Modelado predictivo y visualización del tráfico en la ciudad de Nueva York mediante fusión de datos abiertos, sobre plataforma big data.

TÍTULACIÓN: Grado en Ingeniería Informática.

AUTOR: Juan Manuel Ruiz Fránquiz.

TUTORIZADO POR:

Javier Jesús Sánchez Medina.

Carolina Peña Alonso.

Julio de 2021.

Agradecimientos

Con este trabajo se culmina una fase de formación académica que no hubiese sido posible sin el apoyo de mis tutores, que me han orientado y guiado en el proceso.

También mis agradecimientos para mi familia, mis amigos y compañeros, especialmente a María Rodríguez por haberme apoyado durante todo este tiempo.

Resumen

La gestión del tráfico en las ciudades desarrolladas pasa inevitablemente por el modelado y simulación del mismo, que permite evaluar alternativas de optimización de los diferentes parámetros relacionados.

El propósito de este trabajo es desarrollar modelos predictivos para el estado del tráfico aprovechando la plataforma de datos abiertos de la ciudad de Nueva York y otras fuentes de datos abiertas. Para ello, será necesaria una fase de ingestión de datos de diferentes estructuras, tipos y periodicidades dentro de una única base de datos. Una vez realizado, se genera un modelo predictivo para el tráfico cuyos resultados serán representados en una página web sencilla.

Abstract

Traffic management in developed cities inevitably goes through its modelling and simulation, which allows evaluating optimization alternatives of the different related parameters.

The purpose of this project is, taking advantage of New York City open data plataform and other sources of open data, develop predictive models of the status of traffic. To accomplish this, an ingestion phase will be necessary which will integrate data of different structures, types and periodicities within a single database. Once done, a predictive model will be created for the traffic status, whose results will be represented on a simple webpage.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1
Introducción

La ocupación humana a partir de núcleos urbanos se ha intensificado a nivel mundial en las últimas décadas. Esto ha supuesto nuevos retos a la hora de planificar y gestionar las ciudades por parte de las administraciones. Uno de los mayores retos a los que se enfrentan los gobiernos es la gestión del tráfico para evitar grandes retenciones durante las horas punta.

Nueva York es un ejemplo de una ciudad cuya gestión del tráfico ocupa una buena parte de la agenda de la administración local. Esta ciudad es una de las mayores metrópolis del mundo. Su población, contando el área metropolitana, supera los 19 millones de personas (Wikipedia, 2021), convirtiéndola en la ciudad más poblada de Estados Unidos y la segunda más poblada de todo el continente americano, solo por detrás de Ciudad de México. Su gran densidad de población hace que muchos de los servicios puedan saturarse. Un ejemplo claro de esto es el tráfico. Es muy común ver en fotografías las calles repletas de coches, indicando la saturación de las calles.

De entre los cinco distritos de Nueva York (Manhattan, Brooklyn, Queens, Bronx y Staten Island), es Manhattan el que registra un mayor número de retenciones. En circunstancias normales, cruzar en un vehículo calles como puede ser la Calle Canal de Manhattan no tardaría más de 2 minutos, pero, por culpa de la gran cantidad de tráfico, este tiempo se puede demorar hasta más de 20 minutos. (Shaftoe, s.f.).

El objetivo de este proyecto es relacionar distintos factores como son la calidad del aire y distintos parámetros climáticos con el tráfico, para generar modelos predictivos en tiempo real sobre el estado de distintas calles del distrito de Manhattan y visualizarlos en gráficos dentro de una página web. Para ello, el proyecto contará con dos módulos: el primero de los módulos se centra en la ingestión, análisis y procesamiento de datos históricos y en tiempo real para generar modelos predictivos. El segundo de los módulos utilizará las predicciones generadas por los modelos para mostrar los resultados en una página web.

El proyecto en cuestión podría clasificarse como un proyecto de Big Data, debido a que cumple las 5 dimensiones que caracterizan a este tipo de proyectos. Son conocidas como las 5 “V”. Estas “V” son: (Juan, 2019)

- Volumen: Los proyectos de Big Data suelen tener unas cantidades bastante elevadas de datos. Durante el desarrollo del proyecto, se trabajará con grandes cantidades de datos para entrenar y analizar modelos que permitan realizar predicciones sobre el estado del tráfico.
- Velocidad: En un proyecto de Big Data, el flujo de datos es constante. El proyecto recibirá datos nuevos de las distintas fuentes de información utilizadas de forma constante para generar las predicciones en tiempo real.
- Variedad: Debido a que en el proyecto se recogen datos de múltiples fuentes, los datos tienen características propias que deberán ser analizadas antes de proceder con la realización de los modelos.
- Veracidad: La veracidad en cuanto a los datos recogidos depende de las fuentes de información utilizadas. En este caso, los datos recogidos son de fuentes gubernamentales o páginas que hacen uso de datos oficiales.
- Valor: Los datos recogidos deberán ser analizados para buscarle un sentido y un valor que ayude en la labor de solucionar el problema en cuestión. En el proyecto, el valor residirá en la capacidad de realizar predicciones a futuro sobre el estado del tráfico que puedan ser de utilidad tanto para los ciudadanos como para las autoridades que tengan competencias y puedan aportar soluciones reales que mejore la calidad de vida de los ciudadanos.

La realización del proyecto comenzará por el estudio de las herramientas y las fuentes de información disponibles, antes de comenzar con la recolección de datos históricos y en tiempo real. Tras obtener los datos, se realizará un análisis de distintos modelos para la predicción del estado del tráfico y se escogerá aquel modelo que presente mejores resultados. Una vez se cuente con el mejor modelo posible, se creará una página web para representar los resultados de las predicciones. Finalmente, se realizará una integración de las distintas fases realizadas de tal forma que sea posible generar y mostrar predicciones en tiempo real en la página web.

Este proyecto nace de una motivación personal por aprender y profundizar sobre algunos de los aspectos del Big Data, el Machine Learning y todas las tecnologías y metodologías utilizadas en la actualidad para resolver este tipo de problemas.

Capítulo 2
Estado actual y objetivos iniciales

2.1 Estado del arte

Mucha gente en Nueva York, al igual que en otras grandes ciudades del mundo, viven alejados de su trabajo. A diario, todas estas personas deben desplazarse hacia los núcleos económicos de la ciudad para poder desempeñar su vida laboral. Pese a que millones de personas utilizan el metro u otros tipos de transporte a diario, muchos optan por trasladarse utilizando su vehículo privado por una u otra razón, generando retenciones importantes en los núcleos urbanos. Otro de los motivos de la saturación en el tráfico es la propia geografía de la ciudad: Muchos de los distritos están conectados por un número muy limitado de túneles o puentes, lo que hace que en hora punta el tráfico se acentúe en esos puntos.

Un tráfico saturado generalizado en una ciudad puede acarrear muchos problemas. En 2019, “los viajeros en Nueva York perdieron 133 horas o casi $1900 dólares al año, al permanecer sentados en el tráfico”. (Ny1, 2019).

No en vano, esta ciudad es calificada como “la cuarta ciudad más congestionada del país” (Ny1, 2019) y “una de las ciudades con peor tráfico vehicular del mundo”. (Univision, 2017).

Además de problemas económicos, el tráfico puede ser muy perjudicial para la salud de las personas en las ciudades. Hay una clara correlación entre la contaminación y la salud de las personas. La congestión hace que las emisiones de los vehículos aumenten y, por tanto, el aire ambiental se vea perjudicado. Además, “estudios recientes han demostrado un exceso de morbilidad y mortalidad para los conductores, los viajeros y las personas que viven cerca de las carreteras principales” (Kai Zhang, 2013). Este problema causa al año “aproximadamente 1400 muertes prematuras y pérdidas de decenas de millones de dólares en gastos sanitarios por la contaminación causada por los camiones y coches que pasan por las calles”. (J.Cuba, 2021)

Es por ello por lo que la gestión del tráfico es una prioridad en la ciudad. Para intentar mejorar la situación, desde comienzos de 2021, hay que pagar por circular por Manhattan. (Pozzi, 2019). Esta medida pretende incentivar el uso del transporte público

para moverse por la ciudad que, además, se modernizaría gracias a los ingresos recaudados por la nueva ley.

Además de este tipo de medidas, una forma de planificar soluciones es mediante el modelado y la simulación del tráfico. Gracias a ello, se pueden estudiar los distintos factores que pueden influir en el estado del tráfico y evaluar las distintas alternativas de optimización. Para ello, se debe monitorizar. Las administraciones públicas de ciudades como Nueva York han proporcionado en plataformas de datos abiertos las mediciones que obtienen, de manera que los ciudadanos o científicos puedan utilizar estas fuentes para proyectos alternativos que añadan valor adicional a los datos.

2.2 Objetivos

El objetivo principal de este trabajo es la generación de predicciones en tiempo real para el estado del tráfico en Nueva York. Para realizar estas predicciones se utilizarán los datos generados por las API en tiempo real. Estos resultados serán representados en una página web. Durante la planificación del proyecto se planteó una serie de objetivos que fueron realizados durante el desarrollo del trabajo. Los objetivos propuestos son los siguientes:

- Realizar un estudio de las diferentes fuentes de información disponibles
- Identificar las diferentes tecnologías de big data e inteligencia artificial que sean convenientes para el estudio a realizar.
- Efectuar la ingestión y fusión de datos dentro de una arquitectura big data.
- Desarrollar el modelo predictivo del estado del tráfico en la ciudad de Nueva York.
- Crear página web para mostrar los resultados del análisis
- Evaluar los resultados obtenidos.

Capítulo 3
Competencias específicas y aportaciones del trabajo

> CP07: Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos. (Escuela de Ingeniería Informática ULPGC, s.f.)

El desarrollo del proyecto ha concluido en la realización de esta competencia. Durante el desarrollo del proyecto se ha trabajado con un gran volumen de datos de distintas fuentes para obtener datos de tráfico, clima y calidad de aire. Esta información se ha obtenido de forma automática haciendo uso de llamadas a las distintas API. Una vez obtenidos todos estos datos, han sido pre procesados y manipulados para poder entrenar modelos de aprendizaje computacional que tenían como objetivo la predicción del estado del tráfico a futuro. Todas estas herramientas han sido utilizadas finalmente para la creación de una página web en la que se muestra en tiempo real los resultados de los modelos predictivos.

Este proyecto aporta un ciclo completo en cuanto a la extracción, preparación, análisis e implementación de soluciones que hacen uso de grandes cantidades de datos para resolver un problema como es la falta de modelos que sirvan para evitar problemas de tráfico en la ciudad de Nueva York. Durante el proyecto se han tocado las distintas fases del Big Data para resolver un problema que puede extrapolarse a otras ciudades o problemas.

Capítulo 4
Desarrollo

4.1 Metodología aplicada

La metodología aplicada a este trabajo fue la realización de los distintos apartados en bloques. El proyecto cuenta con una estructura que hace que sea muy difícil poder realizar más de un apartado al mismo tiempo. Esto es debido a que, por ejemplo, no se puede realizar un modelo predictivo para el estado del tráfico sin haber realizado previamente un estudio de las tecnologías a utilizar o de los datos con los que se va a contar a la hora de trabajar con los modelos. Este punto es aplicable a otros bloques de los que se compone el proyecto. Por ello, el proyecto se dividió en los siguientes bloques:

- Estudio previo. Este bloque comenzará con un estudio de los conceptos básicos que serán utilizados durante el proyecto. Posteriormente, se hará un estudio de las fuentes de información disponibles, sus API y la estructura de datos. En este paso se seleccionarán, analizarán y comprenderán todas las posibles fuentes de información que puedan aportar utilidad al proyecto. Esta parte es fundamental, puesto que es la base sobre la que se trabajará en el resto del proyecto. Finalmente, se hará un estudio de todas las herramientas utilizadas y otras alternativas de tecnologías que podían ser utilizadas también, pero no fueron seleccionadas por una u otra razón.
- Recolección de datos. El bloque de recolección de datos consta de dos apartados. Para poder trabajar con datos y crear modelo es necesario buscar y analizar los datos históricos de las fuentes de datos elegidas. Por ello, el primer paso será realizar un tratamiento y estudio de los datos históricos. Este apartado ayudará a comprender también la estructura de estos datos y las características de los mismos. Una vez se cuente con un histórico de datos con los que trabajar, se puede proceder al segundo apartado de este bloque: la recolección de datos en tiempo real. En este caso, los datos recolectados serán del mismo tipo de los históricos, pero habrá que realizar un tratamiento especial teniendo en cuenta que los datos vienen en diferentes frecuencias de tiempo.

- Creación y evaluación de los modelos predictivos para el tráfico. Una vez obtenido un archivo o una base de datos con los datos deseados, se procedió a crear una serie de modelos que pudiesen ayudar a predecir el estado del tráfico en un intervalo de tiempo futuro. Los modelos fueron creados haciendo uso de tecnologías escogidas previamente. Una vez realizados los modelos, se buscó optimizarlos para poder obtener el mejor resultado posible y se realizó una comparación entre los resultados obtenidos en cada uno de los modelos para seleccionar el más apropiado para aplicar a los datos.
- Desarrollo de la página web en la que se muestren los resultados del análisis: Una vez seleccionado el modelo con el que se va a realizar las predicciones, se procedió al desarrollo de la página web que tenía como objetivo mostrar de forma visual los resultados obtenidos del modelo. Esta página web funciona de forma dinámica y sus gráficas se irán actualizando a medida que se generen nuevas predicciones del modelo en tiempo real para el estado del tráfico en cada una de las calles medidas.
- Integración de todas las fases: en este bloque, todas las fases realizadas anteriormente fueron conectadas para tener dos módulos funcionales que ayudasen a realizar predicciones y visualizar los datos en tiempo real.

4.2 Estudio previo

4.2.1 Conceptos básicos

En el transcurso del proyecto se hará uso de algunos acrónimos y conceptos técnicos, por lo que es importante conocer el significado de ellos previamente:

- API: Una API es una interfaz de programación de aplicaciones, “que permite el intercambio de información entre dos componentes de software independientes (Slate, 2019).

  En el caso de este proyecto, se hará uso de API abiertas. Este tipo de API “proporcionan a los desarrolladores externos un modo de acceder fácilmente a la información e integrarla entre herramientas” (Slate, 2019). El motivo de usar API públicas es que la información con la que se trabajará es externa al proyecto, por lo que se deberá acceder a ella.

- Inteligencia Artificial: “La inteligencia artificial o IA se refiere a los sistemas o máquinas que imitan la inteligencia humana para realizar tareas y que tienen la

capacidad de mejorar iterativamente a partir de la información que recopilan" (Oracle, s.f.).

El propósito de este proyecto es utilizar la inteligencia artificial para detectar los patrones entre las distintas variables analizadas para generar modelos predictivos sobre el estado del tráfico.

- Machine Learning: Es un campo dentro de la inteligencia artificial que "permite que las máquinas aprendan sin ser expresamente programadas para ello. Es una habilidad indispensable para identificar patrones entre los datos" (Alameda, 2020).

  En el proyecto se va a tratar con distintos datos y se buscará encontrar patrones entre ellos para las predicciones. Es por ello por lo que este concepto es muy importante a la hora de trabajar en este proyecto.

- Deep Learning:

  "El Deep Learning o aprendizaje profundo se define como un algoritmo automático estructurado o jerárquico que emula el aprendizaje humano con el fin de obtener ciertos conocimientos. Destaca porque no requiere de reglas programadas previamente, sino que el propio sistema es capaz de aprender por sí mismo para efectuar una tarea a través de una fase previa de entrenamiento" (SmartPanel, 2019).

  El aprendizaje profundo hace uso de una estructura con redes neuronales artificiales entrelazadas en una capa de entrada, una o varias capas ocultas y una capa de salida. La primera de las capas recibe los datos de entrada. Las capas ocultas son las capas que realizan el procesamiento y los cálculos para entrenar el modelo. Finalmente, en el caso de este proyecto, la salida es el valor de las predicciones que ha generado el modelo teniendo en cuenta los datos que se le han pasado anteriormente. Estos modelos funcionan como una caja negra, ya que únicamente se conocen los valores de entrada y los de salida, pero no se sabe qué ha ocurrido exactamente en el interior (por eso reciben el nombre de capas ocultas)

- Redes Neuronales: Una red neuronal artificial busca "imitar el funcionamiento de las redes neuronales de organismos vivos: un conjunto de neuronas entre sí y que trabajan en conjunto, sin que haya una tarea concreta para cada una" (Julián, 2014). Las capas tienen una serie de neuronas que se encargan de

ajustar su valor interno a medida que se entrena el modelo con el objetivo de tener el menor error posible. Este ajuste se realiza mediante los pesos que se le aplica a cada una de las neuronas. Al comenzar el entrenamiento, estos pesos son asignados de forma aleatoria, y se van modificando a medida que se realiza el entrenamiento.

Durante en el desarrollo de este trabajo se hará uso de redes neuronales (y, por ende, Deep Learning) para generar los modelos predictivos.

4.2.2 Estudio de las fuentes de información disponibles

El estudio de las fuentes de información fue un proceso bastante complejo. Esto es debido a que las fuentes de información debían cumplir obligatoriamente las siguientes condiciones:

- Ser gratuitas o de libre acceso: Existen muchas fuentes de datos, especialmente de las condiciones climáticas, a las que sólo se podía acceder a esos datos tras pagar una suscripción. El propósito de este proyecto es realizar todo el proceso desde la extracción de la información hasta la visualización de las predicciones utilizando fuentes de datos abiertas o gratuitas.
- Poder realizar peticiones sobre fechas pasadas. Muchas de las fuentes de información estudiadas proporcionaban datos únicamente del día actual o de un número limitado de días pasados. Estos datos no son de gran utilidad ya que es necesario tener muchos registros históricos para poder entrenar los modelos predictivos.
- Proporcionar datos que puedan resultar de utilidad para el proyecto: Existen muchas fuentes de datos sobre Nueva York, pero no todas proporcionan datos que ayuden en la creación de un modelo para la predicción del tráfico.

