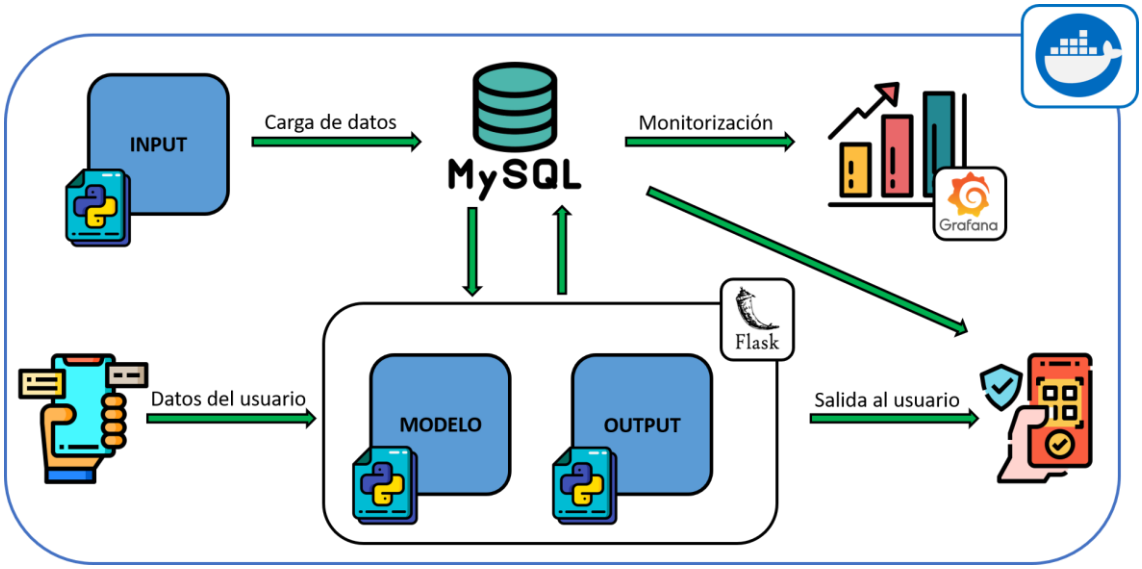
La planificación del proyecto ha estado estructurada por tiempos y entregables que se han ido marcando a lo largo de los meses del desarrollo del trabajo fin de máster. Para una mayor organización se ha trabajado con un fichero Excel en el que se han ido marcando las tareas pendientes, el tiempo estimado para realizarlas y la asignación a cada participante del trabajo. Este calendario se encuentra disponible en el repositorio de GitHub. Cronológicamente el esquema de trabajo ha sido el siguiente, aunque algunas tareas se han solapado en el tiempo:

- **Elección del trabajo fin de máster**: tras varias reuniones y brainstorming sobre distintos posibles temas se decide la simulación de rutas para coches eléctricos como tema principal del proyecto. Para tomar la decisión, se realiza un estudio en internet sobre la disponibilidad de los datos que se van a utilizar y se determinan las distintas herramientas y tecnologías a emplear.
- **Estado del arte**: antes de comenzar a trabajar en el problema, se dedica un tiempo a conocer más sobre el tema y familiarizarse con algunos conceptos.
- **Planteamiento del problema**: una vez decidido el tema se profundiza en detectar el target del problema, así como conseguir todos los conjuntos de datos con los que se desea trabajar.
- **Modelización**: una de las etapas más largas y complejas. Se comienza con un análisis exploratorio de los datos (EDA) para cada conjunto. Una vez conocidos los datos de los que se dispone y sus características, se plantea el problema como la modelización de una red compleja dirigida. Mediante el uso de la librería *networkx* y la definición tanto de la función objetivo (tiempo total) y de las restricciones del problema, se obtiene la ruta óptima por el algoritmo de A* para los

parámetros fijados (tipo de estacionamiento, origen, destino, marca y modelo de coche y carga inicial y final de la batería).

- **Despliegue herramienta web**: en esta parte del trabajo, se definen tareas como definir la estructura de la aplicación o el diseño de las diferentes plantillas, así como las distintas funcionalidades que presenta: sign-up, log-in, conexión con la base de datos, implementación del modelo, etc.
- **Productivizar el modelo**: quizás la parte que más reto ha supuesto técnicamente. Para desplegar de manera automática en un entorno operacional la aplicación se han utilizan dos tecnologías distintas: Docker/Docker-Compose y Google Cloud Platform, por lo que se ha trabajado en comprender e implementar una solución basada en ellas.
- **Grafana**: a pesar de que el objetivo principal, en lo que a interfaz gráfica se refiere, era el desarrollo de la aplicación web, una vez evolucionado el proyecto se considera que las bases de datos tienen mucha información relevante. Por este motivo, se decide mostrar al usuario todo este detalle mediante unos dashboard elaborados en Grafana.
- **Documentación**: la parte final del proyecto queda reservada a realizar esta memoria resumen de todo lo trabajado y realizar la presentación y el vídeo que se entrega como material complementario.

Respecto a la estructura o el flujo de trabajo se describe gráficamente en el siguiente esquema:

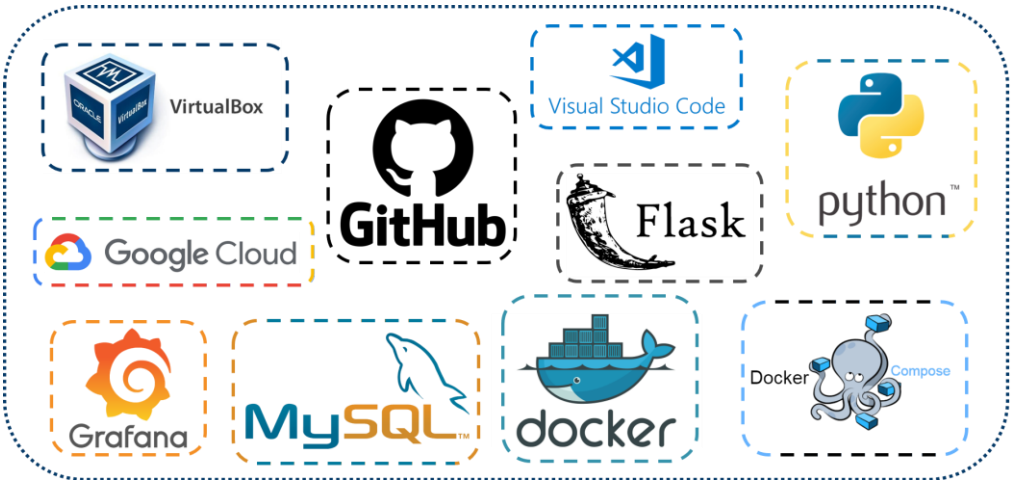


1.2. Herramientas Software

Para el desarrollo de trabajo se utilizan diferentes softwares que se explican brevemente a continuación:

- **GitHub**: es una herramienta web para la gestión de repositorios basada en Git. Permite almacenar y gestionar código, además de llevar un control de versiones durante el desarrollo del trabajo. Se utiliza como repositorio del proyecto completo, incluyendo los conjuntos de datos, los scripts utilizados para la obtención y depuración de los datos, el desarrollo del modelo, el despliegue de la web y la creación de contenedores de aplicaciones, así como la documentación del proyecto (tanto a nivel de planificación como de algoritmos).
- **VirtualBox**: es un software de virtualización que permite el uso de máquinas virtuales. Se utiliza como herramienta para el desarrollo de la aplicación en local. El sistema operativo usado ha sido Ubuntu 18.04.

- **Visual Studio Code**: es un IDE que permite programar, depurar y probar código para distintos lenguajes de programación. Se utiliza como entorno básico para desarrollar el código fuente de la aplicación en local.
- **MySQL**: es un sistema de gestión de bases de datos relacional. Se utiliza para gestionar las diferentes bases de datos con las que se trabaja.
- **Python**: lenguaje de programación elegido.
- **Flask**: es un microframework para Python que permite crear aplicaciones web. Para el desarrollo de la aplicación web se hace uso de Flask, incluyendo otros lenguajes de programación necesarios como HTML, JavaScript o CSS.
- **Grafana**: es un software libre basado en licencia de Apache, que permite la visualización y el formato de datos métricos. Se utiliza para realizar análisis exploratorio de datos y monitorización del sistema.
- **Docker**: es una herramienta, basada en un proyecto de código abierto, que permite el despliegue de aplicaciones dentro de contenedores software, facilitando la automatización de virtualización de aplicaciones en diversos sistemas operativos.
- **Docker-Compose**: es una herramienta, utilizada como complemento a Docker, que permite definir y ejecutar aplicaciones Docker multicontenedor y simplificar así su uso a partir de archivos YAML. De esta forma resulta más sencillo crear contenedores que se relacionen entre sí, conectarlos y habilitar puertos y volúmenes.
- **Google Cloud Platform**: es una plataforma de computación en la nube que ofrece diversos servicios. Se utiliza para productivizar y desplegar el sistema de la aplicación en un entorno remoto.



Además de estos softwares se hace uso de algunas APIs de Google o de Tomtom para obtener información. Estas aplicaciones son las siguientes:

- **Google Maps API:** Se utiliza para obtener datos geográficos de distinta naturaleza. Los servicios usados son:
  - **Geocoding Service**: Se utiliza para obtener las coordenadas geográficas de las localizaciones de Origen y Destino, seleccionadas de antemano en estaciones de autobús y de tren de las ciudades principales de la Península Ibérica.
  - **Places Service**: Se utiliza para obtener la información geográfica de los puntos de recarga para vehículos eléctricos en las distintas Comunidades Autónomas de la Península Ibérica.
  - **Distance Matrix Service**: Se utiliza para obtener las distancias entre los distintos puntos (teniendo en cuenta puntos de origen y destino, puntos de recarga y gasolineras).

- **Tomtom – EV Charging Stations Availability:** Se utiliza para obtener información específica de los puntos de recarga como el número y tipo de conectores y la potencia suministrada por cada conector.

2. Comprensión de los datos

El conjunto de datos con el que se trabaja proviene de diversas fuentes y se vuelca en distintas tablas de la base de datos. Una de ellas incluye información relativa a los coches eléctricos; otra, recoge los distintos puntos de recarga existentes en la Península Ibérica y, tal y como se ha introducido en los objetivos, se obtiene información referente a las estaciones de servicio. Además, durante el planteamiento del problema, surge una nueva tabla que contiene datos de interés sobre las ciudades de la península.

2.1. Coches eléctricos

El listado de coches eléctricos se encuentra publicado en la página de Kaggle, [EVs – One Electric Vehicle Dataset – Smaller]. Se compone de dos ficheros: *ElectricCarData_Clean.csv* y *ElectricCarData_Norm.csv*. En el primero, las variables numéricas contienen las unidades para cada observación mientras que en el segundo se han eliminado y renombrado las variables. Se utilizará, por tanto, el fichero: *ElectricCarData_Clean.csv.*

El análisis exploratorio de este conjunto de datos (EDA) se basa en la comprobación de la correcta tipología y rol de las variables, un análisis descriptivo básico de las variables numéricas, dónde se detecta que a priori no existen valores ausentes o datos erróneos. Además, se estudia la representatividad de las variables categóricas y los diferentes valores que toman las categorías de cada una de ellas. En esta etapa, se detectan cinco valores missing del dataframe, los cuales, pertenecen en su totalidad a la variable *Fastcharge.* Debido a que no se trata de un número elevado, se realiza una búsqueda sobre los registros que presentan el valor ausente para completar la información. Finalmente, se observa que esos coches no disponen de carga rápida y, por ese motivo, se imputan los missing con valor cero.

A continuación, se corrigen los errores de escritura, eliminando los caracteres especiales y pasando a mayúsculas todos los nombres de las variables y valores de las mismas. Por último, se procede a descartar las variables irrelevantes para el objetivo de este trabajo y se crea la variable *Battery_capacity* a partir de variables ya existentes, la cual resultará imprescindible de cara a las restricciones del modelo.

El conjunto de datos definitivo tras la limpieza consta de un total de 103 observaciones y 8 variables que se describen a continuación:

- *Brand*: marca del coche. Tipo de dato: string.
- *Model*: modelo del coche. Tipo de dato: string.
- *Range_km*: autonomía del coche en km. Tipo de dato: int
- *Efficiency_whkm*: consumo de energía del coche por cada 100 km. Tipo de dato: int
- *Fastcharge_kmh*: velocidad de carga rápida aproximada en km/h. Tipo de dato: float
- *Plugtype*: tipo de conector compatible con el coche. Tipo de dato: string
- *Battery_capacity*: variable creada como producto de la autonomía del coche y el consumo del mismo. Tipo de dato: float

2.2. Ciudades

De cara a definir los puntos de origen y destino para las distintas rutas que pueda seleccionar el usuario, se decide construir una tabla que contenga las capitales de provincia, tomando como puntos de referencia las estaciones de tren y autobús.

Una vez elaborada la lista de ciudades, se acude a la Geocoding API de Google. Este servicio permite geolocalizar las coordenadas geográficas de cada punto a partir de una descripción basada en texto sobre una ubicación.

Este conjunto de datos tiene un total de 92 observaciones y 6 variables que son:

- *id*: código único asociado a cada punto. Está formado por la ciudad y el tipo de estación del que se trate: tren o bus. Tipo de dato: string.
- *Provincia*: ciudad a la que corresponde el punto. Tipo de dato: string.
- *Dirección*: indica si la estación es de autobuses o de tren. Tipo de dato: string.
- *Latitud*: latitud del punto. Tipo de dato: float64.
- *Longitud*: longitud del punto. Tipo de dato: float64.
- *Coordenadas*: tupla que contiene los valores de la latitud y la longitud. Tipo de dato: object.