Como alternativa para buscar fuentes de información consideró realizar la extracción de esta información utilizando web scraping. Web scraping hace referencia a la extracción de información de sitios web. Finalmente se descartó esta opción ya que se pudo acceder a las fuentes de información haciendo uso de distintas API. Las API pueden dividirse en 3 categorías: tráfico, clima y calidad del aire.

4.2.2.1 API de tráfico

Para la recolección de datos de tráfico se hizo uso de la API de velocidad del tráfico en tiempo real de la plataforma de datos abiertos de Nueva York. Esta plataforma es administrada por el Centro de Gestión de Tráfico del Departamento de Transporte de la Ciudad de Nueva York. Esta API funciona gracias a múltiples sensores que se encuentran repartidos por las calles de Nueva York que calculan la velocidad media a la que han pasado los coches en los últimos 5 minutos. Esta API cumple todos los campos requeridos:

- Es una fuente de datos pública y gratuita
- Los sensores han estado realizando las mediciones desde hace varios años. Por tanto, esta API permite realizar peticiones dada una fecha.
- Proporciona el campo fundamental para la predicción del estado del tráfico. El proyecto realizará predicciones sobre la velocidad relativa de los coches. Este campo no se encuentra en la API, pero es creado gracias a uno de los campos que sí proporciona esta fuente de información.

Los datos de esta API se van actualizando en intervalos irregulares de tiempo, varias veces al día.

4.2.2.2 API de clima

En el caso de la API del clima, resultó bastante complicado encontrar una fuente de datos. Inicialmente se estudió la API oficial del “National Weather Service”, pero esta opción fue descartada debido a que no se permitía acceder a un histórico de datos. Es por ello por lo que se tuvo que estudiar otras fuentes de datos alternativas. Entre ellas, muchas resultaron ser de pago, o los datos que proporcionaban no eran muy interesantes. Finalmente, se decidió utilizar la API de “Visual Crossing Weather”. Esta API permite realizar un número limitado de peticiones gratuitas al mes. Pese a esto, el número de peticiones es lo suficientemente elevado como para que esto no resultase ser un problema. Además, proporciona datos de gran valor sobre el clima en Nueva York y permite realizar peticiones a datos históricos.

4.2.2.3 API de calidad del aire

Encontrar una fuente para obtener los datos de la calidad de aire resultó ser también un proceso bastante arduo. Esto es debido a que muchas de las fuentes de datos eran de pago, como por ejemplo “BreezoMeter”. Páginas web como “aqicn.org” no proporcionaban una API para acceder a datos históricos, por lo que tampoco fue utilizada.

Finalmente se decidió utilizar la API de AirNow. Este es un proyecto realizado por la Agencia de Protección Ambiental de los Estados Unidos. Utilizan una gran cantidad de estaciones ambientales distribuidas a lo largo de Estados Unidos, Canadá y México para medir la calidad del aire en más de 300 ciudades. Es una fuente completamente gratuita que permite seleccionar la estación de la que se quieren obtener los datos. Además, permite acceder tanto a datos históricos como datos en tiempo real.

La estación seleccionada para recoger los datos se encuentra en el distrito de Manhattan y pertenece al Departamento de Conservación Ambiental del Estado de Nueva York.

4.2.3 Estudio de las tecnologías y herramientas empleadas

Antes de empezar con la creación del proyecto, se estuvo estudiaron las distintas herramientas y tecnologías que podían resultar de utilidad. El principal objetivo era estudiar las distintas ventajas y posibilidades de cada una de las tecnologías y poder ver cuál de ellas resultaba más conveniente para el proyecto. Este estudio ayudó a aclarar la forma en la que se enfocó el proyecto.

Tras finalizar el estudio, se obtuvo una lista de tecnologías o herramientas que serían utilizadas durante el resto del proyecto. Estas herramientas pueden ser divididas en 3 categorías: Herramientas para la ingestión de datos, análisis de los datos y creación del modelo, herramientas para la creación de la página web y herramientas auxiliares.

4.2.3.1 Herramientas para la ingestión, análisis de datos y creación del modelo

Todas las herramientas utilizadas en esta fase tienen como base el lenguaje de programación de Python.

> "Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la programación orientada a objetos, la programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma". (colaboradores de Wikipedia, 2021)

Una de las principales ventajas de Python es que tiene muchas librerías y módulos que ayudan a realizar tareas como el procesamiento, el análisis y la visualización de datos. Además, tiene librerías que permiten realizar proyectos de Machine Learning. Estas características han hecho que Python sea uno de los lenguajes más utilizados para este tipo de proyectos. Se decidió utilizar Python ya que este lenguaje proporciona todas las herramientas necesarias para la primera parte del proyecto.

Algunas de las librerías utilizadas de este lenguaje fueron:

- Tensorflow: "Tensorflow es una biblioteca de código abierto para el cálculo numérico y el aprendizaje automático a gran escala" (Yegulalp, 2019). Esta librería fue utilizada junto con Keras para la creación de los modelos predictivos.
- Keras: Es una biblioteca de Redes Neuronales de código abierto escrita en Python. Está diseñada para la experimentación con redes neuronales de Aprendizaje Profundo. En el proyecto es utilizada junto con TensorFlow para generar los modelos predictivos.
- Pandas: "Pandas es una herramienta de manipulación y análisis de datos de código abierto, rápida, potente, flexible y fácil de usar, construido sobre el lenguaje de programación Python". (Pandas, s.f.). Esta librería resulta imprescindible a la hora de trabajar con grandes cantidades de datos en Python. Permite manipular de una forma muy sencilla y rápida las estructuras de datos (DataFrames).
- Matplotlib: Es una librería utilizada para la creación de gráficas en Python. Fue utilizada para visualizar los resultados obtenidos y poder evaluarlos. Algunas de las gráficas generadas se encuentran en este documento.

Estas fueron las principales librerías utilizadas en el proyecto. Se hizo uso de alguna librería adicional como “datetime” o “os”, pero su relevancia en el resultado final del trabajo no es tan significativa como las presentadas anteriormente.

4.2.3.2 Herramientas para la creación de la página web

El segundo módulo de proyecto es la página web. En esta página web se representan los resultados obtenidos de los modelos predictivos del tráfico. Para crearla, se hizo uso de las siguientes herramientas:

- React: “Es una biblioteca JavaScript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página”. (colaboradores de Wikipedia, 2021). Una de las mayores características de React es el uso de componentes. Gracias a estos, se pueden renderizar partes específicas de la pantalla con el contenido de los componentes.
- React Bootstrap: Es una librería para React que ayuda a crear páginas web responsive. Además, tiene componentes personalizados como pueden ser botones, tablas, gráficas, entre otros, para ayudar en el diseño de páginas web. En el proyecto se hizo uso de estos componentes. Algunas de las gráficas generadas en la página web se realizaron gracias a esta librería.
- CSS: Es un lenguaje de hojas de estilo en cascada que se utiliza para personalizar y dar estilo a los elementos de una página web. Fue utilizado para personalizar elementos de la página web como puede ser el fondo, el tamaño y color de la letra o la posición de ciertos elementos en la pantalla.
- ChartJS: Es una biblioteca de JavaScript que permite generar gráficos de distintos tipos (barras, líneas, burbujas…). En el proyecto es utilizada como recurso para ayudar en la generación de gráficas con los datos generados por los modelos.

4.2.3.3 Herramientas auxiliares

Las herramientas auxiliares en el proyecto son aquellas herramientas que han servido como soporte para la realización del proyecto.

- Visual Studio Code: Fue el editor de código con el que se trabajó. Este editor permite la instalación de múltiples extensiones que ayudan en la realización de código.
- Jupyter Lab: Es una interfaz que permite ejecutar partes de código por bloques. Esto es muy útil a la hora de trabajar con temas relacionados con la ciencia de datos, ya que no es necesario tener que compilar nuevamente todas las gráficas y modelos al añadir código nuevo.
- NPM: Es un sistema de gestión de paquetes. Es utilizado para ejecutar el servidor en local de la página web creada con React.
- GitHub: Es una herramienta que permite crear repositorios en la nube en los que se pueden alojar los proyectos software creados. Ha sido de gran utilidad para tener una copia de seguridad del proyecto en todo momento.
- CSV: “Los archivos son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea”. (colaboradores de Wikipedia, 2021). Este es el formato en el que se almacenarán los datos recogidos de las distintas fuentes de datos, ya que resulta muy sencillo cargarlos y manipularlos en Python.
- JSON: “Es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web”. (Mozila, 2021). Este formato fue utilizado para representar las predicciones del modelo en la página web creada.

4.2.3.4 Otras herramientas estudiadas

Existen algunas tecnologías que finalmente no fueron escogidas por una u otra razón para este proyecto, pero que merecen ser mencionadas:

- R: Es un lenguaje de programación con un fuerte enfoque para el análisis estadístico. No fue escogido debido a que Python, además de ofrecer

herramientas similares a R en cuanto a análisis estadístico, también tiene otras librerías muy distintas que eran de utilidad para el proyecto.

- Spark: “Es un motor ultrarrápido para el almacenamiento, procesamiento y análisis de grandes volúmenes de datos” (Sierra, 2019). Pese a su gran velocidad, no se utilizó esta herramienta debido a que para el volumen de datos con el que se trabajaba, no hubiese habido una gran diferencia de rendimiento entre realizar el proyecto completamente en Python o utilizar Spark.
- Hadoop: “Proporciona un almacenamiento masivo para cualquier tipo de datos, un enorme poder de procesamiento y la capacidad de manejar tareas o trabajos prácticamente ilimitados”. (R.PowerData, 2017). En la actualidad, esta herramienta ofrece un menor rendimiento que Spark, por lo que fue descartada.

4.3 Recolección de datos

Una vez estudiadas las distintas herramientas a utilizar y las fuentes de información, se procedió a realizar la primera fase del proyecto, que fue la recolección de los datos. Debido a que la idea del proyecto era realizar predicciones en tiempo real, era necesario un histórico de datos para entrenar al modelo y los datos en tiempo real para realizar las predicciones.

4.3.1 Obtención de datos

Como se comentó anteriormente, la obtención de datos fue realizada en dos partes. Por un lado, era necesario obtener los datos históricos para poder entrenar los modelos predictivos. Por otro lado, era necesario la obtención de datos en tiempo real de estas mismas fuentes de datos para aplicarlas a los modelos y obtener las predicciones del estado del tráfico a futuro. Para realizar operaciones con grandes cantidades de datos como puede ser eliminar una columna o aplicar alguna transformación a una columna en particular, se hizo uso de la librería de Pandas.

4.3.1.1 Datos históricos

En primer lugar, se realizó el trabajo para la obtención de datos históricos. Además, estos datos servirían para poder estudiar y seleccionar aquellas características que pudiesen ser de utilidad a la hora de entrenar los modelos. En una primera instancia se planteó la posibilidad de obtener los datos históricos mediante descargas directas de las propias fuentes de datos. Esta idea fue descartada ya que este proceso sería más complejo de automatizar, además de que las descargas de estos archivos podrían demorarse mucho (los datos de tráfico ocupaban aproximadamente 17Gb).

El enfoque que finalmente se utilizó fue el de hacer uso de las propias API (véase apartado 4.2.2) para ir realizando consultas sobre datos pasados. Una ventaja de este enfoque es que, para los datos de tráfico, a través de la API se puede realizar un filtrado de los datos para traer únicamente aquellos datos que fuesen a ser utilizados. Un ejemplo de este filtrado es la decisión de utilizar únicamente los datos del distrito de Manhattan. Esta decisión nace a raíz de las características de los datos de las API de clima y calidad del aire. En ambas fuentes de datos el origen de las mediciones es en el distrito de Manhattan. Los datos o mediciones de un distrito podrían no ser iguales para el resto de los distritos, lo que podría llevar a utilizar datos erróneos para entrenar modelos. Es por ello por lo que se decidió utilizar únicamente los valores pertenecientes a Manhattan. Gracias al uso de las API para realizar solicitudes sobre datos históricos, se pudo filtrar la descarga únicamente para los valores de este distrito. Además, el filtrado ofrece una gran flexibilidad a la hora de realizar las consultas ya que permiten descargar los datos que pertenezcan a un intervalo de tiempo determinado. De esta forma, no era necesario realizar la descarga de todos los datos para todas las fechas y distritos de Nueva York.

Una de las características de la API del clima es que sólo se podía acceder por cada consulta a datos de 30 días consecutivos. No todos los meses cuentan con 30 días, lo que podría llevar a problemas a la hora de descargar los datos de meses con 31 días. La forma de solucionar este problema fue realizar las consultas cada 15 días. Para todos los meses se realizaría una consulta desde el día 1 al 15 de ese mes. La segunda consulta de cada mes dependería del mes del que se tratase. Para meses con 31 días, realizaría otra consulta para los días 16-31, mientras que para meses con 30 días se realizaría la consulta desde el día 16 hasta el 30. En el caso del mes de febrero, las consultas están adaptadas para aquellos años que sean bisiestos. Si el año fuese bisiesto, la consulta de febrero se realizaría desde el 16 al 29, mientras que si no lo

fuese haría la solicitud desde el día 16 hasta el 28. De esta forma, para cada mes se generarían 2 archivos de datos.

Para cada consulta, se guardan los datos del tráfico, la calidad del aire y el clima por separado y se genera una unión de los datos de esa consulta. La unión se realiza por fecha y hora, de tal forma que se tengan los valores de la misma hora para el tráfico, la calidad del aire y el clima. Como resultado se tiene una unión de datos de las 3 API por cada consulta. Una vez realizadas todas las consultas, queda por realizar la unión de todos los pequeños archivos en un archivo único. Para esta unión se diseñó un método para unir todos los archivos que estuviesen en un mismo directorio. Debido a que todas las consultas se almacenan en un mismo directorio, basta con llamar al método en cuestión con la carpeta en la que se encuentran descargados los datos. El resultado es un archivo único que cuenta con todos los datos históricos a partir de la fecha deseada. Por defecto se utilizaron todos aquellos valores comprendidos entre el 1 de julio de 2019 y el 30 de junio de 2021 para entrenar los modelos. A continuación, se muestra el resultado de la solicitud de datos históricos junto con el archivo de unión de todos ellos.

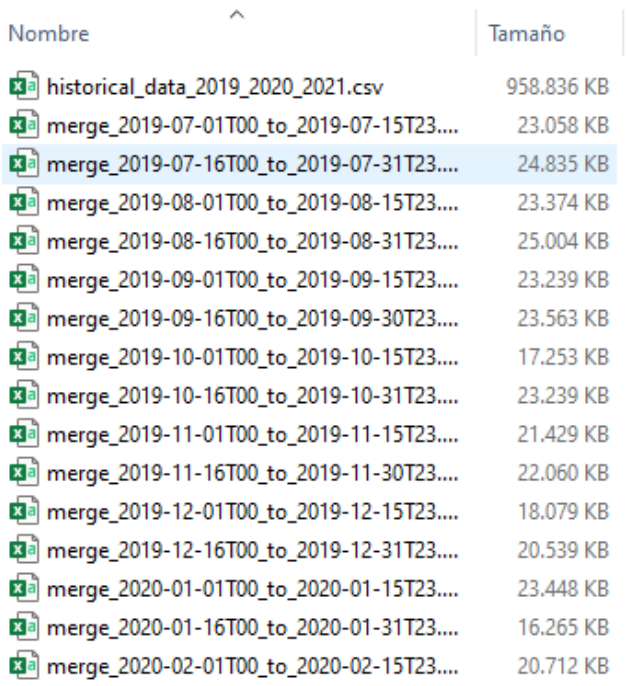

Ilustración 4.1: Histórico de datos

4.3.1.2 Datos en tiempo real

La recolección de datos en tiempo real es una adaptación del código utilizado para generar los datos históricos. En este caso, no se solicitan datos en un rango concreto de tiempo, sino que se solicitan los datos a las API de la hora actual. Un problema para realizar las peticiones en tiempo real es que se necesitan hacer peticiones a 3 API distintas al mismo tiempo. Es por eso por lo que se tuvo que trabajar con computación concurrente. La concurrencia es la habilidad que tiene un computador de realizar varias tareas a la vez (Escobar, 2017).

En este caso se crearon 3 hilos, uno para cada una de las API mediante la librería de Python de “threading”. A continuación, se va a explicar en más detalle el funcionamiento de cada uno de estos hilos:

El primero de los hilos creados es el hilo para la recogida de datos de tráfico. Puesto que la recogida de datos depende de los datos obtenidos sobre el tráfico, este será el hilo cuya rutina será ejecutada con más frecuencia (5 minutos). Una vez se ejecuta esta rutina, se realizan peticiones a la API de tráfico para la hora en la que el programa fue ejecutado. Si se han registrado datos nuevos, esta API automáticamente guardará estos datos en el documento de unión. Si esto ocurre, para las siguientes iteraciones la fecha utilizada para realizar las peticiones será la última fecha que se encuentre en el documento de unión. De esta forma se podrán ir obteniendo todos los datos de tráfico en tiempo real. En el caso en el que no se hayan registrado datos nuevos, el documento de unión no será modificado.

La rutina encargada para la recogida de datos en tiempo real del clima funciona de forma idéntica a la rutina para la recogida de datos en tiempo real de la calidad del aire. En el momento en el que se ejecuten los hilos de estas rutinas, se harán peticiones para la hora en la que se ejecutó el programa. Los valores de clima y calidad del aire obtenidos no se insertan en el documento de unión hasta que se reciban todos los datos de tráfico para la hora consultada. Por tanto, no se obtiene una unión de datos para una hora completa hasta que estén todos los datos del tráfico para esa hora. Una vez se realiza la unión de los datos, las API realizan peticiones para la hora siguiente y así sucesivamente.