2.3. Puntos de recarga

A la hora de obtener los distintos puntos de recarga, en un principio, se utilizó el servicio Places de la API de Google Maps con la intención de extraer toda la información disponible. Sin embargo, el resultado no fue satisfactorio, ya que entre la información no se encontraba ni el número ni el tipo de conector ni la potencia de cada punto. Dada la importancia de dichas variables en el presente trabajo, se indagaron otras opciones que permitieran obtener los datos sin incurrir en un coste elevado. Finalmente, se pudo obtener dicha información a través de la API de Tomtom, la cual no era tan intuitiva como la de Google y obligó a desarrollar un script más complejo.

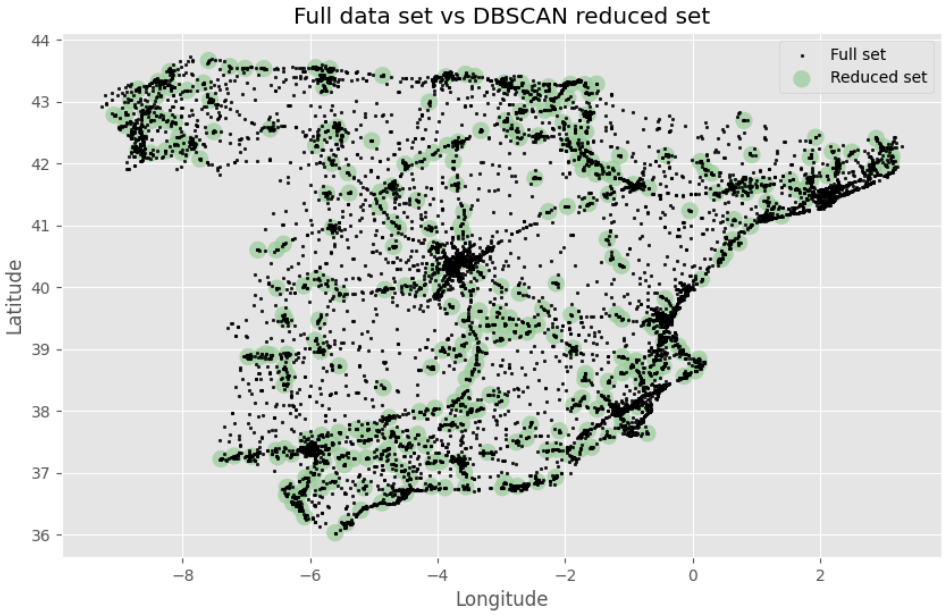
El conjunto de datos definitivo tiene un total de 351 observaciones y 11 variables que se describen a continuación:

- *id*: código único asociado a cada punto de recarga. Tipo de dato: string.
- *Latitude*: latitud del punto de recarga. Tipo de dato: numeric.
- *Longitude*: longitud del punto de recarga. Tipo de dato: numeric.
- *Name*: nombre del punto de recarga (suministradora o municipio). Tipo de dato: string.
- *StreetName*: dirección (ubicación) del punto de recarga. Tipo de dato: string.
- *Provincia*: provincia de ubicación del punto de recarga. Tipo de dato: string.
- *CCAA*: comunidad autónoma de ubicación del punto de recarga. Tipo de dato: string.
- *PostalCode*: código postal de ubicación del punto de recarga. Tipo de dato: numeric.
- *ConnectorType*: tipo de conector proporcionado por el punto de recarga. Tipo de dato: string.
- *RatedPowerKW*: potencia suministrada por el punto de recarga. Tipo de dato: string.
- *Num_connectors*: número de conectores instalados en el punto de recarga. Tipo de dato: numeric.

2.4. Estaciones de servicio

El conjunto de datos de las gasolineras de España se ha obtenido de la página web de [Hub ArcGis]. Dado que el dataset contiene un número de registros muy elevado (10.279), se ha filtrado para determinar las estaciones que más interés pueden tener a la hora de desplegar nuevos puntos de recarga para vehículos eléctricos.

Para llevar a cabo este filtrado, se ha utilizado un algoritmo de Machine Learning no supervisado tipo Clustering [DBSCAN]. Este modelo permite reducir la dimensión de un conjunto de datos basándose en su distribución geográfica, buscando mantener un subconjunto de puntos que representan de manera uniforme el conjunto total.



De esta manera, se obtiene un conjunto de datos que tiene un total de 331 observaciones y 8 variables que se describen a continuación:

- *id*: código único asociado a cada gasolinera. Tipo de dato: string.
- *Provincia*: provincia a la que pertenece la gasolinera. Tipo de dato: string.
- *Municipio*: municipio al que pertenece. Tipo de dato: string.
- *Localidad*: localidad a la que pertenece. Tipo de dato: string.
- *Código_postal*: código postal de la gasolinera. Tipo de dato: float64.
- *Dirección*: dirección de la gasolinera. Tipo de dato: string.
- *Longitud*: longitud de la gasolinera. Tipo de dato: float64.
- *Latitud*: latitud de la gasolinera. Tipo de dato: float64.

3. Modelo: definición y restricciones

El núcleo de la aplicación se basa en el cálculo de rutas óptimas para vehículos eléctricos, para el cual se ha planteado una red compleja dirigida donde los nodos son los puntos de origen/destino y los puntos de recarga/estaciones de servicio propuestas y las aristas tienen un peso igual al tiempo que implica unir dos nodos. Una vez hecho esto, se aborda el problema del camino más corto.

Los parámetros de entrada que necesita el modelo para funcionar son:

- *tipo de estacionamiento*: donde realizar las paradas, puntos de recarga o estaciones de servicio
- *origen y destino*
- *marca y modelo de coche*
- *carga inicial y final de la batería*

Para describir de manera exhaustiva las distintas facetas del modelo, se abordan por separado los aspectos más relevantes de este mismo.

3.1. Definición de la función objetivo y restricciones

El primer paso para plantear el problema como un grafo con nodos y aristas es definir tanto la función objetivo que hay que minimizar (esto es, el peso de las aristas) como las restricciones que hay que aplicar

a la red para que el problema no alcance soluciones no válidas físicamente. A continuación, se describen en profundidad estos aspectos.

3.1.1. Función objetivo

La función objetivo tiene en cuenta tanto el tiempo empleado en desplazarse de un nodo a otro, así como el tiempo estimado en la parada.

Tiempo del trayecto: se calculará como el cociente entre la distancia entre nodos y una velocidad media que se define como constante (100 km/h). Hay que tener en cuenta que la distancia calculada entre dos nodos equivale a la obtenida con la función de Haversine (ver apartado 3.2), por lo que no se tiene en cuenta la distancia por carretera. Esto implica por lo general una subestimación del tiempo del trayecto, por lo que se aplica un factor corrector del 120% para compensar este hecho.

Tiempo de parada: se obtendrá como resultado de la suma entre el tiempo de carga del coche y una estimación del tiempo de espera en la electrolinera.

- **Tiempo de carga:** es el tiempo que tarda en cargar el coche. Se calcula como el cociente entre la capacidad de la batería del coche (energía que es capaz de acumular medida en kWh) y la potencia que le proporcione el punto de recarga donde efectúa la parada (medida en kW). Además, se tiene en cuenta un rendimiento de carga del 90%, por lo que el valor de tiempo de carga obtenido se divide por este coeficiente.
- **Tiempo de espera**: se trata del tiempo que el usuario ha de esperar en la cola de la electrolinera para cargar su coche. Uno de los obstáculos con los que se ha tenido que lidiar en el cálculo de la variable tiempo de espera ha sido la escasa disponibilidad de datos reales relacionados con la afluencia de coches eléctricos en los puntos de recarga.
  Con el objetivo de abordar este problema bajo una cierta coherencia estadística, se ha adaptado un modelo basado en la teoría de colas. No obstante, debido al inconveniente mencionado, se han tenido que asignar valores ficticios a algunas de las variables que intervienen en el modelo en cuestión. El modelo empleado se detalla en el Anexo.

La formulación matemática de lo explicado es la mostrada a continuación:

$$\min t = \frac{\sum_{i=1}^{k-1} d_i}{V_{me}} + t_{parada}$$

Donde:

d_i = distancia entre nodos
k=número de paradas
V_{me} = velocidad media del coche (fijada a 100 km/h)
t_{parada} = tiempo de parada

3.1.2. Restricciones

Se restringe la red sobre la que se trabaja mediante diversas restricciones en desigualdad diseñadas para cumplir requisitos, a veces imprescindibles, de cara al correcto funcionamiento del problema.

Restricciones de obligatoriedad

- **Restricción autonomía**: La distancia entre un nodo y el consecutivo ha de ser siempre menor o igual que el 90% de la autonomía del coche, puesto que si no sería imposible el cálculo de la ruta. Se establece un margen de al menos el 10% de la batería para evitar que, en ningún caso, la batería del coche pueda descargarse por completo antes de llegar a la electrolinera.

$$d_{i,i+1} \leq 0.9 * autonomía \; \forall_i$$

Donde:

$d_{i,i+1}$= distancia entre dos nodos consecutivos de la red
$autonomía$= autonomía del coche
$\forall_i$ = para todo nodo comprendido en la red

- **Restricción de tipo de conector**: Para un coche dado (modelo y marca), hay uno o varios tipos de conectores permitidos. Por este motivo, los puntos de recarga seleccionados han de cumplir con la condición de tener un conector del tipo correcto para el coche en cuestión, por lo que se desestiman aquellos puntos de recarga de la red que no cuentan con un tipo de conector compatible.

Restricciones de no obligatoriedad

Este tipo de restricciones son implementadas como parte de un modelo más avanzado que el propuesto en base a las restricciones obligatorias, las cuales, permitirán ajustar el cálculo de las rutas a las necesidades específicas de cada usuario.

- **Restricción primera parada**: La distancia entre el nodo origen y el siguiente nodo ha de ser menor o igual que la distancia que puede recorrer el coche entre esos dos puntos en función de la carga del coche al principio del trayecto. *Debe recordarse que, por defecto, el coche siempre llegará al menos con un 10% de batería al punto de recarga.*

$$d_1 \leq (0.9 * autonomía) * \frac{C_i}{100}$$

Donde:

d_1 = distancia entre el origen y la primera parada
$autonomía$= autonomía del coche
C_i = carga inicial del coche

- **Restricción última parada**: La distancia entre el nodo destino y el nodo inmediatamente anterior ha de ser menor o igual que la distancia que se permita recorrer en función de la carga final con la que desea llegar el usuario a su destino.

$$d_{k+1} \leq \frac{(100 - C_f)}{100} * (0.9 * autonomía)$$

Donde:

d_{k+1}= distancia entre la última parada y el destino
$autonomía$= autonomía del coche
C_f= carga con la que finaliza el trayecto el coche

3.2. Cálculo de la matriz de distancias

En el momento de plantear el problema como una red compleja compuesta por nodos y aristas (distancias/tiempo), surge la necesidad de calcular una matriz de distancias entre todos los puntos de la red, esto es, ciudades, puntos de recarga y estaciones de servicio.

Primero, se hacen pruebas con la API Google Distance Matrix y se consiguen calcular las distancias y tiempos de recorrido entre las ciudades con los puntos de recarga y las estaciones de servicio. Sin embargo, al ser necesario calcular estas distancias entre todos los elementos de la red, esto supone un gran número de combinaciones y un elevado coste de las llamadas a la API. Se decide entonces calcular esta matriz con ayuda de la distancia de Haversine.

Se definen por tanto dos funciones, una que toma como argumentos las coordenadas de dos puntos y, aplicando la distancia de Haversine devuelve la distancia entre ellos. Y, una segunda función, que

construye iteraciones entre dos filas consecutivas de un dataframe, en este caso se devuelve con la función *zip* una tupla, clave-valor, con el valor del dataframe sobre el que se itera. Finalmente, dado un dataframe que contiene la información sobre todos los puntos de la red, se recorre mediante dos bucles para calcular la distancia.

El dataframe resultante está compuesto por un total de 596.756 observaciones tres variables: origen, destino y distancia entre ambos puntos.

3.3. Cálculo de ruta óptimo entre dos puntos

Una vez definida tanto la función objetivo y las restricciones a aplicar como la obtención de la matriz de distancias entre todos los posibles nodos, se puede abordar la descripción del algoritmo de optimización utilizado para obtener la ruta entre dos puntos.

Como anteriormente se mencionaba, el problema que se trata es el del camino más corto, que consiste en encontrar un camino entre dos nodos de tal manera que la suma de los pesos de las aristas que lo constituyen es mínima. De esta manera, se plantea una red compleja dirigida donde los nodos son los puntos de origen/destino además de los puntos de recarga/estaciones de servicio propuestas y las aristas se generan entre todos los nodos de manera bidireccional. El peso de las aristas es igual al tiempo obtenido con la función objetivo.

Para realizar este proceso de generación de red compleja y optimización de la ruta, se ha recurrido a la librería *networkx*. Esta propone una serie de algoritmos de optimización de grafos tales como el algoritmo de Dijkstra o el de A *.

La idea subyacente del algoritmo de Dijkstra consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. Se trata de una especialización de la búsqueda de costo uniforme y, como tal, no funciona en grafos con aristas de coste negativo.

Partiendo de esta base, el algoritmo de A * es una variante optimizada del algoritmo Dijkstra que trata de buscar un mejor camino mediante el uso de una función heurística que da prioridad a los nodos que se supone que son mejores que otros, mientras que Dijkstra simplemente explorar todos los caminos posibles.

Tras analizar ambas opciones, se decide elegir el algoritmo A * como base para el optimizador de rutas de Electric Route, dado que las soluciones encontradas por éste llevan a un tiempo total de viaje equiparable al de Dijkstra y, sin embargo, conlleva un menor tiempo de ejecución.

4. Herramienta web: Electric Route

Se decide utilizar como FrontEnd de la herramienta *Electric Route* el microframework de Python Flask, desarrollando una aplicación web que permita a los usuarios registrarse y simular sus rutas, así como acceder e interactuar con la información en la que se basa el modelo. Además, de la librería Flask, se hace uso del kit de herramientas de código abierto Bootstrap, el cual proporciona diferentes componentes para maquetar la página web.