Puesto que la unión de los datos de clima y calidad del aire dependen de tener todos los datos de tráfico para una hora dada, no es necesario que la frecuencia de

ejecución de las rutinas de calidad del aire y clima sea tan alta como la de la rutina de tráfico.

A continuación, se van a mostrar unos gráficos explicativos sobre el funcionamiento de la recogida de datos en tiempo real.

1. Archivos en el momento inicial. Se cuenta con 3 archivos. Uno con la unión de todos los datos y en donde se irán añadiendo los datos del tráfico junto con su fecha y hora. Los otros 2 archivos contienen los valores de calidad de aire y clima.

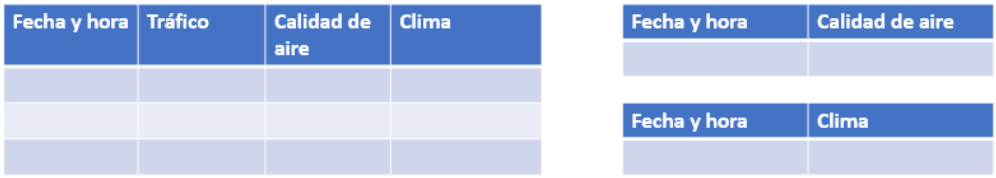

Ilustración 4.2: Archivos en el momento inicial

2. Una vez se ejecute el programa para la recolección de datos en tiempo real, se irán actualizando los valores para esa hora de tráfico, calidad de aire y de clima en sus respectivos archivos. Los datos de tráfico se irán añadiendo al documento de unión de forma automática.

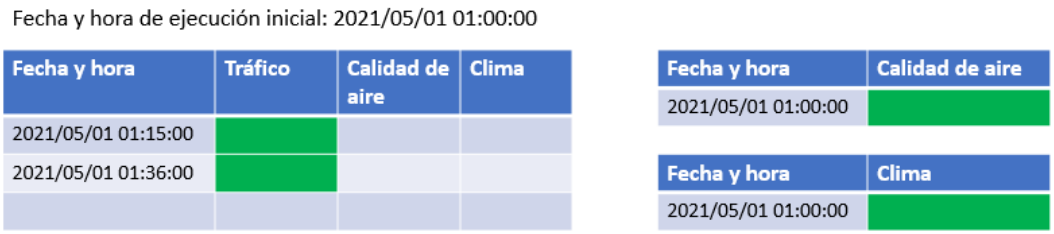

Ilustración 4.3: Se cuenta con los datos de las API, pero no se realiza la unión

En este caso, puesto que no se ha terminado la hora y, por tanto, pueden aparecer nuevos valores de tráfico para esa hora, no se realiza la unión con los datos de calidad de aire y clima.

3. Para este siguiente ejemplo, los datos de tráfico para la hora de ejecución ya han terminado puesto que el último valor de tráfico ya pertenece a la siguiente hora. En este caso, sí se realiza la unión de los datos de calidad de aire y clima.

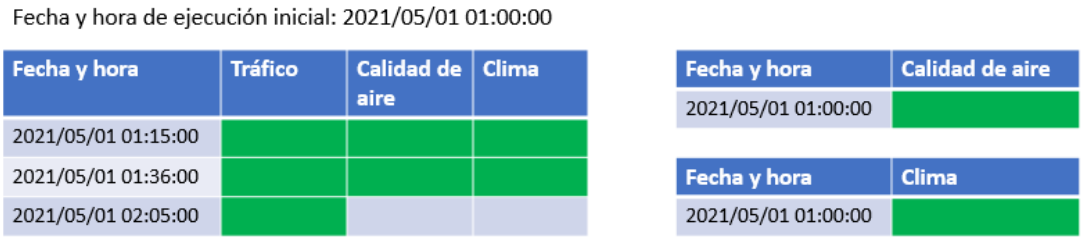

Ilustración 4.4: Unión de todos los datos para la hora

Como se ha realizado la unión de los datos, los hilos encargados de traer los datos de la calidad del aire y el clima actualizan su hora para traer datos de la hora siguiente y se vuelve al estado de la segunda imagen.

4.3.2 Análisis de los datos

Antes de empezar a realizar modelos predictivos se tuvo que hacer un estudio de los datos obtenidos por las fuentes de información, para determinar si ciertas variables podían ser de utilidad o no. Este estudio fue posible gracias a que los datos recogidos por las API son estructurados. Es decir, están sujetos a un formato muy concreto.

Las variables pueden ser clasificadas de muchas maneras. Por ejemplo, se pueden clasificar variables dependiendo del tipo de datos que son (float, int, string…). Conocer los tipos de cada una de las variables ayuda a la hora de trabajar con los datos, puesto que se conoce la naturaleza de estos y las posibles transformaciones que se le pueden aplicar.

Una de clasificaciones más importantes a la hora de trabajar con datos es la clasificación entre atributos continuos y atributos categóricos:

- Un atributo continuo es un atributo que tiene un rango infinito de valores posibles. La velocidad a la que pasa un coche es un atributo continuo. Puede adoptar un rango infinito de posibles valores.
- Un atributo categórico es un atributo que tiene un número finito de posibles valores. Los días de la semana son un claro ejemplo de atributos categóricos, puesto que el atributo solo puede tomar uno de 7 valores.

Otro ejemplo sería el mes del año. En este caso este atributo sólo podría tomar uno de 12 valores.

Es muy importante identificar si un atributo es categórico o continuo para poder realizar las modificaciones adecuadas a los atributos categóricos antes de entrenar modelos predictivos. La importancia reside en que en general no se puede trabajar de forma directa con datos categóricos para predecir valores continuos. Esto es debido a que, en el caso de valores categóricos, en muchas ocasiones estos valores pueden ser cadenas de caracteres. Por ejemplo, se puede tener una variable categórica con los días de la semana: “lunes”, “martes”, … Estos valores no pueden ser aplicados directamente a un modelo, por lo que es importante conocer la naturaleza de estos datos para transformarlos en valores numéricos. (Por ejemplo, el 1 que haga referencia al lunes, el 2 que haga referencia al martes y así sucesivamente).

A continuación, se muestran las variables utilizadas para entrenar los modelos de cada una de las API junto con su tipo de variable, significado y si se trata de un atributo categórico o continuo:

Variables de la API del tráfico:

Nombre de la variable	**Descripción**	**Tipo de variable**	**Continuo / Categórico**
Datetime_traffic	Es la hora en la que los datos fueron recogidos. Esta columna pasó a llamarse datetime_traffic en el conjunto de datos final.	Datetime	Continuo
Speed	Velocidad media a la que circularon los coches durante últimos 5 minutos en esa calle. (km/h)	Float	Continuo
travel_time	Tiempo medio que tardó un coche en pasar por esa calle. (segundos)	Int	Continuo
Link_name	Nombre de la calle o referencias sobre la localización del sensor.	String	Categórico

Tabla 1: Variables de la API del tráfico

Variables de la API de la calidad de aire:

Nombre de la variable	**Descripción**	**Tipo de variable**	**Continuo / Categórico**
datetime	Fecha y hora de la medición de los datos de la calidad del aire.	Datetime	Continuo
AQI_PM2.5	Índice de calidad de aire en cuanto a partículas PM2.5	Float	Continuo
Value_PM2.5	Concentración de partículas PM2.5 en el aire ($\mu g/m^3$)	Float	Continuo
AQI_OZONE	Índice de calidad de aire en cuanto a partículas de ozono (O_3)	Int	Continuo

Value_OZONE	Concentración de partículas de ozono (O_3) en el aire ($\mu g/m^3$)	Float	Continuo

Tabla 2: Variables de la API de la calidad de aire

Variables de la API del clima:

Nombre de la variable	**Descripción**	**Tipo de variable**	**Continuo / Categórico**
Datetime	Fecha y hora de la medición de los datos del clima	Datetime	Continuo
Temperature	Temperatura (en grados Fahrenheit).	Float	Continuo
Relative Humidity	Humedad Relativa en el aire (porcentaje)	Float	Continuo
Precipitation	Cantidad de precipitación en el momento (mm)	Float	Continuo
Snow Depth	Profundidad de la nieve (en pulgadas)	Float	Continuo
Visibility	Visibilidad (en millas)	Float	Continuo
Conditions	Condiciones generales (por ejemplo: “parcialmente nublado”, “soleado” …)	String	Categórico

Tabla 3: Variables de la API del clima

A la hora de trabajar con valores categóricos, se puede realizar de dos maneras:

- Transformar de forma aleatoria todos los valores categóricos a números
- Almacenar un diccionario que tenga el valor original de cada uno de los valores categóricos y utilizarlos para transformar los valores categóricos a los valores establecidos en el diccionario.

Para este trabajo se utilizaron ambos enfoques. Por un lado, para aquellos valores categóricos como “conditions”, puesto a que el valor al que se transformase cada valor categórico no importaba, se realizó aleatoriamente.

En el caso de “link_name” ocurre lo contrario. Este valor es necesario conocerlo, ya que será utilizado tanto en los modelos como en la página web para diferenciar una calle de otra. Por ese motivo, se creó un diccionario de valores que contenía el nombre de las 26 calles de las que se realizan mediciones y su transformación a valor numérico. De esta forma, cuando se desee acceder al valor de cada calle, se puede buscar el nombre original de la calle en el diccionario.

Las tablas anteriores muestran las variables que fueron utilizadas de cada una de esas API, pero estas fuentes de información proporcionaban muchos más valores que fueron descartados tras considerar o probar que no eran útiles a la hora de realizar predicciones sobre el estado del tráfico. Algunas de estas variables eliminadas fueron las siguientes:

- Presión a nivel del mar
- Dirección del viento
- Punto de rocío
- Sensación térmica

Se consideró que estas variables no tienen la misma relación que las otras columnas escogidas a la hora de influir en la velocidad de los coches en la ciudad. Por ejemplo, la presión a nivel del mar no debería tener ningún impacto en la velocidad a la que circulan los coches dentro de una ciudad. Este tipo de variables fueron descartadas utilizando la lógica para trazar correlaciones entre la propia variable y los factores que puedan influir en el estado del tráfico. Algunas de las variables fueron descartadas tras aplicar los modelos y observar que empeoraba el error en las predicciones. A continuación, se procede a crear una tabla con la justificación del uso de cada una de las variables mostradas anteriormente:

Nombre de la variable	Justificación de uso
Datetime/ Datetime_traffic	Esta variable es indispensable para realizar la unión de los datos de distintas fuentes. Mediante el uso de esta columna, se puede realizar la unión de los datos cuyas horas coincidan. Adicionalmente, en la creación del modelo final se utilizó esta variable para representar el tiempo como señales de seno y coseno.
Speed	La velocidad a la que circulan los coches es una variable muy importante a la hora de medir el tráfico en una calle. Si un coche circula por una calle a menor velocidad, existe una mayor probabilidad que sea porque hay un gran volumen de vehículos circulando por esa calle.
Travel_time	Al igual que para la velocidad, el tiempo medio que tarda un coche en circular a lo largo de una calle es muy indicativo del estado de la carretera. Si el tiempo en cruzar / recorrer la calle es alto, existe una mayor probabilidad que sea porque el tráfico en esa calle se encuentre congestionado.
Link_name	Es necesario separar las distintas calles para analizar las características de cada una de ellas, y con ello realizar los modelos, puesto que el estado el tráfico en cada calle puede variar de una a otra. Por ejemplo, no se puede esperar el mismo tráfico en una calle a las afueras de la ciudad que en una calle central de Manhattan.
AQI_PM2.5	La calidad del aire tiene una correlación directa con el estado del tráfico. Si hay un mayor número de coches circulando por la calle, sus emisiones generarán un mayor número de partículas finas (PM2.5), lo que empeorará la calidad del aire. AQI es una escala que resulta bastante útil a la hora de entrenar modelos que hagan uso de variables relacionadas con la calidad del aire.
Value_PM2.5	El valor exacto de las partículas PM2.5 sirve también de ayuda a la hora de entrenar modelos predictivos. Para continuar con

	la relación entre las partículas PM2.5 y el tráfico, artículos científicos aseguran que “se ha demostrado que las fuentes de emisiones vehiculares contribuyen a las concentraciones elevadas de contaminación del aire en el entorno cercano a la carretera, incluidas las PM2.5” (Mohammad Hashem Askariyeh, 2020)
AQI_OZONE	Al igual que con el PM2.5, el ozono en la calidad del aire puede variar en aquellas ciudades cuyo tráfico sea más elevado. Además, en la creación de modelos se obtuvo buenos resultados utilizando variables relacionadas con el ozono.
Value_OZONE	El valor exacto del ozono resultó de utilidad también a la hora de crear modelos para la predicción del estado del tráfico. Un estudio afirma que “los aumentos en la concentración de ozono en la superficie como resultado del tráfico rodado son típicamente del 5% al 15% en latitudes medias del hemisferio norte”. (Brasseur, 2003).
Temperature	La temperatura puede verse influida por un mayor número de coches. Debido a que los motores de los coches generan calor, si se da el caso de tener un gran número de vehículos circulando en una ciudad, la temperatura podría aumentar. Es por ello por lo que esta variable fue utilizada.
Relative Humidity	A priori, el tráfico no parece tener mucha relación con la humedad relativa, pero, tras realizar un estudio con artículos científicos se encontró una correlación. La humedad relativa puede verse influenciada por las partículas generadas en las emisiones de los motores de los vehículos. (Milán Jamriska, 2008). Este artículo, además de una mejoría en los resultados obtenidos para los modelos fue el motivo por el que se utilizó esta variable.
Snow Depth	Tener un indicador sobre la nieve en la ciudad tiene relación directa con el tráfico, ya que en el caso en el que haya nieve es muy probable que los coches circulen a menor velocidad, lo que puede empeorar el estado del tráfico.
Visibility	La visibilidad es un factor importante, que puede tener especial relevancia en las carreteras principales o autopistas. En este tipo de carreteras es importante tener una buena visión para poder mantener la distancia de seguridad con los vehículos. Si la visibilidad disminuye, es probable que los coches circulen a una menor velocidad para poder tener un mayor tiempo para reaccionar en el caso en el que ocurra algún incidente, ya que, si un vehículo circula a alta velocidad, el tiempo de frenada de un vehículo sería mayor que circulando a una velocidad inferior.
Conditions	Este valor es de utilidad ya que muestra las condiciones generales en la ciudad. Por ejemplo, el valor de esta variable podría ser “soleado” o “lluvioso”. Este tipo de variables más generales pueden ser de utilidad a la hora de entrenar el modelo predictivo.

Tabla 4: Justificación de uso de cada variable

Además, algunas de las variables que aparecen en las tablas fueron utilizadas para generar columnas nuevas que pudiesen aportar información adicional:

- “Weekday”: Esta columna fue creada utilizando la columna de “datetime” que contiene la fecha y la hora del registro. Utilizando el método “day_name()” de la librería de datetime se obtiene el nombre en inglés del día de la semana. Este atributo es continuo ya que sólo puede tomar uno de 7 valores.
- “Zone”: Este fue un atributo creado en el que se dividió Manhattan en cuadrantes a los que se llamaron zonas. Dependiendo de las coordenadas de la calle, se le asignó la zona a la que perteneciese. Para crear y asignar las zonas se utilizó un atributo de la API de tráfico llamado “link_points”. Este atributo es una ristra con las coordenadas de cada uno de los puntos de referencia de la calle. El atributo fue utilizado únicamente para ayudar en el proceso de división de las calles por zonas. Estos datos se trabajaron haciendo una media de las coordenadas de la calle con la finalidad de tener una aproximación de la calle en un único punto.

Utilizando el valor máximo y el mínimo entre todas las coordenadas resultantes se creó una matriz en la que se encontraban todas las coordenadas de las calles. Esta matriz a su vez fue dividida para crear las zonas. Finalmente, se asignó la zona correspondiente a cada una de las calles. A continuación, se muestra la representación de las coordenadas y la división por zonas junto con un mapa de Manhattan:

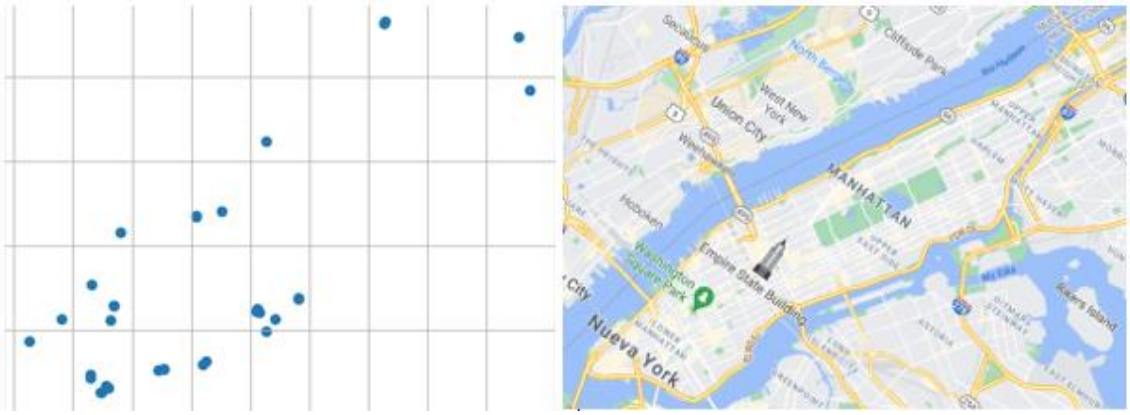

Ilustración 4.5: Representación de Manhattan por zonas

La cuadrícula de la gráfica de la izquierda marca la división por zonas. La forma de asignar zonas fue mediante las coordenadas de esta nueva matriz. Es decir, si una calle dentro de la matriz se encuentra en el punto (4,3), la zona a la que será asignada será 43. Como se puede observar, la representación de los puntos en el mapa se asemeja bastante bien al mapa real de Manhattan.

El objetivo con este atributo era hacer predicciones por zonas, en la que cada zona podía incluir una o más calles que podrían estar o no conectadas entre sí. Finalmente, se decidió no utilizar este atributo, ya que en una misma zona puede haber dos calles en las que se realicen mediciones que no estén conectadas entre sí. Esto podría dar lugar a predicciones erróneas, ya que el estado de esas dos calles podría variar significativamente. Por ejemplo, en una misma zona puede haber 2 calles no conectadas entre sí. Una de estas calles podría ser una calle principal y la otra una calle secundaria. El estado del tráfico en cada una de las calles podría ser muy distinto al tráfico de la otra, lo que resultaría en datos menos fiables y, por tanto, se generarían peores predicciones.

- "relative_speed": El mejor indicador sobre el estado del tráfico que se obtiene de la API del tráfico es la velocidad a la que circulan los coches. La lógica para este atributo sería que, si la velocidad registrada es baja, hay una mayor probabilidad de que el estado del tráfico en esa calle se encuentre congestionado o que haya alguna incidencia que lo ralentice. Utilizar la velocidad como indicador del tráfico puede ser bastante problemático, ya que cada calle tiene su límite máximo de velocidad. Es decir, no es lo mismo registrar un valor de 45km/h en una calle que tiene como máximo 50km/h a registrar 45km/h en una calle que tenga como máximo 120km/h. Es por ello por lo que se crea el término de velocidad relativa, que sigue la siguiente fórmula:

$$velocidad\ relativa = \frac{velocidad\ registrada}{velocidad\ máxima\ de\ la\ calle}$$

El valor de este atributo suele oscilar entre 0 y 1 pero se puede dar el caso en el que un coche circule a una velocidad superior a la permitida en esa calle. En ese caso, el valor será superior a 1. El objetivo con este atributo es utilizar esta proporción para poder comprender en un mayor grado el estado del tráfico y ser capaz de comparar el estado del tráfico entre calles que tengan límites de velocidad distintos.

Para poder realizar este cálculo hace falta conocer los límites de velocidad de las calles de Nueva York. Esta información no está disponible en ningún tipo de API para poder ser extraída de forma automática, por lo que esta parte del trabajo tuvo que realizarse a mano. Una de las fuentes de datos utilizadas fue un documento de la página oficial de la ciudad de Nueva York (NYC Government, 2015). Este documento proporciona una imagen con la velocidad máxima por calle:

Ilustración 4.6: Velocidad máxima de Manhattan por calle (NYC Government, 2015)

Como se pudo observar en la representación de los puntos de medición de las calles (véase ilustración 4.5), prácticamente la totalidad de las calles tienen su velocidad máxima representada en la ilustración superior, por lo que se pudo asignar de forma correcta la velocidad máxima para cada calle.

Una vez se obtuvo la velocidad máxima de cada calle, se transformó este valor de millas por hora (mph) a kilómetros por hora (km/h), ya que el atributo de velocidad de la API de tráfico tiene como unidad km/h. Una vez añadida la nueva columna con la conversión, se pudo empezar a trabajar con la velocidad relativa. Este atributo es fundamental ya que será el valor de salida de los modelos predictivos.

4.4 Selección del modelo

En un comienzo, para realizar el modelo se planteó la opción de realizarlo como un problema de regresión lineal o clasificación.

> "La regresión lineal es una técnica de modelado estadístico que se emplea para describir una variable de respuesta continua como una función de una o varias variables predictoras. (…) Las técnicas de regresión lineal permiten crear un modelo lineal. Este modelo describe la relación entre una variable dependiente como función de una o varias variables independientes. " (Mathworks, s.f.)

El motivo que indujo a pensar en utilizar este modelo fue debido a que el tráfico podría depender en buena medida de otras variables como puede ser la precipitación o la contaminación, por ejemplo. Además, el valor de salida del modelo sería un valor numérico (velocidad relativa), por lo que un modelo de regresión lineal encajaba con el contexto del problema.

Otra forma de plantear el desarrollo del modelo sería mediante la discretización de la variable de salida. Esto es, dividir la velocidad relativa (la variable de salida) en categorías y realizar el problema de regresión lineal. También se puede realizar el modelo de regresión lineal primero y discretizar los resultados. Un ejemplo de discretización o categorización sería:

- 0 – Para todo aquel valor cuya velocidad relativa sea inferior a 0,5. Esto representaría el tráfico en su estado de congestión.

- 1 – Para todo aquel valor cuya velocidad relativa sea igual o superior a 0,5. Esto indicaría que el estado del tráfico en ese momento es bueno.

El valor categórico no tiene por qué ser un número en el caso que se realice la categorización tras realizar la predicción, pero en caso de realizar la categorización antes de realizar el modelo, es necesario que sean números ya que los modelos no pueden entrenar valores que no sean numéricos.

A este tipo de modelos se le podría aplicar los algoritmos de Lasso y Ridge. Estos algoritmos ayudan a evaluar el peso que tiene cada una de las columnas en los resultados. De esta forma, se podría eliminar aquellas columnas que no tuviesen ningún impacto. Las diferencias entre Lasso y Ridge es que Lasso pone a 0 el peso de todos aquellos valores con poco impacto, mientras que Ridge reduce el peso de las columnas con menos impacto, pero sin llegar nunca a 0.

Ambas formas de afrontar el problema fueron probadas y evaluadas, pero el modelo resultó no ser el adecuado para el proyecto. Esto es debido a que las predicciones dependen de los valores del resto de columnas. A la hora de realizar pruebas a futuro, no se cuenta con los valores a futuro de las variables de las que dependa el modelo, como puede ser el AQI_PM2.5 o la precipitación, por lo que no se podría realizar predicciones a futuro. Se podría hacer uso de páginas o API que proporcionasen predicciones, pero eso significaría utilizar las predicciones de un modelo desconocido, que podría resultar en valores muy erróneos y dependientes en la calidad de esas predicciones. Es por ello por lo que se decidió dejar de trabajar con los modelos de regresión lineal.

Tras descartar el modelo de regresión lineal, se planteó la realización de las predicciones mediante el uso de series temporales.

> “Una serie temporal (o cronológica) conforma un conjunto de datos u observaciones que han sido medidas en determinados momentos y que, además, han sido ordenados cronológicamente. Dichos datos pueden encontrarse espaciados a intervalos iguales o diferentes. (Conecta Software, 2020)

Tal y como explica Conecta Software, 2020, las series temporales tienen 2 características fundamentales:

- El orden de los datos importa
- Los datos del pasado importan en el futuro

Son estas dos características las que determinan que el uso de series temporales es adecuado para resolver el problema del estado del tráfico. Para determinar el estado del tráfico en un futuro, es necesario conocer los datos de pasado. Es decir, el tráfico dentro de 1 hora está muy influenciado por cómo ha estado, por ejemplo, las 2 últimas horas. Por esto mismo, el orden de los datos importa. De nada sirve tener los datos sin ordenar, puesto que representaría una tendencia de tráfico falsa. Por tanto, tras realizar un estudio de este tipo de modelo se puede concluir que las series temporales son idóneas para resolver problemas del estado del tráfico. Para realizar una parte del desarrollo de las series temporales se hizo uso de la documentación oficial de Tensorflow, que contiene una guía para la realización de series temporales. (Tensorflow Team, 2021)

4.4.1 Intervalos de tiempo

El primer paso para crear una serie temporal es decidir los intervalos de tiempo con los que se va a trabajar. Como se comentó en el apartado del estudio de las fuentes de información disponibles, la API de tráfico genera valores con intervalos de 5 minutos cada uno para cada una de las calles. Es decir, el resultado que se muestra en la medición de la velocidad de una calle en ese intervalo de tiempo no es más que la media de velocidad a la que circulan los coches que han pasado por esa calle durante ese intervalo de 5 minutos.

Teniendo en cuenta la frecuencia de los datos se plantean los siguientes intervalos de tiempo para ser utilizados como predicciones:

- 5 minutos
- 30 minutos
- 1 hora

El motivo de la elección de estos rangos es la utilidad que puedan presentar a la hora de realizar predicciones. Una predicción realizada con intervalos de 5 minutos hace que el modelo tenga un menor grado de error a la hora de generar predicciones. Esto es porque, a intervalos tan pequeños de tiempo, es muy complicado que el tráfico en una calle pueda fluctuar excesivamente.

Aunque un modelo pueda presentar mejores predicciones para intervalos muy cortos, lo más importante es la utilidad que pueda presentar para el usuario. Por un lado, un usuario no mira cómo va a estar el estado del tráfico en los siguientes 5 minutos, sino

que generalmente desean saber cómo va a estar el tráfico a horas vista. Por ello, si se desease generar predicciones para dentro de 3 horas, utilizando un intervalo de 5 minutos significaría que son necesarias un total de 36 predicciones. Generar tantas predicciones daría lugar a confusión y a una mayor probabilidad de que esas predicciones que genera el modelo tuviesen un error muy elevado. Por tanto, la opción de utilizar intervalos de 5 minutos queda descartada.

Los dos siguientes intervalos de tiempo propuestos solucionan el problema presentado anteriormente. Aunque puedan presentar un mayor grado de error a la hora de realizar predicciones, el valor que aportan es mayor, debido a que las predicciones cubren un mayor rango temporal. En este momento queda por decidir cuál de los dos intervalos es más conveniente para la realización de las predicciones. Aquí es cuando entra en juego otro de los factores, que en este caso es el tiempo que tarda la API del tráfico en actualizarse con los nuevos datos. Esta API actualiza los datos en frecuencias irregulares de tiempo. Puede darse el caso que los datos sean actualizados cada 1, 2 o incluso 6 horas durante la noche. Por tanto, muchas de las predicciones que generarían estos modelos serían a horas pasadas. Para entender este punto mejor, se va a explicar con un ejemplo:

La hora local en Nueva York es las 20:00. La última actualización de los datos de la API del tráfico fue a las 18:00. El modelo predictivo genera predicciones haciendo uso de los datos de las 18:00. Para un intervalo de 30 minutos, si el modelo predictivo desea generar predicciones a una hora superior a la hora local en Nueva York, necesitaría realizar un mínimo de 5 predicciones (entonces generaría predicciones como máximo las 22:30). Si se utiliza el intervalo de tiempo de 1 hora, bastaría con realizar 3 predicciones para tener valores a futuro (generaría como máximo valores para las 21:00). Cuanto mayor sea el retraso a la hora de actualizar los datos de la API de tráfico, más perjudicado será utilizar un intervalo de 30 minutos. Si por algún motivo los datos de la API tienen un retraso de 6 horas, el modelo de 30 minutos necesitaría un mínimo de 13 valores para tener valores a futuro, mientras que para el modelo de 1 hora le bastaría con tener 7 predicciones.

Aunque un modelo que genere valores cada hora pueda resultar tener un mayor error, aporta también un mayor valor que utilizar intervalos de tiempo menores como 5 o 30 minutos. Por tanto, se ha decidido realizar las predicciones por hora. Para ello, es necesario unificar todos los datos de cada una de las calles por hora y guardar este archivo en el directorio que se desee, que será el utilizado para entrenar los modelos.

4.4.2 Preparación de los datos: análisis de las columnas

Una vez se cuenta con los datos preparados, se puede proceder con la creación de los modelos de serie temporal. Para ello, se cargará en otro archivo el CSV creado anteriormente con los datos agrupados.

La realidad del proyecto es que no se pretende generar un solo modelo. La API de tráfico contiene datos sobre 26 calles del distrito de Manhattan. Cada una de estas calles tiene sus propias peculiaridades y tendencias en cuanto al estado de su tráfico. Por ejemplo, un túnel que conecte 2 de los distritos podría tener un peor estado de tráfico que una calle normal. Por tanto, se realizará un modelo único para cada una de las calles. Para poder realizar la decisión sobre qué modelo elegir, resulta muy complejo poder comparar los valores de 26 calles distintas para cada uno de los modelos. Por tanto, se escoge aleatoriamente una calle que será utilizada para generar los resultados con los que se elegirá el modelo a utilizar. Para asegurar que el modelo escogido era el adecuado, se cambió de calle posteriormente para comprobar si los resultados de los modelos seguían siendo parecidos.

Antes de acotar el DataFrame para que sólo contenga las estadísticas de las calles en cuestión, conviene revisar las estadísticas de las distintas columnas. Esto ayudó a comprender la naturaleza de los datos que se tienen y a distinguir de valores anómalos o erróneos que pueda haber.

	count	mean	std	min	25%	50%	75%	max
link_name	392702.0	12.511474	7.491042	0.00	6.000000	13.000000	19.000000	25.000000
weekday	392702.0	2.989636	2.002742	0.00	1.000000	3.000000	5.000000	6.000000
speed	392702.0	24.846429	15.208967	0.00	13.668333	26.092500	37.070000	80.770000
travel_time	392702.0	281.365036	627.431464	0.00	100.000000	180.083333	362.750000	169417.000000
AQI_PM2.5	392702.0	24.057509	43.050772	-999.00	15.000000	24.000000	33.000000	103.000000
Value_PM2.5	392702.0	6.241771	3.619950	-0.30	3.700000	5.700000	8.000000	36.300000
AQI_OZONE	392702.0	22.911322	12.196097	0.00	15.000000	22.000000	30.000000	108.000000
Value_OZONE	392702.0	24.510754	12.315242	0.00	16.000000	24.000000	32.000000	73.000000
Temperature	392702.0	53.468450	20.758599	-19.10	36.500000	53.100000	71.200000	99.200000
Relative Humidity	392702.0	68.803551	18.851449	15.73	55.470000	71.055000	84.780000	100.000000
Precipitation	392702.0	0.004286	0.042983	0.00	0.000000	0.000000	0.000000	3.130000
Snow Depth	392702.0	0.050087	0.326116	0.00	0.000000	0.000000	0.000000	5.120000
Visibility	392702.0	9.405284	1.376028	0.00	9.900000	9.900000	9.900000	9.900000
Conditions	392702.0	0.936224	1.331924	0.00	0.000000	0.000000	2.000000	5.000000

Ilustración 4.7: Estadísticas de las columnas del DataFrame

Como se puede comprobar, para el valor mínimo de AQI_PM2.5 se encuentra un dato atípico.

> "Un valor atípico es una observación extrañamente grande o pequeña. Los valores atípicos pueden tener un efecto desproporcionado en los resultados estadísticos, como la media, lo que puede conducir a interpretaciones engañosas (…). Es necesario investigar los valores atípicos, porque pueden proporcionar información útil sobre los datos o el proceso. " (Identificar valores atípicos - Minitab, s.f.)

El AQI_PM2.5 hace referencia a la cantidad de partículas de menos de 2,5 micras en el aire. Por la propia definición de la característica, este valor nunca puede ser negativo. Por tanto, un valor de -999.0 es un valor atípico que puede influenciar de forma negativa en el cálculo de las estadísticas para la columna, como puede ser la media. Además, para el Value_PM2.5 también se puede observar que existe un valor negativo. Nuevamente no es posible que exista un valor negativo para este campo, por lo que es considerado como valor atípico. La forma en que fueron tratados este tipo de datos fue mediante la eliminación de la fila en la que se encontrasen. El último valor atípico que se encuentra es en el "travel_time". Este valor hace referencia al número de segundos de media que tarda un coche en cruzar el sensor de inicio de la calle y el sensor al final de la calle. El valor máximo de esta columna es de 169417, lo que sería, 2823 minutos o 47 horas. Se podría poner a el valor de esos campos a 0, pero eso podría también influenciar negativamente el entrenamiento de los modelos o el cálculo de las estadísticas de la columna. Además, debido a que se trabajan con grandes cantidades de datos, no supone un problema eliminar una fila. Una vez eliminados estos valores, se comprueba nuevamente que no exista ninguna otra anomalía que pueda alterar las estadísticas o los modelos a la hora de entrenarlos. Para el "travel_time", se ha decidido eliminar todos aquellos valores cuyo travel_time sea mayor a 10000 segundos, de tal forma que se eliminen todos los valores atípicos que se encuentren para esa columna. Las estadísticas de las columnas quedarían de la siguiente forma:

	count	mean	std	min	25%	50%	75%	max
link_name	392071.0	12.511673	7.490946	0.00	6.000000	13.000000	19.000000	25.000000
weekday	392071.0	2.989576	2.003101	0.00	1.000000	3.000000	5.000000	6.000000
speed	392071.0	24.850516	15.207645	0.00	13.715833	26.093333	37.070000	80.770000
travel_time	392071.0	279.219837	349.421816	0.00	100.000000	180.000000	362.666667	7744.000000
AQI_PM2.5	392071.0	25.685863	13.707888	0.00	15.000000	24.000000	33.000000	103.000000
Value_PM2.5	392071.0	6.252005	3.613803	0.00	3.700000	5.700000	8.000000	36.300000
AQI_OZONE	392071.0	22.918051	12.201953	0.00	15.000000	22.000000	30.000000	108.000000
Value_OZONE	392071.0	24.517904	12.320565	0.00	16.000000	24.000000	32.000000	73.000000
Temperature	392071.0	53.462373	20.771837	-19.10	36.500000	53.000000	71.200000	99.200000
Relative Humidity	392071.0	68.780791	18.838452	15.73	55.450000	71.030000	84.740000	100.000000
Precipitation	392071.0	0.004287	0.043014	0.00	0.000000	0.000000	0.000000	3.130000
Snow Depth	392071.0	0.050167	0.326372	0.00	0.000000	0.000000	0.000000	5.120000
Visibility	392071.0	9.407127	1.372603	0.00	9.900000	9.900000	9.900000	9.900000
Conditions	392071.0	0.936404	1.332004	0.00	0.000000	0.000000	2.000000	5.000000
relative_speed	392071.0	0.406432	0.233627	0.00	0.244503	0.444044	0.590468	1.635934

Ilustración 4.8: Estadísticas de las columnas del DataFrame tras eliminar los valores atípicos

Como se puede comprobar, las estadísticas de las distintas columnas son más razonables. Por tanto, ya se puede acotar los datos según la calle con la que se trabaje. La razón por la que se mostraron las estadísticas de las columnas antes de acotar el DataFrame es porque al acotar el archivo se pueden omitir algunos de los valores atípicos.