4.1. Estructura del proyecto

El proyecto se divide utilizando Blueprints, esto es, una colección de vistas, plantillas y recursos estáticos que pueden ser utilizados por la aplicación. La estructura es la siguiente:

```
+ project
|_ __init__.py
|_ auth.py
|_ main.py
|_ static/
|_ templates/
```

Cada uno de los elementos contiene:

- **__init__**: Contiene parte de la lógica de la aplicación, en particular la creación e inicialización de la app, así como los distintos componentes y extensiones.
- **auth.py**: Este módulo inicializa el Blueprint *auth* y contiene la gestión de usuarios: *signup*, *login* y *logout*. Toda la información y registro de usuarios se lleva a cabo mediante el objeto *session* de la librería *flask*.
- **main.py**: Este módulo inicializa el Blueprin *main* y contiene las principales vistas de la app: *index, Route, frequentroutes, profile, cars* y *resetpassword*.
- **static**: Directorio que contiene todos los recursos estáticos, es decir, imágenes, css, y ficheros javascript necesarios para el desarrollo de la web, tanto propios como los descargados de la librería Bootstrap.
- **templates**: Contiene las plantillas de la web, esto es, los ficheros HTML/Jinja2 que darán formato a las páginas vistas por el usuario. Existen dos ficheros base: *base_login* y *base*, a partir de los cuales heredan el formato el resto de plantillas.

4.2. Configuración

Toda aplicación web depende de un conjunto de valores parametrizables. Estos parámetros de configuración permiten, entre otros, definir la cadena de conexión a la base de datos, el idioma por defecto, etc.

De esta forma, una aplicación Flask también depende de una serie de parámetros a configurar. En el fichero *__init__*, se define el método *create_app*, el cual contiene la configuración de dichos parámetros, y son accesibles desde cualquier parte del código. Principalmente, los valores configurados hacen referencia a la conexión con la base de datos: *mysql_db*, *mysql_database_port*, etc.

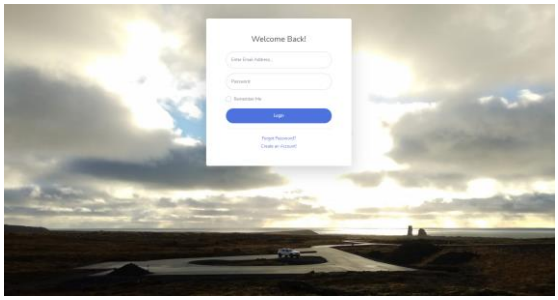
Cabe destacar que el proyecto y, en particular, el desarrollo de la aplicación, tiene el objetivo de poder desplegar la aplicación tanto en local como en un entorno de producción. Por este motivo ha sido necesario configurar parámetros tales como *mysql_host*, *db_host*, *mysql_user*, etc.

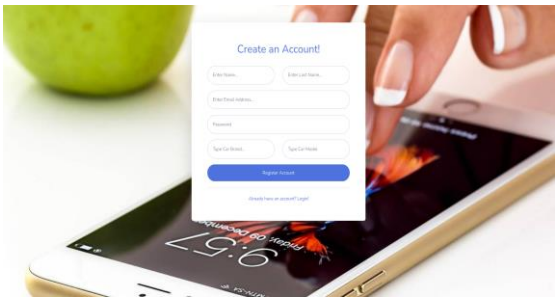
Por último, Flask utiliza el mecanismo de cookies para disponer de la información entre cada una de las peticiones que realiza el usuario. Es necesario, por tanto, definir una clave de sesión Flask con la información de la sesión. Se utiliza el comando **app.secret_key**.

4.3. Guía del usuario

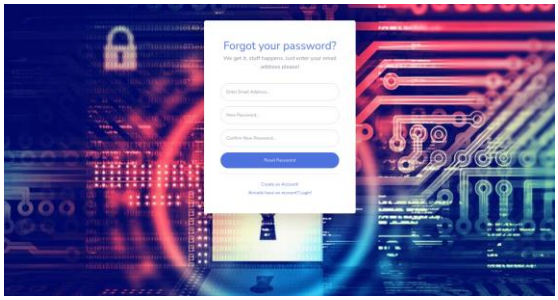
Electric Route consta de una interfaz gráfica simple e intuitiva para el usuario, que le permitirá navegar con facilidad.

El control de acceso a la aplicación se implementa a través de la página de *login* de usuarios, la cual garantiza la seguridad de la aplicación. Así mismo, se crea un formulario para dar de alta a nuevos usuarios que recoge información como el nombre, el email, el password y el tipo de coche.



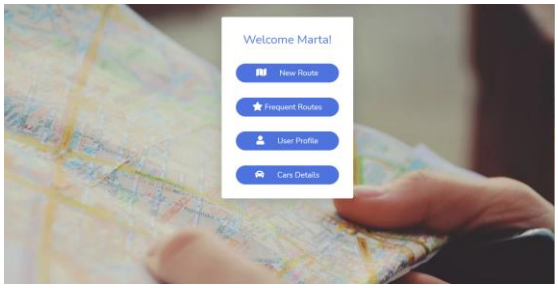


Por último, de cara a la autenticación de los usuarios se permite la opción de resetear la contraseña.

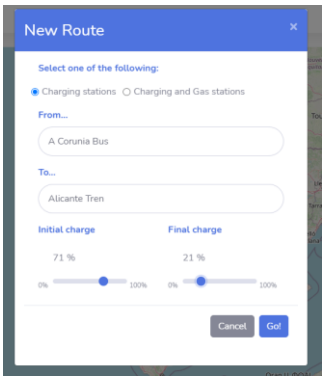


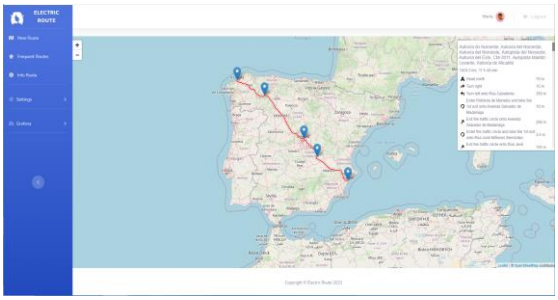
Estas tres páginas están configuradas de forma que detectan y devuelven mensajes de error según sean los datos introducidos por el usuario: campos faltantes, email erróneo, usuario no registrado, etc.

Una vez logueado en la aplicación, el usuario tendrá acceso a la página del *index,* desde la que podrá acceder a las distintas visualizaciones: cálculo de una nueva ruta, consulta de las rutas frecuentes, información del usuario e información acerca del coche con el que se registró.



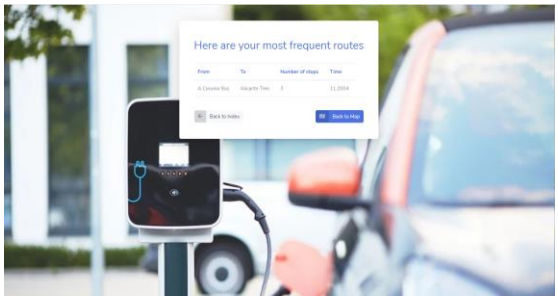
Con el primer botón se accede a la plantilla de *route*, la cual es la página principal y permite al usuario visualizar nuevas rutas, para ello, debe acceder desde la barra lateral al botón *new route* y completar el formulario. La información solicitada será: origen y destino de la ruta, carga inicial del coche y carga final con la que se desea llegar al destino, y el tipo de estacionamiento donde realizar las paradas, puntos de recarga o estaciones de servicio. Será necesario cumplimentar todo el formulario.