Se procede a acotar el DataFrame para que solo tenga las entradas de la calle en cuestión. Se muestran los primeros valores del DataFrame para comprobar que sólo hay valores de esa calle (link_name).

```
                datetime  link_name  weekday  speed  travel_time  AQI_PM2.5  \
4    2019-07-01 00:00:00          4      1.0    0.0          0.0       10.0
30   2019-07-01 01:00:00          4      1.0    0.0          0.0       10.0
56   2019-07-01 02:00:00          4      1.0    0.0          0.0        9.0
82   2019-07-01 03:00:00          4      1.0    0.0          0.0        9.0
108  2019-07-01 04:00:00          4      1.0    0.0          0.0        7.0

     Value_PM2.5  AQI_OZONE  Value_OZONE  Temperature  Relative Humidity  \
4            2.5       13.0         14.0         76.5              78.74
30           2.4       12.0         13.0         74.8              83.07
56           2.2       12.0         13.0         74.4              82.41
82           2.2       12.0         13.0         73.5              84.68
108          1.6       12.0         13.0         73.1              83.97

     Precipitation  Snow Depth  Visibility  Conditions  relative_speed
4              0.0         0.0         9.9         0.0             0.0
30             0.0         0.0         9.9         0.0             0.0
56             0.0         0.0         9.9         0.0             0.0
82             0.0         0.0         9.9         0.0             0.0
108            0.0         0.0         9.9         0.0             0.0
```

Ilustración 4.9: Visualización de los primeros valores del DataFrame

Además, debido a que solo hay valores para una misma calle, la columna del DataFrame de “link_name” carece de utilidad, por lo que puede eliminarse.

El siguiente paso será añadir alguna columna adicional para complementar al resto y mejorar la calidad de los resultados.

4.4.3 Preparación de los datos: creación de nuevas columnas

Al haber agrupado previamente todos los valores por la calle y hora, los índices del DataFrame son estos dos valores. Eso no resulta un problema, pero se podría utilizar el tiempo para ayudar en el entreno del modelo. Debido a que el tráfico presenta un periodo claro de 1 día (de día hay un mayor tráfico y de noche normalmente disminuye), se puede representar el tiempo utilizando senos y cosenos que tengan como periodo esta frecuencia de tiempo. De esta manera, el modelo podría recibir el momento del día en el que se recogen esos datos y utilizarlos para generar mejores predicciones. Para poder convertir el tiempo en señales de senos y cosenos, primero se debe transformar la columna de “datetime” a segundos.

Tras haber convertido la hora a la que se registró cada uno de los datos a segundos, se puede realizar la transformación del tiempo a señales de seno y coseno teniendo en cuenta el periodo mencionado anteriormente de 24 horas. La fórmula aplicada para la conversión de los datos a señales de seno y coseno sería la siguiente (Tensorflow Team, 2021):

$$\text{Señal seno} = \sin\left(\frac{2\pi \times timestamp}{(24 \times 60 \times 60)}\right)$$

$$\text{Señal coseno} = \cos\left(\frac{2\pi \times timestamp}{(24 \times 60 \times 60)}\right)$$

Siendo el “timestamp” la conversión de la fecha y hora a segundos. El denominador hace referencia al número de segundos que hay en un día (24 horas, cada hora tiene 60 minutos y cada minuto tiene 60 segundos). Al aplicar esta fórmula, se obtendría una

columna con la señal en seno y coseno de la fecha. Para comprobar los resultados, podría imprimirse por pantalla el valor de algunos de los valores de las nuevas columnas creadas, pero esto no sería muy útil a la hora de interpretar los datos. Es por ello por lo que es preferible graficar los valores de estas dos nuevas columnas a lo largo del tiempo. De esta forma se podrán apreciar las curvas características del seno y del coseno:

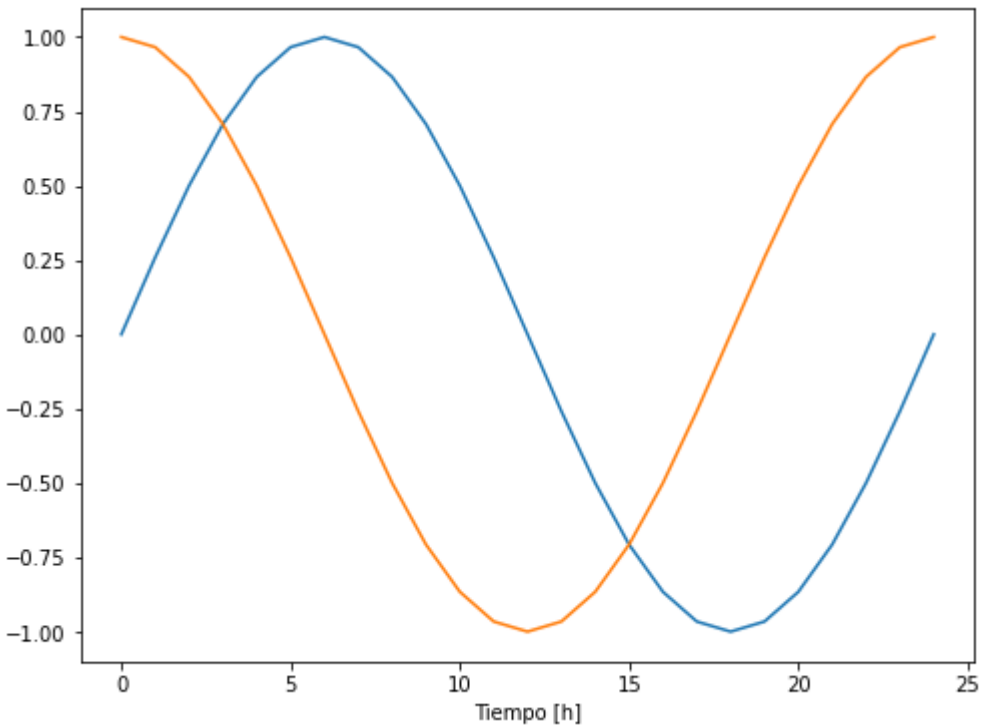

Ilustración 4.10: Representación gráfica de las nuevas columnas

4.4.4 Preparación de los datos: división de los datos

Llegados a este punto, se han creado las nuevas columnas que se utilizarán para el análisis y se ha comprobado que no tengan valores atípicos. Antes de poder empezar a definir los modelos hay que separar el conjunto de datos. Esto es debido a que, a la hora de entrenar los datos, un modelo necesita una serie de datos para entrenar los modelos y otra serie de datos para evaluar los resultados de los modelos. Para poder entrenar y hacer pruebas con los modelos para ver su eficacia, es necesario dividir el conjunto de datos. Existen 3 conjuntos de datos:

- Conjunto de entrenamiento: Los modelos utilizan estos datos para analizar las relaciones entre las variables y generar los modelos predictivos.
- Conjunto de validación: Es una fracción de los datos de entrenamiento que serán utilizados como datos de validación. El modelo no entrena con estos datos, pero evalúa la pérdida y las métricas de error con la intención de buscar aquel modelo del que se obtengan los mejores resultados. (K. Team, s.f.)
- Conjunto de prueba/test: Son datos que no han sido utilizados en ningún momento para el entrenamiento del modelo y sirven para ver el error real del modelo.

Las divisiones de datos pueden ser realizadas de dos maneras: una división que tenga datos de entrenamiento y de prueba, o una división de datos que tenga los datos de entrenamiento, validación y prueba. La primera de las opciones parece contradecir el párrafo anterior, en el que se exponían que hay 3 conjuntos diferenciados de datos. La razón por la cual se puede realizar la división de datos con un conjunto de entrenamiento y uno de prueba es que la librería de Keras automáticamente divide el conjunto de entrenamiento en el conjunto de entrenamiento y el de validación: (Nieves, s.f.)

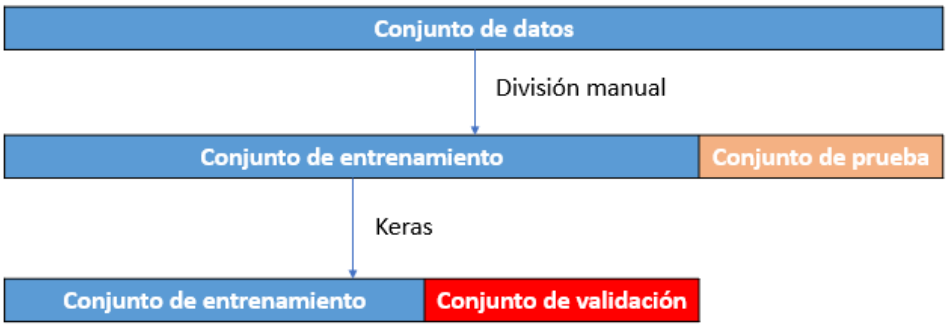

Ilustración 4.11: División automática de datos de Keras

La otra forma de realizar la división de datos sería dividir manualmente los conjuntos:

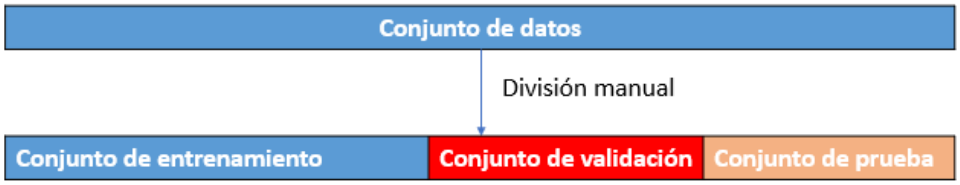

Ilustración 4.12: División manual de datos

Como los modelos que se pretenden realizar son series temporales, no se pueden dividir aleatoriamente los resultados porque son necesarias ventanas de muestras consecutivas de datos. Es por ello por lo que la división debe realizarse de forma manual, ya que Keras realiza la división del conjunto de validación de manera aleatoria.

La proporción utilizada para la división de datos puede variar. Lo importante es mantener una proporción muy alta de datos de entrenamiento, ya que son los datos que se utilizarán para entrenar el modelo. En este caso, la división escogida para los datos es un 70% para el conjunto de entrenamiento, un 20% para el conjunto de validación y un 10% para el conjunto de prueba.

4.4.5 Preparación de los datos: normalización de los datos

Además, para ayudar a que los modelos realicen mejores predicciones, es importante realizar un escalado de los datos. Una forma de escalar los datos, y la escogida para este caso, es la normalización de los datos.

Para normalizar un dato se debe restar ese valor a la media de valores de esa columna y dividir ese resultado por la desviación estándar de esa columna. La media y la desviación estándar utilizada es la del conjunto de entrenamiento. Es decir, esta sería la fórmula a aplicar para cada uno de los conjuntos:

$$dato\ normalizado\ = \frac{dato\ \mathrm{sin}\ normalizar - media\ conjunto\ de\ entrenamiento}{desviación\ estándar\ conjunto\ de\ entrenamiento}$$

Además, es importante almacenar la media del conjunto de entrenamiento y la desviación estándar del conjunto en un archivo, puesto que estos datos serán necesarios para desnormalizar los datos una vez el modelo genere predicciones.

4.4.6 Ventanas de datos

Una serie temporal hace uso de muestras consecutivas de datos para entrenar los modelos y realizar las predicciones. Una ventana de datos cuenta con una serie de características que deben ser comprendidas antes de poder proceder a la creación de los modelos que las utilizan. Para ello, se ha creado el siguiente gráfico explicativo:

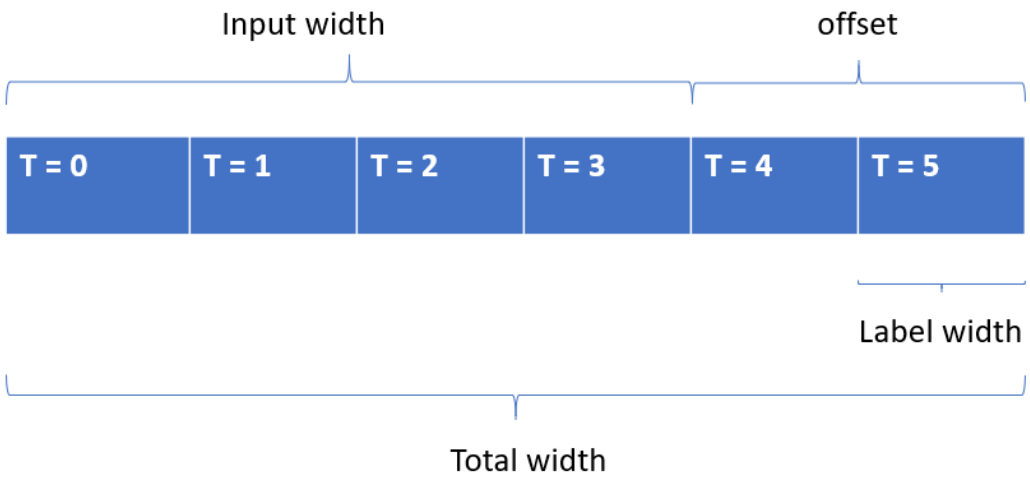

Ilustración 4.13: Representación de una ventana de datos

- Input width: También conocido como ancho o tamaño de entrada, es el número de valores anteriores que se utilizan para alimentar al modelo para poder sacar las predicciones.
- Offset: Es la cantidad de horas de desfase o tiempo de compensación que se desea entre las entradas y las predicciones.
- Label width: También conocido como el tamaño de la etiqueta, es el número de predicciones que se desea sacar.

- Total width: Es la suma del input width y del offset. Es el rango total de tiempo que utiliza la ventana de datos.
- Características utilizadas como entradas y etiquetas: Aunque no se muestra en la imagen explícitamente, las etiquetas de salida tendrán las características sobre las que se desea crear predicciones, mientras que el tamaño de la entrada contendrá todas aquellas características que serán utilizadas para ayudar a sacar esas predicciones.

Para el ejemplo del gráfico, la ventana de datos utilizaría datos de 4 horas como tamaño de entrada y generaría una predicción (label width) de 2 horas en el futuro (offset). Como el objetivo del proyecto es tener una serie de valores continuos sobre el estado del tráfico para las próximas horas, el tamaño de la ventana de salida será el mismo que el del offset para todos los modelos que se probarán. En cuanto a las características, la etiqueta de salida será el "relative_speed", mientras que el resto de las características serán utilizadas como entrada.

Queda, por tanto, implementar un método que haga uso de las características mencionadas anteriormente para ayudar a entrenar los modelos. Para ello, se hace uso de la documentación generada por TensorFlow en su guía para el pronóstico de series temporales (Tensorflow Team, 2021). En esta guía, se presenta un método llamado "WindowGenerator" que cumple con las especificaciones mencionadas anteriormente. Este método puede realizar múltiples tareas como dividir las ventanas en las etiquetas y entradas, implementar las compensaciones entre los valores de entrada y las predicciones y utilizar el tipo de datos tf.data.Dataset para crear ventanas con los datos de entrenamiento, validación y prueba. Para generar una ventana, se deberá especificar el número de valores de entrada, el número de valores de salida, el tiempo de compensación entre ambos y la columna sobre la que se desea generar esas predicciones.

Finalmente, como preparación para los modelos que se van a implementar posteriormente, se va a crear una ventana de datos que será pasada a los modelos para generar las predicciones (Tensorflow Team, 2021):

```
OUT_STEPS = 4
multi_window = WindowGenerator(input_width=12,
                               label_width=OUT_STEPS,
                               shift=OUT_STEPS)
```

El motivo por el que el label_width y el shift son el mismo valor es porque se desea generar predicciones para los primeros valores a futuro, por lo que el shift/offset será el mismo que el tamaño de etiqueta. Por tanto, los valores a modificar son el tamaño de entrada y el número de predicciones que se deseen. Estos valores serán modificados para intentar optimizar y tener el modelo ideal para el proyecto.

4.4.7 Configuración de los modelos

El último paso antes de poder comparar los modelos es generar un método para ejecutar los modelos. El método no es obligatorio si se desea trabajar únicamente con un modelo. En este caso, se pretende realizar una comparación entre modelos. Para ello, es importante asegurar que las condiciones a las que están sometidas todos los modelos sean las mismas. Así pues, resulta más sencillo y óptimo crear un método específico que controle los parámetros de ejecución de todos los modelos. Este método contendrá elementos clave para el entrenamiento como es el "early stopping" o características como la pérdida y optimizador. El método en cuestión es el siguiente (Tensorflow Team, 2021):

```
def compile_model(model, window, patience=2):
  early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                    patience=patience,
                                                    mode='min')

  model.compile(loss=tf.losses.MeanAbsoluteError(),
                optimizer=tf.optimizers.Adam(),
                metrics=[tf.metrics.MeanAbsoluteError()])

  history = model.fit(window.train, epochs=50,
                      validation_data=window.val,
                      callbacks=[early_stopping])
  return history
```

Algoritmo 1: Método para compilar los modelos

Es importante explicar los distintos campos de este método. En primer lugar, el modelo recibe tres parámetros, aunque únicamente el modelo y la ventana de datos son obligatorios. El parámetro que no es obligatorio es la "paciencia". La paciencia se puede definir como el "número de épocas sin mejora después de las cuales se detendrá el entrenamiento" (K. Team, s.f.). Es decir, para el ejemplo del método mostrado, si no se produce una mejora en los resultados, se ejecuta el modelo un número de veces igual

al valor de la paciencia. Si en esas nuevas épocas no hay una mejora, el entrenamiento finaliza. Es importante llegar a un equilibrio en la paciencia, ya que dejar un valor muy alto para ésta puede resultar un gran aumento del tiempo de ejecución sin grandes mejoras o generando overfitting/sobreajuste. El overfitting se produce cuando el modelo recibe demasiadas iteraciones de los datos y tiende a aprender los resultados en vez de generalizar las relaciones entre variables. Aportar una paciencia muy baja puede también suponer que el modelo genere un subajuste/underfitting. En este caso, el modelo no ha tenido las iteraciones suficientes para generalizar las relaciones entre las variables. Se ha probado utilizar distintos valores de este campo a la hora de probar los modelos y se ha encontrado que no hay una diferencia muy significativa en una paciencia superior a 2.