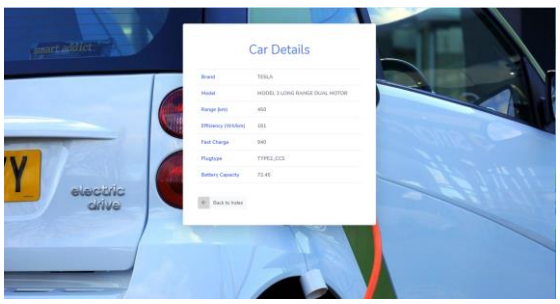


Como resultado se obtiene, por un lado, la ruta gráficamente en el mapa con las indicaciones para realizar el trayecto y, por otro lado, desde el menú *Info Route* se accede al detalle de la información: origen, destino, tiempo de trayecto, número de paradas y tiempo invertido, carga inicial y final.

El usuario tendrá la opción de acceder a sus rutas frecuentes y ver el detalle de cada una de ellas, con el botón *Frequent Routes*.



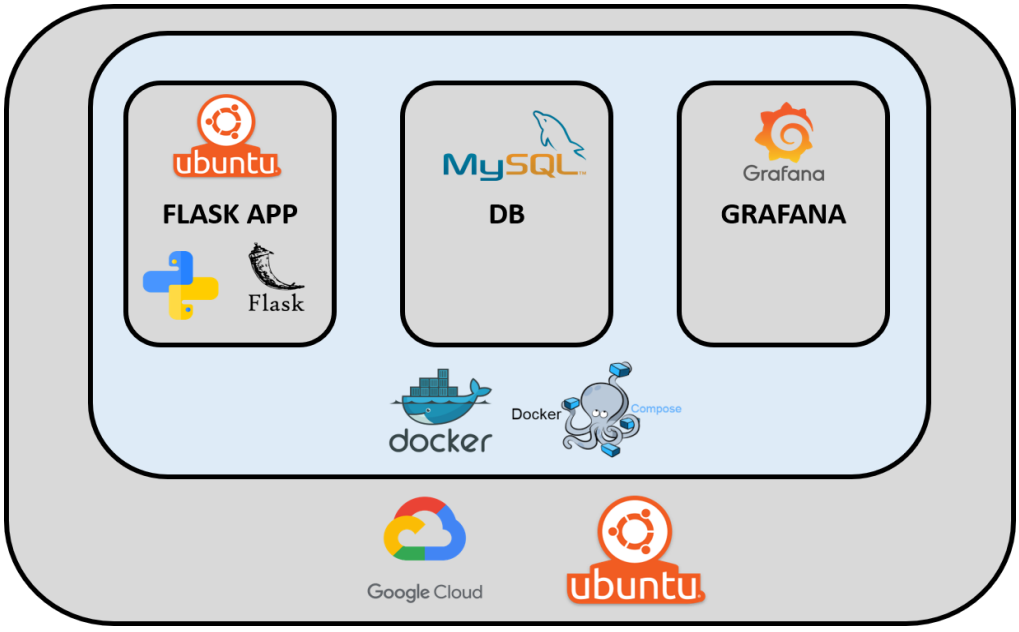
Existen dos vistas para el perfil del usuario, por un lado, desde *Car details*, podrá ver información relevante de su coche, como la autonomía o el consumo de energía por cada 100km, entre otros detalles. Por otro lado, en la página *User Profile* podrá consultar y editar sus datos de registro.



Por último, Electric Route considera interesante que el usuario pueda acceder a la información en la que se basan los algoritmos del modelo de recomendación. Por este motivo, se ha habilitado un apartado en la vista principal que dirige al usuario a varios dashboard en Grafana. Estos informes se centran en tres KPIs: coches, puntos eléctricos y distribución geográfica, y sirven al usuario como comparativa global. Por ejemplo, puede comparar los distintos coches eléctricos que existen en el mercado y las características de cada uno de ellos. Conocer, tanto la distribución de los puntos de recarga por provincia antes de realizar un viaje como el número de conectores disponible o el tipo de conector.

5. Productivización

La productivización del sistema que permite el correcto funcionamiento de la aplicación *Electric Route* en un entorno operacional se basa en el uso de dos tecnologías distintas: Docker/Docker-Compose y Google Cloud Platform. Mientras que la primera permite la virtualización de aplicaciones en contenedores de manera automática y la interacción entre distintos contenedores, la segunda proporciona una plataforma de computación remota en la nube donde desplegar el sistema.



5.1. Docker y Docker-Compose

El sistema de la aplicación está formado por tres contenedores Docker distintos:

- ***flask_app***: basado en una imagen base de Ubuntu 18.04, este contenedor tiene la función de ejecutar los scripts de Flask que componen el FrontEnd de la aplicación y, a su vez, de llamar a los scripts de Python que forman el núcleo del modelo de cálculo de rutas óptimas en el BackEnd.

De esta manera, a parte de la base de Ubuntu 18.04, este contenedor se basa en una imagen que contiene Python3 y todas las librerías necesarias para el correcto funcionamiento de los scripts del FE y BE (obtenida mediante la generación de un fichero *DockerFile*).

- ***db***: basado en la imagen de la última versión disponible en el repositorio MySQL, este contenedor contiene la base de datos MySQL con las tablas necesarias, que se cargan desde un fichero mysqldump (*tfm.sql*) al crear el contenedor.
- ***grafana***: basado en la imagen de la última versión disponible en el repositorio grafana/grafana, este contenedor contiene la aplicación de monitorización y visualización de datos Grafana con los dashboards generados a partir del contenido de la base de datos. Tanto la información referente a la base de datos con la que tiene que conectar (*datasources*) como los dashboards utilizados (*dashboards*) se cargan al crear el contenedor y pueden actualizarse mediante un reinicio del contenedor.

Para que los contenedores puedan interactuar entre sí y formen parte de la misma red, se utiliza Docker-Compose, mediante la definición de los contenedores como servicios en un fichero *docker-compose.yml*.

5.2. Google Cloud Platform

Para realizar el despliegue en una plataforma remota en servicio tipo nube, se utiliza el servicio *Compute Engine* de Google Cloud Platform. De esta manera, se construye una instancia de VM con las siguientes características:

- **Tipo de máquina**: e2-medium (2 vCPUs, 4 GB de memoria)
- **Region**: europe-west
- **Zona**: europe-west1-b
- **Disco de arranque**: SO Ubuntu 18.04 LTS
- **Cortafuegos**: Permitir el tráfico HTTP/HTTPS

Además, para poder acceder a la aplicación desde el exterior, es necesario habilitar una regla de Firewall desde el apartado de Redes/Red de VPC con las siguientes especificaciones:

- **Dirección**: Entrada
- **Acción tras coincidencia**: Permitir
- **Filtros de origen**: Intervalos de IP 0.0.0.0/0
- **Protocolos y puertos**: tcp:3000, tcp:3306, tcp:5000

Una vez hecho esto, el despliegue se realiza siguiendo las instrucciones del fichero GCP.txt, que se basa en los siguientes pasos:

1) Acceder a través de SSH a la instancia VM
2) Clonar el repositorio de [GitHub]
3) Instalar Docker y Docker-Compose mediante la ejecución del script de bash *install_docker.sh*
4) Preparar el entorno mediante la ejecución del script de bash *prepare_app.sh*
5) Levantar los contenedores con Docker-Compose y comprobar que el sistema se ha levantado de manera correcta
6) Acceder a través del navegador a la web: http://*ip_externa*:5000/login

   donde *ip_externa* es la External IP de la instancia VM del Compute Engine de GCP

En el siguiente apartado se clarifica una vía más rápida para su acceso.

6. Acceso a las herramientas ofrecidas

6.1. Interfaz Electric Route

Para poder acceder de forma más sencilla a la aplicación desarrollada se dispone de la siguiente ruta: **Electric Route**. Debido al cargo económico que se puede recibir debido a un despliegue continuo en GCP esta IP solo estará **disponible hasta el domingo 21 de febrero de 2020**. En caso de querer acceder a la aplicación una vez pasada la fecha, por favor, póngase en contacto con las integrantes del equipo.

6.2. Repositorio proyecto

Como se ha subrayado a lo largo del documento, el desarrollo del trabajo se ha dejado disponible en el siguiente repositorio: Proyecto Electric Route.

Se está trabajando en el cambio de licencia del repositorio para permitir el acceso a cualquier persona externa y que no se puedan realizar cambios. Hasta entonces el equipo ha decidido no dejar el repositorio libre y dar accesibilidad a los docentes Carlos Ortega y Santiago Mota. En caso de necesitar más accesos póngase en contacto con las integrantes del equipo.

7. Discusión y conclusiones

En conclusión, se podría determinar que se han cumplido con los objetivos marcados al inicio del proyecto, puesto que se ha conseguido realizar un modelo de rutas para coches eléctricos y se ha podido mostrar al usuario.

El principal inconveniente que se encuentra en los consumidores a la hora de la compra de este tipo de coches es la autonomía y su carga, puesto que se consideran vehículos sólo utilizables para ciudad. Este proyecto debe de considerarse como un punto de palanca para la incentivación del uso y compra de coches eléctricos. Asimismo, remarca la necesidad de inversión en cuanto a los puntos de carga de estos.

No obstante, los principales inconvenientes que se han detectado en el desarrollo de este trabajo son, por un lado, la autonomía de los diferentes coches eléctricos impidiendo la creación de rutas óptimas y recomendables y la escasez de puntos de carga para estos.

En cuanto al segundo punto comentado, cabe destacar la distribución no uniforme de estos. Debido a la gran inversión que conlleva la construcción de puntos de recarga y la baja compra de coches eléctricos, en estos instantes, estos puntos se encuentran principalmente en ciudades importantes dificultando, por ello, la realización de rutas de larga distancia con este tipo de coches. Por este motivo, resulta interesante evaluar cuáles son los lugares con menor densidad de puntos de recarga para poder desplegar una red más uniforme.

Por otro lado, en este mercado se puede detectar una elevada competencia puesto que cada marca de coches crea su propio tipo de conector y, así, hacerlo más exclusivo. Esto conlleva a que sea más difícil ajustar las rutas para cada coche puesto que estos conectores no son compatibles y, por lo tanto, los puntos de carga tampoco.

8. Próximos pasos

Este proyecto tiene un ámbito de ampliación muy extenso desde los datos hasta la generación de rutas y la aplicación en sí. No obstante, estas mejoras no solo dependen del propio equipo sino también del actual mercado de coches eléctricos y lo relacionado con él.

En primer lugar, en un futuro próximo sería idóneo poder incluir en la base de datos más marcas y tipos de coches de los ya propuestos a medida que vayan subiendo las flotas ofrecidas por las diferentes

marcas de automoción. Además, podría ser interesante adaptar el algoritmo para poder introducir vehículos híbridos.

Asimismo, debido a la no uniformidad en la red de puntos de recarga, el tiempo de la ruta se eleva considerablemente. Con la fomentación del uso de coches eléctricos el número de puntos aumentará y se podrán conseguir menores tiempo de recorrido. Gracias al análisis relacionado se ha podido detectar la escasez de lo comentado y la gran mejora que hay en este aspecto si se introducen puntos de recarga en zonas donde ahora mismo hay escasez.

En estos instantes, la aplicación ubica tanto el destino como el origen en sitios clave de cada una de las ciudades. En una segunda fase del proyecto se podrían añadir todas las calles en el ámbito nacional para poder dar la posibilidad de realizar rutas más personalizadas.

Por último, sería muy interesante añadir la función en tiempo real geolocalizando al usuario y añadiendo datos del tráfico para una devolución de tiempos más exacta, para lo cual habría que introducir nuevas tecnologías.

Uno de los mayores factores limitantes de la aplicación actualmente es el tiempo de ejecución, debido al hecho de que Python es un lenguaje interpretado. Para poder llevar a cabo los puntos comentados y no impactar en el tiempo de ejecución de manera excesiva, se proponen dos opciones:

- El uso de tecnologías tipo Hadoop/Spark que permiten procesar grandes volúmenes de datos beneficiándose de la computación distribuida y poder de esta manera soportar la gran volumetría que se espera en un futuro.
- La introducción de las partes de código que más tiempo de ejecución conllevan (como es el cálculo en RT de las distancias) en lenguaje compilado utilizando lenguajes de programación para simplificar la escritura de módulos de extensión para Python en C y C++ como puede ser Cython.

Por otro lado, la opción de adaptar esta aplicación al móvil sería de gran utilidad ofreciendo así un servicio más óptimo y fácil al usuario.