Puesto que se ha explicado el concepto de paciencia, resulta conveniente explicar la parte del método en el que es utilizado: el "callback". Los callbacks "están diseñados para monitorizar el rendimiento con métricas en ciertos puntos del entrenamiento del modelo y realizar alguna acción que dependa de los valores obtenidos en las métricas de rendimiento" (Chen, 2020). En este caso, el modelo va a ser entrenado 50 veces/épocas. Este es un número bastante grande y probablemente excesivo. Para evitar que los datos sean sobreentrenados, el "callback" utilizado garantizará parar el entrenamiento del modelo una vez se llegue al número de paciencia sin que los resultados mejoren. Para ello, debido a que se busca el valor mínimo de error ("val_loss" en el código), el modo en el EarlyStopping será puesto a "min" (mínimo).

Por otro lado, dentro del modelo se encuentran los parámetros utilizados para la compilación del modelo: el optimizador, la pérdida y las métricas.

En primer lugar, se comentará el optimizador. "Los optimizadores se utilizan para mejorar la velocidad y el rendimiento para entrenar un modelo específico". (Equipo de Tutorialspoint, s.f.). Existe una gran variedad de optimizadores, entre los que se encuentra:

- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- SGD

Para decidir qué optimizador utilizar en este proyecto, se ha utilizado un modelo de ejemplo para ver cómo influye cada optimizador en el resultado. Los resultados del estudio se muestran a continuación:

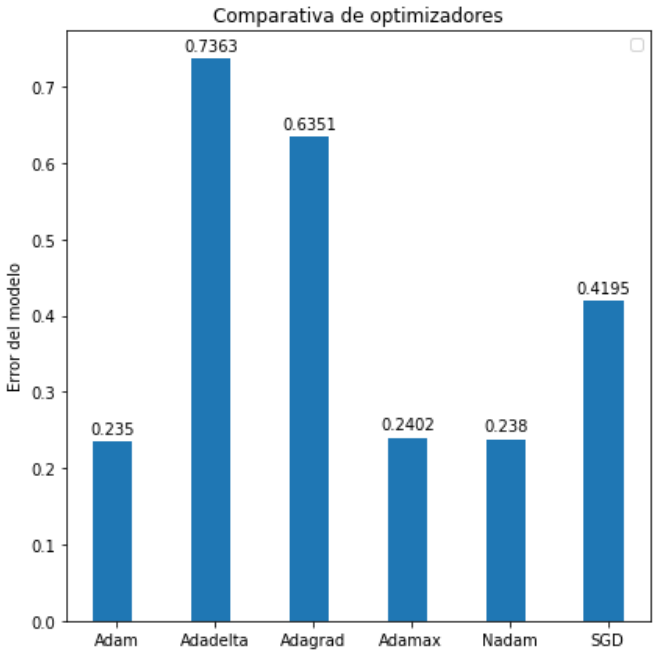

Ilustración 4.14: Comparativa de optimizadores

Como se puede observar, dependiendo del optimizador, un modelo puede variar sus resultados de manera substancial. Por ello, es importante realizar el estudio para saber qué optimizador utilizar. Pese a que Adam, Adamax y Nadam tienen valores muy cercanos, se ha decidido optar por Adam como el optimizador para los modelos.

Queda por analizar el “loss” y el “metrics”:

El “loss” o pérdida hace referencia a “un valor escalar que intentamos minimizar durante el entrenamiento del modelo. Cuanto menor sea la pérdida, más cercanas serán nuestras predicciones a los valores reales” (Ramadan, 2016). Existen varios tipos de métricas para utilizar como pérdida, como, por ejemplo:

- Mean Absolute Error / Error medio absoluto
- Mean Squared Error / Error medio cuadrático

Se ha decidido escoger el valor del error medio absoluto. Esto es debido a que el error medio cuadrático penaliza enormemente grandes desviaciones. Teniendo en cuenta que el tráfico en ocasiones es algo muy imprevisible, puede ocurrir que haya

grandes disparidades entre el valor real y el que predijo el modelo. Por ello, utilizar el error medio cuadrático penalizaría mucho al modelo en estas ocasiones. Por otro lado, el error medio absoluto es una métrica bastante fiable ya que devuelve el valor exacto del error, sin penalizar.

Finalmente, la métrica es “una función que se utiliza para juzgar el rendimiento de un modelo” (K. Team, s.f.). Debido a que se ha utilizado como pérdida el error medio absoluto, es conveniente mantener la misma medición para la métrica.

4.4.8 Creación de modelos

A la hora de crear un modelo de serie temporal, es importante entender los distintos tipos de modelos que hay. De esta forma, será más sencillo decidir qué modelos son óptimos para resolver el problema en cuestión.

El primer tipo de modelo, y el más simple, son los modelos de un solo paso. Estos modelos sacan predicciones para una característica a un solo paso de tiempo. Para ello, hacen uso de las condiciones actuales (t=0). Para poder entenderlo mejor, se ha creado un gráfico ilustrativo:

Ilustración 4.15: Modelos de un solo paso

Como se puede ver, el modelo recibe únicamente como valor de entrada los datos de las características del tiempo actual. Esto lo utiliza para generar un modelo que tendrá como resultado la predicción para el siguiente paso de tiempo. En el caso de este proyecto, dado que se utiliza un paso de tiempo de 1 hora, este tipo de modelo no es ideal, debido al desfase que tiene la API de tráfico en actualizar los datos, que puede rondar las 2 horas. Por tanto, si se utilizase este modelo, estaríamos prediciendo

datos a un tiempo pasado, por lo que no aportaría valor real. Por tanto, este tipo de modelo queda descartado.

Los modelos más interesantes son los modelos de varios pasos. Estos modelos toman un rango de valores pasados como entrada, y tras analizarlos, generan varias predicciones de valores futuros. Para esto existen dos posibles tipos: los modelos de un solo disparo y los modelos autorregresivos.

Durante el desarrollo de este trabajo se probó a implementar y comparar los modelos autorregresivos y los modelos de un solo disparo. Un modelo autorregresivo requiere una complejidad mucho mayor que un modelo de un disparo. Además, tras realizar las comparativas entre los modelos autorregresivos y los de un solo disparo, los resultados que se obtuvieron eran peores en todos los autorregresivos. Por tanto, se ha descartado realizar el análisis con los modelos autorregresivos y únicamente se van a analizar los tipos de modelos de un solo disparo.

Los modelos de un solo disparo son aquellos que utilizan uno o varios valores de tiempo pasados para generar múltiples salidas de golpe. Estas salidas dependerán en buena medida de los valores anteriores pasados. Se muestra a continuación un gráfico general de los modelos de un solo disparo:

Ilustración 4.16: Modelo de un solo disparo

En la gráfica expuesta, el modelo utiliza una ventana de tamaño 4 para realizar 3 predicciones a futuro.

4.4.9 Creación de modelos: modelo lineal de varios pasos

Dentro de los modelos de un solo disparo se encuentran varios tipos. En primer lugar, el modelo más simple es el modelo lineal. En esta ocasión, el modelo lineal recibe el último valor del input y con ello realiza múltiples predicciones de forma lineal. Pese a

que este modelo pueda dar buenos resultados, sus predicciones están muy limitadas ya que sólo tiene de entrada un único valor. Gráficamente sería así:

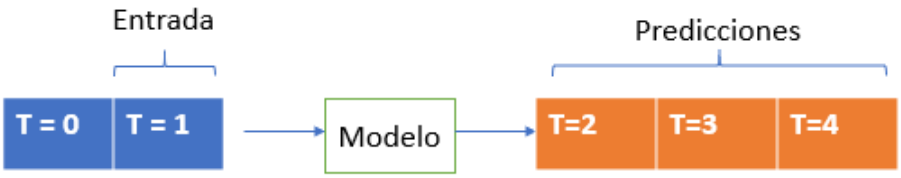

Ilustración 4.17: Modelo lineal de varios pasos

Este modelo va a ser implementado junto al resto de los expuestos en las próximas secciones. Una vez implementos todos y explicado su funcionamiento, se procederá a compararlos entre sí y seleccionar aquel que presente mejores resultados. Estos métodos están adaptados de la documentación oficial de Tensorflow (Tensorflow Team, 2021).

La implementación de este modelo se encuentra a continuación (Tensorflow Team, 2021):

```
linear_model = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    tf.keras.layers.Dense(64),
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

Algoritmo 2: Implementación del algoritmo lineal para series temporales de un solo disparo

Como el modelo lineal recibe como entrada únicamente el último valor de los datos, se necesita una capa lambda que recoja únicamente el último valor de esa ventana. Además, se le añade una capa densa que ayude al modelo a realizar las predicciones. Finalmente hay que crear una capa que devuelva los resultados de capas anteriores en las dimensiones apropiadas, que sería el número de elementos de salida que se desea y el número de características que hay.

Para compilar y ejecutar un modelo, se utiliza la función compile_model creada anteriormente. A esta función se le pasará por parámetros el nombre del modelo que se quiere utilizar y la ventana de datos a utilizar. Para la ejecución de este modelo, quedaría así:

```
compile_model(linear_model, multi_window)
```

De la ejecución del modelo se obtiene el error. Este error será almacenado para mostrarse en la comparativa entre modelos. Por otro lado, también se puede graficar el modelo. A continuación, se encuentra un ejemplo de 3 ventanas de tiempo con las predicciones y el valor real:

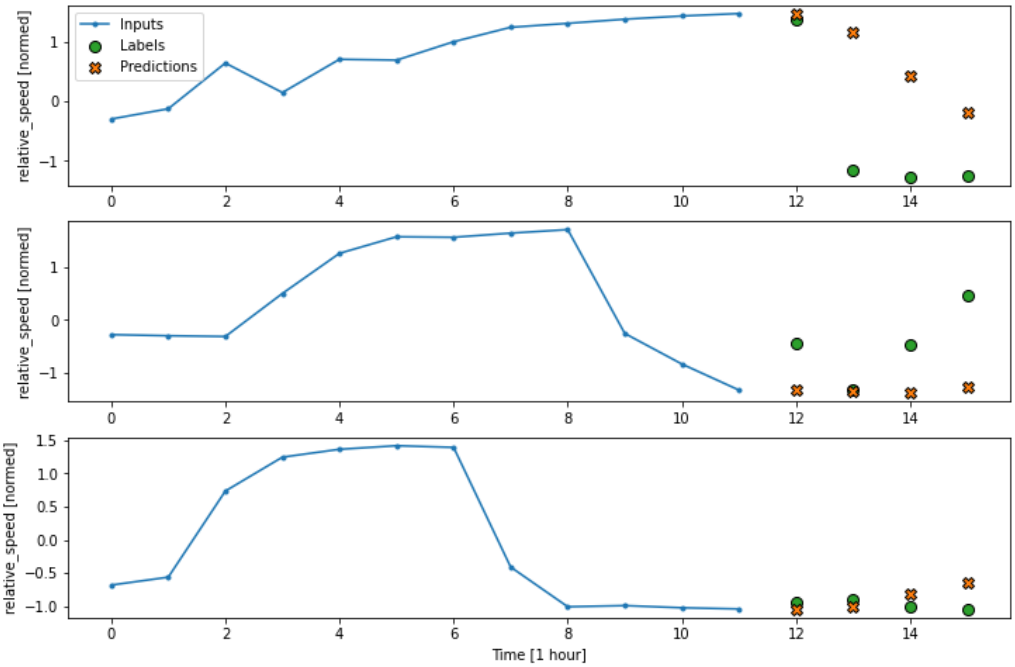

Ilustración 4.18: Ventana de datos para el modelo lineal

Hay que tener en cuenta que para el modelo lineal el único valor que se utiliza como entrada es el último valor del input. Esta gráfica sigue una tendencia que comparte el resto de los modelos: Si el estado del tráfico no fluctúa mucho, las predicciones suelen ser bastante acertadas. Esto se puede observar, por ejemplo, en la tercera ventana de datos. Por otro lado, para la primera y segunda ventana de datos se puede observar que las predicciones resultaron ser muy erróneas debido al gran cambio en el valor real del estado del tráfico.

También resulta interesante observar cómo progresa el modelo a medida que éste recorre los datos múltiples veces (épocas):

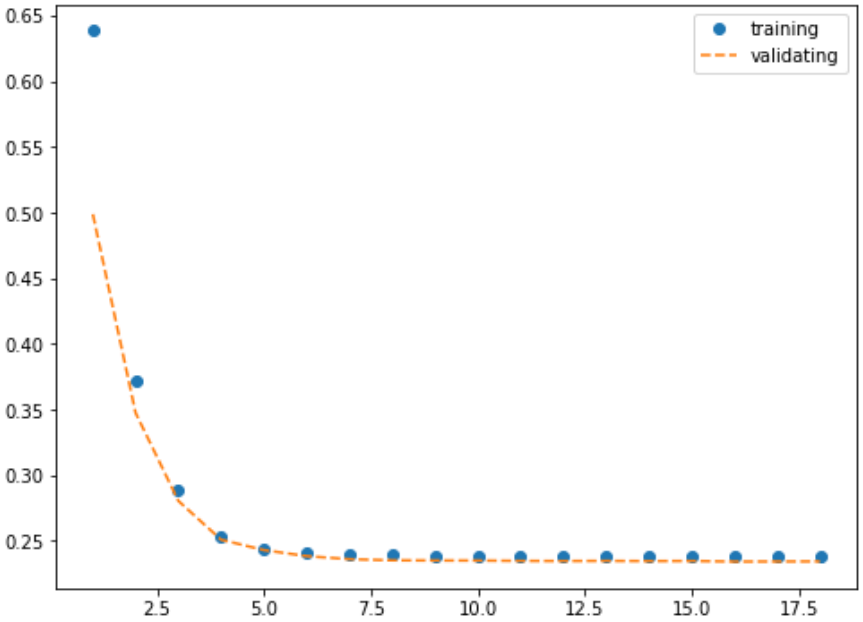

Ilustración 4.19: Evolución del error del modelo lineal por época

Como se puede comprobar, a partir de las 7-10 épocas, el error del modelo no disminuye significativamente. Pese a esto, continúa ejecutándose varias veces más. Esto es debido a la paciencia que se aplicó en el modelo, de 2 épocas.

4.4.10 Creación de modelos: modelo denso de varios pasos

Por otro lado, se encuentra el modelo denso. La base del modelo denso es la misma que la del modelo lineal, es decir, utiliza el último valor para realizar predicciones. Este modelo hace uso de una o varias capas densas para generar las predicciones. Una capa densa es una capa cuyos nodos están conectados con todos los nodos de entrada y todos los nodos de salida. Es un tipo de capa muy utilizada y útil a la hora de realizar modelos. Además, este tipo de capas recibe una función de activación.

> "Una función de activación se encarga de devolver una salida a partir de un valor de entrada, normalmente el conjunto de valores en un rango determinado como (0,1). Se buscan funciones que las derivadas sean simples, para minimizar el coste computacional." (Calvo, 2018)"

Existe una gran variedad de funciones de activación, pero las más conocidas son:

- Sigmoide: En una función sigmoide, todos los valores son transformados a la escala (0,1). Aquellos valores superiores tenderán a 1, mientras que los valores más bajos tenderán a 0.
- Tanh: Muy parecido a la función sigmoide, la tangente hiperbólica transforma los valores a una escala de -1 y 1. De la misma forma, aquellos valores superiores tenderán a 1 y los menores a -1.
- ReLU (Rectified Lineal Unit): Esta función de activación elimina todos los valores negativos (los pone a 0) y todos aquellos valores positivos mantienen su valor.

Para saber qué función de activación utilizar es conveniente utilizar todas y ver cual proporciona mejores resultados en el modelo. Para ello, se ha creado una tabla comparativa entre las 3 funciones de activación mencionadas anteriormente, para ver cuál es la idónea para el entrenamiento del modelo.

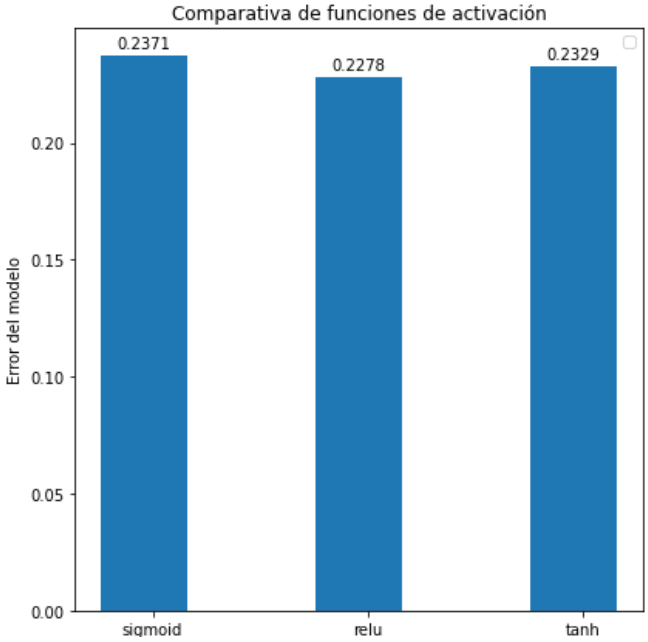

Ilustración 4.20: Comparativa de funciones de activación

Pese a que las diferencias entre las funciones de activación no son muy elevadas, resulta conveniente utilizar la que proporcione unos mejores resultados, ya que se busca la minimización del error en el modelo. Por tanto, se utiliza “relu” como función de activación. El código sobre el que se implementó este experimento fue la implementación del modelo denso de varios pasos, que será mostrado a continuación (Tensorflow Team, 2021):

```
dense_model = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),

    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

Algoritmo 3: implementación del algoritmo denso para series temporales de un solo disparo

Como se puede comprobar, el código para implementar un modelo denso es el mismo que el código para implementar un modelo lineal. Lo único que los diferencia es que en el modelo denso se añade una o más capas densas adicionales con el objetivo de entrenar mejor el modelo y tener resultados mejores.