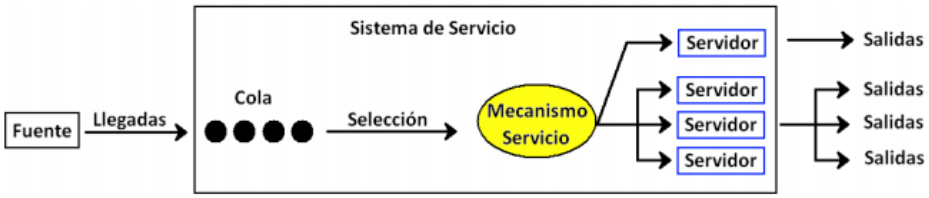
9. Referencias

- Apuntes del módulo de MySQL. Ofrecidos por la docente Isabel Riomoros.
- Apuntes del módulo de Python. Ofrecidos por el docente Cristóbal Pareja.
- Apuntes del módulo Minería de datos y modelización predictiva. Ofrecidos por el docente Aida Calviño.
- Apuntes del módulo Aplicaciones del Big Data en la empresa. Ofrecidos por el docente Carlos Ortega.
- Apuntes del módulo Productivizar Un Modelo. Ofrecidos por el docente Carlos Ortega.
- J2LOGO - Tutorial de Flask en español: Desarrollando una aplicación web en Python
  Disponible en: https://j2logo.com/tutorial-flask-espanol/
- Stack Overflow - Dudas genéricas a lo largo del desarrollo del TFM
  Disponible en: https://es.stackoverflow.com/
- Kaggle - Obtención del dataset coches eléctricos
  Disponible en: https://www.kaggle.com/geoffnel/evs-one-electric-vehicle-dataset
- Tomtom Developer Portal - Obtención del dataset de los puntos de recarga
  Disponible en: https://developer.tomtom.com/search-api/search-api-documentation/ev-charging-stations-availability
- API Google Distance Matrix - Pruebas para el cálculo de la matriz de distancias
  Disponible en:
  https://medium.com/how-to-use-google-distance-matrix-api-in-python/how-to-use-google-distance-matrix-api-in-python-ef9cd895303c
- Hub ArcGis- Obtención del dataset de gasolineras
  Disponible en:
  https://hub.arcgis.com/datasets/4d08b1eaf10c466dafd1dfbd81b07f64_0?geometry=-90.494%2C22.783%2C76.762%2C47.537
- Geoff Boeing - Clustering para reducir dataset espacial con DBSCAN
  Disponible en: https://geoffboeing.com/2014/08/clustering-to-reduce-spatial-data-set-size/
- Sukiweb - Encontrando caminos óptimos con grafos en Python
  Disponible en:
  http://sukiweb.net/archivos/2018/05/30/encontrando-caminos-optimos-con-grafos-en-python/
- Medium - Representar datos no temporales en Grafana
  Disponible en:
  https://medium.com/grafana-tutorials/graphing-non-time-series-sql-data-in-grafana-8a0ea8c55ee3
- GitHub - Uso de Google Maps Services
  Disponible en:
  https://github.com/googlemaps/google-maps-services-python
- Towards Data Science - Desplegar Docker en Google Cloud Platform
  Disponible en:
  https://towardsdatascience.com/how-to-deploy-docker-containers-to-the-cloud-b4d89b2c6c31

Anexo

Modelo de cola M/M/s

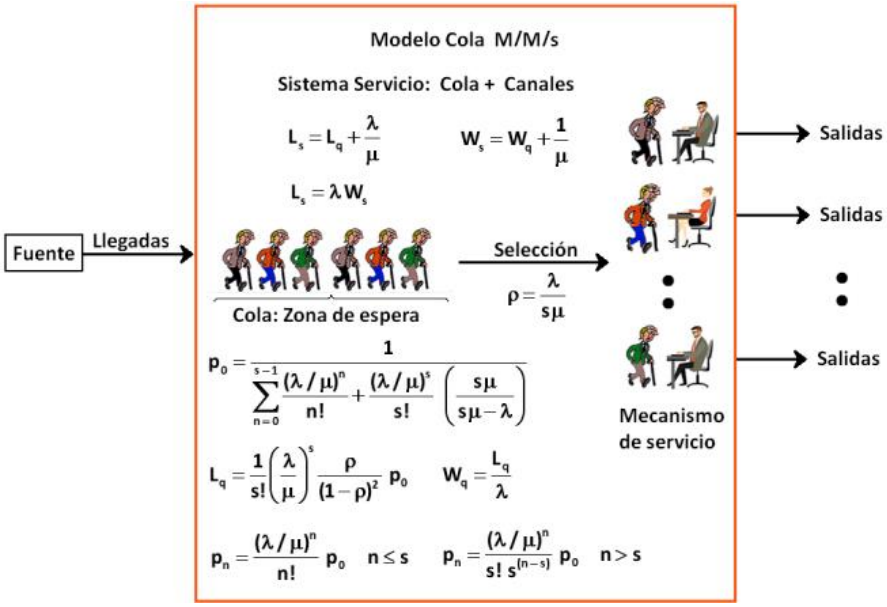
Se trata de un tipo de modelo de teoría de colas que supone que los tiempos entre llegadas y los tiempos de servicio son variables aleatorias distribuidas exponencialmente, la disciplina es FIFO y la población es infinita. Además, el número de servidores (en nuestro caso surtidores) puede ser cualquier número natural tal que $s \geq 1$.



A continuación, se describe brevemente los detalles de un sistema de colas y las características de este modelo en concreto:

- Fuente de llegada de clientes: La llegada de clientes hasta un momento específico sigue una distribución de Poisson.
- Patrón de servicio de servidores: En este caso, es estacionario ya que no varía con el tiempo transcurrido.
- Disciplina de cola: FIFO (first in first out): Se atiende al cliente en el orden que llegan a la cola, el primero en llegar será el primero en ser atendido.
- Capacidad del sistema: Es el número máximo de clientes que pueden estar dentro del sistema haciendo cola antes de ser atendidos para recibir el servicio. En este caso, se ha considerado infinito.
- Número de canales de servicio: En este modelo, se trata de sistemas multiservicios con una única línea de espera para todos.
- Número de etapas de servicio: Se establece como unietapa.
- Tiempo de servicio: es el tiempo que transcurre desde el inicio del servicio para el cliente hasta su terminación en una electrolinera (tiempo de recarga del coche). En este modelo, la distribución de probabilidad de los tiempos de servicio de cada servidor se ha supuesto la misma para todos ellos.

Las fuentes de variación en los problemas de colas provienen del carácter aleatorio de la llegada de clientes y de las variaciones que se registran en los distintos tiempos de servicio. Generalmente cada una de esas fuentes suele describirse mediante una distribución de probabilidad. En este caso, se asume que el tiempo entre diferentes llegadas de clientes sigue una distribución exponencial Exp (λ), o lo que es equivalente, que el ritmo de llegadas sigue una distribución de Poisson.



A través de la aplicación de las fórmulas que se muestran en la ilustración número se consigue calcular el tiempo medio de espera en cada punto de recarga del modelo de optimización. En el supuesto de haberse manejado datos reales, tanto el parámetro λ como μ serían el resultado de estimaciones procedentes de muestras de la población en cuestión.

La variable lambda (λ) es el número medio de coches que llegan en una hora a una electrolinera cualquiera. Se trata de una de las variables cuyos valores han sido asignados de manera ficticia, pero siguiendo una cierta lógica. Dichos valores varían en función de tres factores: la ubicación de la electrolinera (a menos de 50km de grandes ciudades o más), los meses del año (verano o invierno) y la franja horaria del día (día o noche).

La variable mu se trata de la tasa de servicio por surtidor, es decir, el número de clientes por hora en una electrolinera cualquiera. En el modelo adaptado aparece con valor 3.

ρ = El factor de utilización o congestión del sistema

p_0 = La probabilidad de que ningún cliente se encuentre en el sistema de colas

L_q= El número de clientes de la cola

W_q = El tiempo medio de espera en la cola