Nuevamente se mostrará la gráfica con el error por épocas:

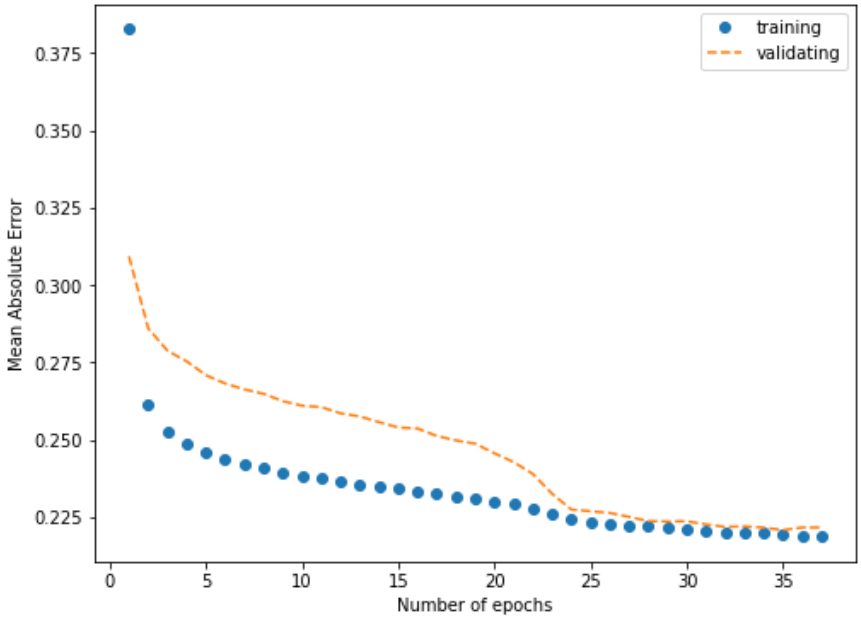

Ilustración 4.21: Evolución del error del modelo denso por época

Existen dos diferencias claras entre la gráfica del modelo lineal y el modelo denso. En el modelo lineal sólo se necesitaba un total de 18 épocas para el entreno del modelo, mientras que en el modelo denso este número aumenta hasta casi 40. Además, La línea que muestra el error para los valores de prueba muestra un mayor error también para las primeras épocas.

Otra gráfica interesante para ver los resultados es representar los pesos de las columnas gráficamente para ver qué columnas son las que están influyendo más en el modelo:

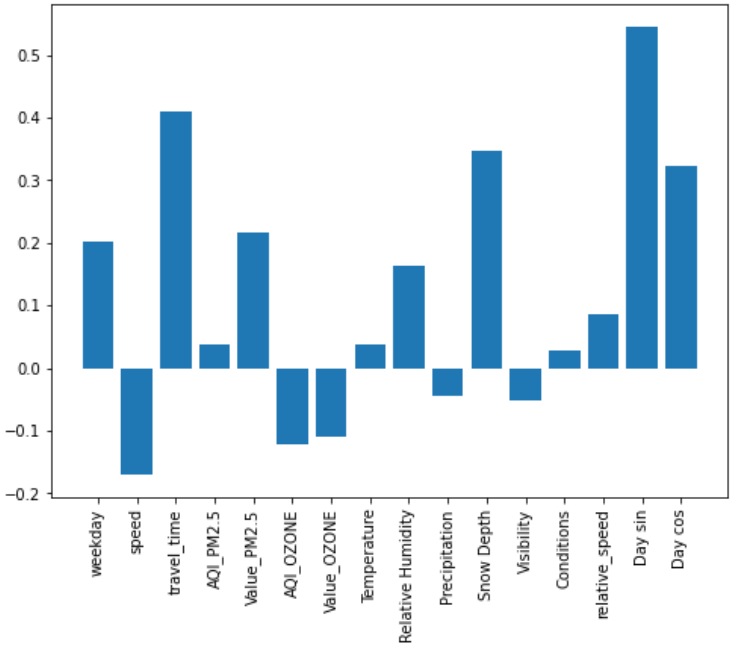

Ilustración 4.22: Pesos de cada columna en el modelo denso

Es importante remarcar que, pese a que en esta gráfica se muestren unos pesos determinados, en cada ejecución del modelo estos pesos puedan variar. Esto es debido a que la red neuronal va realizando pruebas distintas que pueden resultar en pesos distintos entre ejecuciones. Teniendo esto en cuenta, en la gráfica mostrada se puede visualizar como prácticamente las columnas tienen algún tipo de peso. Esto muestra que todas las características utilizadas para entrenar el modelo aportan en la predicción de resultados, tanto las características que tengan pesos positivos como aquellas que tengan pesos negativos son útiles.

4.4.11 Creación de modelos: Red Neuronal Convolucional

Una Red Neuronal Convolucional (CNN) aplicada a series temporales es un modelo que utiliza un historial para crear un modelo que será entrenado para generar varias predicciones a futuro. Gráficamente el modelo funciona de la siguiente forma:

Ilustración 4.23: Red Neuronal Convolucional (CNN)

En el ejemplo de la gráfica se muestra como hay varios valores pasados, de los que utiliza 3 como entrada para entrenar el modelo que generará 3 predicciones. Este modelo tiene la ventaja con respecto a los vistos anteriormente que utiliza valores pasados para ayudar a generar predicciones. De esta forma, si una tendencia es decreciente, tras utilizar 3 valores como entrada el modelo es más probable que realice una predicción acorde a la tendencia vista en los valores anteriores.

La implementación de la Red Neuronal Convolucional para series temporales es la siguiente (Tensorflow Team, 2021):

```
CONV_WIDTH = 4
CNN_model = tf.keras.Sequential([
    tf.keras.layers.Lambda(lambda x: x[:, -CONV_WIDTH:, :]),
    tf.keras.layers.Conv1D(256, activation='relu', kernel_size=(CONV_WIDTH)),
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

Algoritmo 4: implementación de la CNN para series temporales de un solo disparo

La implementación de la Red Neuronal Convolucional contiene elementos vistos en los anteriores modelos. En primer lugar, se utiliza una capa lambda para coger los valores que se desee utilizar como entrada para el modelo. Es importante jugar con este valor, ya que el error medio absoluto puede variar en función del número de valores de entrada que le proporcionemos. Si se le proporciona muy pocos valores de entrada, es posible que el modelo no vea la tendencia de los datos. Si se proporcionan muchos datos, es probable que el modelo sea sobreentrenado con los valores anteriores y las predicciones salgan peor. Para este proyecto se probó con diferentes valores, y “4” resultó ser el valor con el que se obtenía un menor error medio absoluto.

La gran diferencia con respecto a los otros modelos es la capa convolucional que se añade. Debido a que el resultado que se desea son una serie de valores, la dimensión de esta capa será de 1. Si se quisiese utilizar una CNN para analizar patrones en imágenes, las dimensiones de esta red serían mayores. En esta capa, “el modelo aprende a extraer características de secuencias y cómo mapear las características internas a diferentes tipos de actividades” (Brownlee, 1D Convolutional Neural Network Models for Human Activity Recognition, 2020). En esta capa se le indica el número de filtros que se desea utilizar, el tipo de activación y finalmente el tamaño del kernel. Este último parámetro hace referencia al tamaño de pasos de tiempo que se le aplica como

entrada. Normalmente haría falta añadir otro parámetro que sea el “input_shape”, pero en este caso no es necesario debido a que la capa anterior ya ha hecho el trabajo de preparar el tamaño de la entrada.

Además, se añade la misma capa densa utilizada en los otros modelos con la intención de añadir complejidad a la red para que pueda realizar predicciones más acertadas. Finalmente, se añade una nueva capa para que la salida de la red neuronal contenga el número de predicciones que queremos realizar.

Los resultados de este modelo pueden representarse nuevamente para ver el progreso del modelo por época:

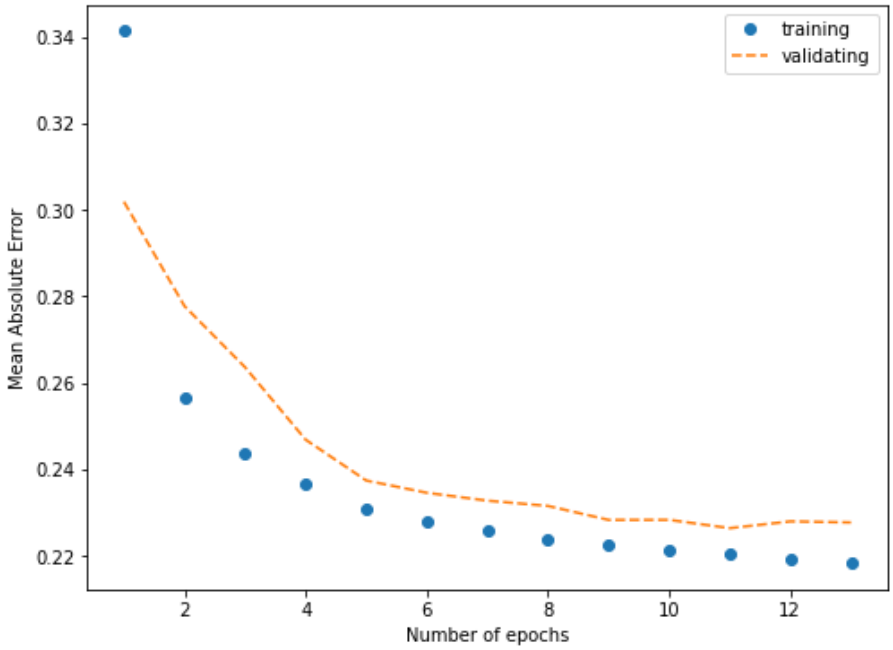

Ilustración 4.24: Evolución del error del modelo convolucional por época

Como se puede ver, el error disminuye progresivamente hasta aproximadamente la iteración 13, a partir de la cual se llega a la condición de “early stopping”.

4.4.12 Creación de modelos: Red Neuronal Recurrente

La Red Neuronal Recurrente (RNN) es el último de los modelos de un disparo que se va a utilizar para realizar la comparativa entre modelos. Al igual que la Red Neuronal Convolucional, este modelo hace uso de varios valores del pasado para realizar sus predicciones. La gran diferencia entre la Red Neuronal Recurrente y la Red Neuronal Convolucional es que en el entrenamiento de la RNN se va actualizando el modelo por

cada valor de entrada, y en la última iteración del modelo se generan las predicciones a futuro. Gráficamente el modelo funcionaría de la siguiente forma:

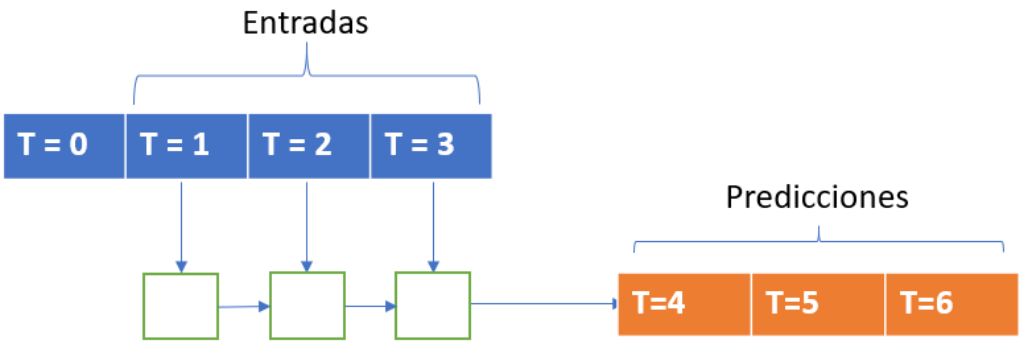

Ilustración 4.25: Red Neuronal Recurrente (RNN)

Como se puede observar, se van generando modelos para cada uno de los valores de entrada. Estos modelos a su vez son influenciados por los valores anteriores del modelo. Para implementar esto, se hace uso del LSTM o "Long Short-Term Memory". "En lugar de neuronas, las redes LSTM tienen bloques de memoria que están conectados a través de capas". (Brownlee, Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras, 2020).

A continuación, se muestra la implementación de la Red Neuronal Recurrente con LSTM:

```
RNN_LSTM_model = tf.keras.Sequential([
    tf.keras.layers.LSTM(128, return_sequences=False),
    tf.keras.layers.Dense(OUT_STEPS*num_features,
                          kernel_initializer=tf.initializers.zeros()),
    tf.keras.layers.Reshape([OUT_STEPS, num_features])
])
```

Algoritmo 5: implementación de la RNN para series temporales de un solo disparo

En este caso la primera capa es la específica de una RNN. Una capa LSTM recibe por parámetros el número de unidades de LSTM que se desea utilizar. Este valor debe ser modificado para buscar el mejor resultado posible. Algo importante en cuanto a la capa es el parámetro "return_requences=False". Esto es debido a que se busca sacar una única predicción. Recordemos que, en un modelo de disparo único, pese a que se realiza más de una predicción, todas ellas se realizan al mismo tiempo (en el último paso de tiempo).

Otra gran diferencia con respecto a los otros modelos es que no hace falta el uso de una capa adicional al principio para especificar el número de entradas que se desea utilizar. En este caso, el modelo recibe la ventana de datos completa y va generando su modelo haciendo uso de todos los datos. El resto de la red funciona de manera idéntica al resto. Se añade una capa densa para ayudar en la predicción de resultados y se cambia el tamaño de la salida para que sea con el número de predicciones que se desea realizar.

Tras ejecutar el modelo con la ventana de datos, se vuelve a visualizar la gráfica de entrenamiento con respecto a las épocas:

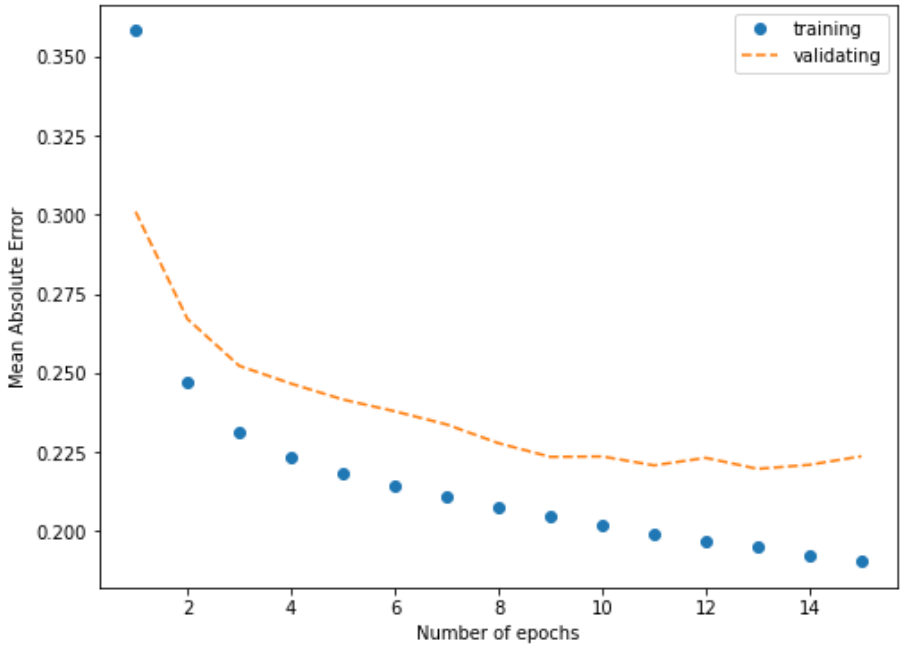

Ilustración 4.26: Evolución del error del modelo recurrente por época

4.4.13 Comparativa de modelos

Tras realizar todos los modelos de un disparo, se va a realizar una comparativa entre ellos. El objetivo con la comparativa es decidir qué modelo será el utilizado para calcular las predicciones en tiempo real. Para realizar esto, se hace uso del set de “test” de los datos (véase 4.4.4) y se evalúa el modelo con esos datos. A continuación, se va a representar en una gráfica el error medio absoluto en cada uno de los modelos para decidir cuál utilizar.

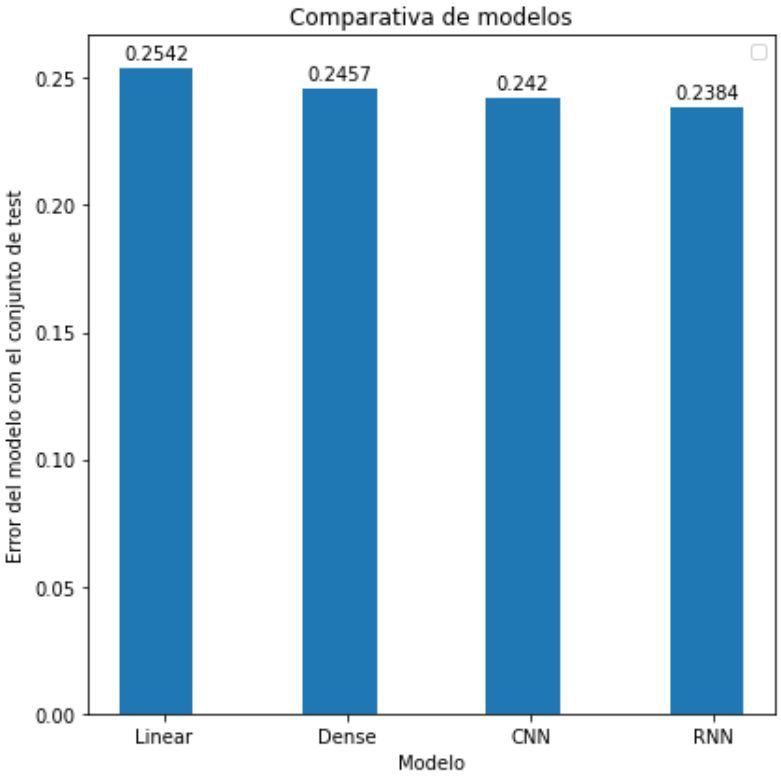

Ilustración 4.27: Comparativa de los modelos

La comparativa muestra que el mejor modelo para realizar predicciones es la Red Neuronal Recurrente. Pese a esto, todos los modelos tienen un error similar. Hay que remarcar también que cada vez que se realicen estas predicciones, los resultados pueden variar ligeramente. Esto es porque el modelo va aprendiendo automáticamente y sus predicciones pueden ser distintas de una ejecución a otra. Teniendo esto en cuenta, se va a utilizar igualmente el modelo RNN puesto que su análisis técnico también lo hace ideal para el trabajo. Es un modelo cuyo entrenamiento se ve influenciado por el valor del modelo para el valor de tiempo anterior. Esto hace que el modelo sea optimizado para mostrar los valores lo más cercanos a la realidad posibles.

A continuación, se va a estudiar los parámetros del modelo RNN que serán utilizados para realizar predicciones. Esto es, por ejemplo, el tamaño de la ventana de datos, el número de predicciones… Para ello, se ha realizado una tabla en la que se prueban distintas combinaciones. Esta tabla va a medir el error medio absoluto (MAE) al pasarle el conjunto de prueba al modelo. Se ha escogido el conjunto de prueba debido a que son una serie de datos que el modelo no ha visto anteriormente. Por tanto, esto se podría asemejar con los datos en tiempo real, que son datos nuevos que se le pasan al modelo. Además, se ha añadió una tabla con el tiempo de ejecución de cada uno de los modelos:

Número de predicciones	Tamaño de la ventana de datos	Error con el conjunto de prueba (MAE)	Tiempo de ejecución (segundos)
2	12	0.1868	42.81
2	24	0.1873	69.01
2	36	0.1825	115.19
3	12	0.2163	45.09
3	24	0.2154	75.97
3	36	0.2143	122.64
4	12	0.2380	52.35
4	24	0.2358	71.49
4	36	0.2333	102.81
5	12	0.2602	40.15
5	24	0.2590	79.56
5	36	0.2637	103.36
6	12	0.2760	46.04
6	24	0.2773	81.68
6	36	0.2773	65.65

Tabla 5: Comparativa del error para distintos tamaños de ventana de datos y número de predicciones en el modelo RNN

Se ha realizado una gráfica para complementar la tabla y que sea más sencillo realizar las comparaciones:

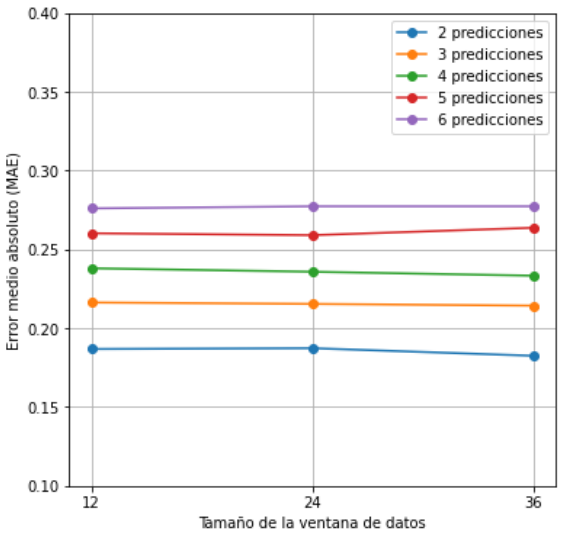

Ilustración 4.28: MAE de modelos con distintos números de predicciones y tamaño de ventana de datos

Como se puede apreciar, tanto en la gráfica como en la tabla, no existe una gran mejora en cuanto al error para ventanas de datos mayores. Esto ha hecho que se haya

decidido utilizar un tamaño de ventana de 12, ya que el tiempo de ejecución es menor y el error prácticamente no varía entre ventanas de datos mayores y menores.

Queda por decidir entonces el número de predicciones que se van a realizar. Si el número de predicciones es menor, estas predicciones tienen un menor error. El problema con utilizar pocas predicciones es la utilidad que puede representar. Como ya se argumentó en el apartado 4.6.1, un número muy pequeño de predicciones puede no tener ningún tipo de valor ya que los datos en tiempo real pueden tardar varias horas en ser actualizados en la API del tráfico. Por ello, se decidió realizar 4 predicciones. Este número de valores puede aportar valor dentro de un margen razonable de error.

4.4.14 Evaluación de las predicciones

El error de todos los modelos puede resultar algo elevado. Esto es por la naturaleza del problema que se intenta tratar. El estado del tráfico es un tema muy complejo de predecir ya que puede variar por muchos factores ajenos a los datos utilizados:

- Si ocurre algún tipo de accidente, el estado del tráfico puede pasar repentinamente de estar en buen estado a estar congestionado.
- Un vehículo no tiene por qué circular a velocidades cercanas al máximo en esa calle, aunque el estado del tráfico esté muy bien.
- Factores como un coche intentando aparcar en la calle o un camión realizando descargas que se encuentra estacionado en un carril son algunos ejemplos que pueden significar la reducción de la velocidad a la que circulan los coches en un instante dado, lo que puede generar fluctuaciones entre los valores reales y las predicciones.

Los ejemplos mencionados son algunos de los factores que muestran lo complejo que es realizar predicciones para el tráfico. Pese a ello, los modelos generan predicciones muy acertadas para valores de tráfico que no fluctúen mucho a lo largo del tiempo. Para los casos en los que haya un cambio drástico en la velocidad media a la que circulan los coches es donde se produce un mayor error en las predicciones (véase ilustración 4.18).

4.4.15 Generación de predicciones y guardado de modelo

Una vez escogido el modelo a utilizar y el número de predicciones a realizar, se tuvo que guardar el modelo. Este paso es muy importante ya que, de no hacerlo, se tendría que volver a entrenar el modelo para cada una de las 26 calles de las que se tienen datos de Manhattan. Esto haría que el proceso de generar predicciones fuese muy costoso en cuanto a recursos y tiempo del sistema. Es por ello por lo que se creó un directorio en el que se almacenó el modelo para cada una de las calles:

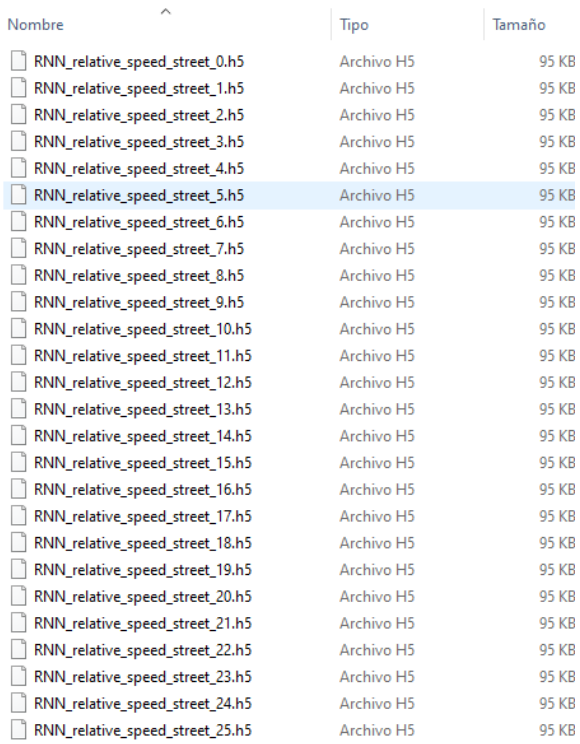

Ilustración 4.29: Modelos guardados

El nombre de los modelos muestra en primer lugar las siglas del modelo utilizado. Como se justificó anteriormente, el modelo a utilizar es el de RNN. Seguidamente se encuentra la variable de salida de los modelos, es decir, la variable sobre la cual se realizan predicciones. Finalmente se encuentra el número de la calle sobre la cual se ha creado el modelo. Pese a que el nombre de la calle aparece como número, se puede conocer el valor real de la calle (véase 4.3.2).

Gracias a tener los modelos guardados es posible cargarlos en cualquier momento para realizar predicciones. Para realizar nuevas predicciones es necesario normalizar

los datos nuevamente, puesto que los modelos guardados fueron entrenados con datos normalizados y la salida del modelo también está normalizada. Para ello, se utiliza el archivo guardado durante el entrenamiento de los modelos que contiene la media y la desviación estándar para cada uno de los modelos entrenados. Es decir, el proceso para realizar predicciones con datos nuevos sería el siguiente:

1. Cargar los nuevos datos y acotarlos según la calle sobre la que se quiera realizar la predicción.
2. Añadir las columnas que puedan faltar a los nuevos datos
3. Normalizar los datos utilizando la media y la desviación estándar de la calle sobre la que se realiza la predicción. La media y la desviación estándar se obtienen de un archivo que se generó al entrenar los modelos.
4. Crear la ventana de datos para pasarle al modelo. Para ello se hace uso de un método llamado windowPredictions(), que es una adaptación del método WindowGenerator() comentado en el apartado 4.4.5
5. Llamar al modelo con la nueva ventana de datos y obtener los resultados

El resultado obtenido del modelo es una predicción, pero su valor no es de gran utilidad, ya que el valor de la predicción que devuelve el modelo está normalizado. Hace falta desnormalizar los datos para obtener el resultado final de las predicciones. Como recordatorio, la forma de normalizar datos es la siguiente:

$$dato\ normalizado = \frac{dato - media\ de\ valores\ de\ esa\ columna\ (mean)}{desviación\ estándar\ de\ la\ columna\ (std)}$$

En este caso, como se cuenta con el dato normalizado y se busca desnormalizar el dato, la fórmula a aplicar quedaría de la siguiente forma:

$$dato = (dato\ normalizado\ \times desviación\ estándar\ de\ la\ columna) + media\ de\ esa\ columna$$

De esta forma se puede obtener el valor real de las predicciones del modelo.

4.4.16 Guardado de las predicciones

Antes de poder trabajar con las predicciones es necesario almacenarlas de alguna manera para que el resto de los módulos que hagan uso de ellas puedan acceder a los

datos en cuestión. En este proyecto, la forma de almacenar las predicciones es mediante un formato JSON.

Cuando los modelos generan predicciones o reciben datos en tiempo real, estos valores son almacenados dentro de un archivo CSV que tiene como objetivo almacenar un histórico con los datos reales de la velocidad relativa para cada hora y calle, además del valor de la predicción. De esta forma, resulta más sencillo realizar comparaciones entre las predicciones y los valores reales del modelo. Este archivo, además, es utilizado por el método de creación del JSON a la hora de generar el archivo con la información. El archivo JSON contiene los datos de las predicciones por hora para cada una de las calles y los valores reales por hora para cada una de las calles. La estructura del JSON es la siguiente:


```
{
   "Data":[
      {
         "street":"FDR N - TBB E 116TH STREET - MANHATTAN TRUSS",
         "2021-07-22 13:00:00":0.606,
         "2021-07-22 12:00:00":0.58,
         "2021-07-22 11:00:00":0.57,
         "2021-07-22 10:00:00":0.628,
         "2021-07-22 09:00:00":0.661,
         "2021-07-22 08:00:00":0.666,
         "2021-07-22 07:00:00":0.682,
         "2021-07-22 06:00:00":0.646,
         "2021-07-22 05:00:00":0.528,
         "2021-07-22 04:00:00":0.266,
         "2021-07-22 03:00:00":0.408,
         "2021-07-22 02:00:00":0.513,
         "2021-07-22 01:00:00":0.575,
         "2021-07-22 00:00:00":0.607,
         "2021-07-21 23:00:00":0.594,
         "2021-07-21 22:00:00":0.56
      }
   ],
   "RealValues":[
      {
         "street":"FDR N - TBB E 116TH STREET - MANHATTAN TRUSS",
         "2021-07-22 09:00:00":0.6542661813016071,
         "2021-07-22 08:00:00":0.6609412961184249,
         "2021-07-22 07:00:00":0.6796057888661992,
         "2021-07-22 06:00:00":0.7087957619343573,
         "2021-07-22 05:00:00":0.6573441060700228,
         "2021-07-22 04:00:00":0.6830410111951443,
         "2021-07-22 03:00:00":0.3594513007191689,
         "2021-07-22 02:00:00":0.286779721211045,
         "2021-07-22 01:00:00":0.3951151396769793,
         "2021-07-22 00:00:00":0.5543668057651898,
         "2021-07-21 23:00:00":0.5543816034804226,
         "2021-07-21 22:00:00":0.5837846636479328
      }
   ]
}
```


Ilustración 4.30: Estructura del JSON creado

La ilustración anterior muestra la estructura del JSON creado. Para esta ilustración sólo se muestra 1 calle, pero el archivo JSON contiene los datos de las 26 calles. El JSON tiene 2 secciones. Por un lado, se encuentran los valores de las predicciones

dentro del array de “Data”. Por otro lado, se pueden ver los valores reales de algunas de las horas anteriores (“RealValues”). Los valores vienen dados por la fecha y hora del registro. Como se puede comprobar, el apartado del JSON para las predicciones tiene predicciones para todas las horas que aparecen en el apartado de valores reales y 4 horas más, que hacen referencia a las 4 últimas predicciones que ha hecho el modelo. Estos datos servirán para representar gráficamente los resultados.

4.5 Creación de la página web

Una vez obtenidos los datos con los que trabajar y el modelo que utilizase esos datos para generar predicciones, se realizó una página web para representar estos resultados. La página web en cuestión es simple y tiene como objetivo mostrar de distintas formas las predicciones generadas. De esta forma, se podrá comprender en mayor medida la evolución del tráfico.

4.5.1 Requisitos previos

Para la creación de la página web se utilizó “React” para crear la estructura de la página web y las funcionalidades y CSS para darle estilo a la página web. Además de React se hizo uso de las siguientes librerías para ayudar con las funcionalidades de la página:

- React Bootstrap: Es la adaptación de la librería de Bootstrap para React. Fue utilizada en algunos estilos de la página.
- Mdbreact: Su nombre significa “Material Design for Bootstrap (React versión). Contiene distintos componentes que pueden ser utilizados para dar estilo a la página web. Algunos de los componentes fueron utilizados para ayudar a generar gráficas en la página web.
- React-chartjs-2: Es una librería que adapta los gráficos que pueden ser generados en chartjs para React. Se hizo uso de esta librería para generar gráficas en la página web.

Todas estas herramientas son necesarias para ejecutar la página web en local. Para poder instalar las herramientas se utiliza el comando “npm install”. Este comando instalará todas aquellas librerías necesarias.

4.5.2 Estructura de archivos

El proyecto cuenta con la siguiente estructura de archivos:

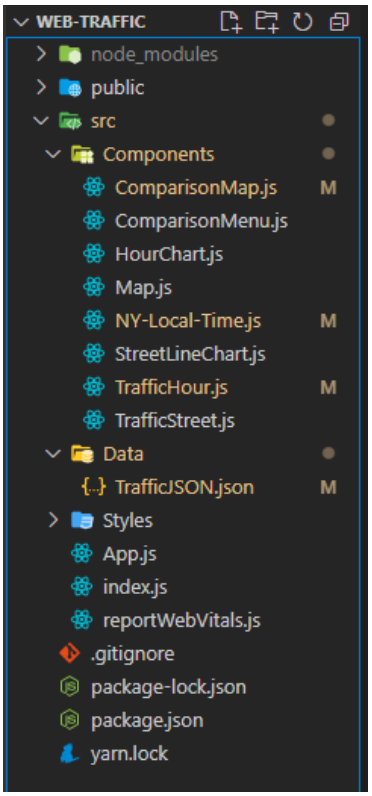

Ilustración 4.31: Estructura de archivos de la página web

Un proyecto en React consta de 3 carpetas principales: node_modules, public y src. De todas ellas, el código de la página web se realiza dentro de la carpeta /src. Dentro de este directorio se encuentran 3 carpetas que fueron creadas para organizar mejor el proyecto:

- Components: contiene una serie de componentes que son renderizados en pantalla. Estos componentes son la parte principal de la página web ya que aquí se encuentra toda la lógica y la creación de las gráficas que se muestra en la página.
- Data: Este repositorio únicamente cuenta con el archivo JSON con las predicciones y datos pasados sobre el estado del tráfico (véase ilustración 4.30). Los distintos componentes acceden a este archivo para generar las gráficas.
- Styles: Contiene los estilos para cada uno de los componentes renderizados.

Finalmente, fuera de todas las carpetas se encuentran 2 archivos que resulta conveniente explicar. El primero de ellos es “index.js”. Este archivo es creado por React

y llama al componente de App.js para renderizar los contenidos de la página. El archivo App.js tiene como objetivo llamar a los 5 componentes principales que aparecen en la página:

```
class App extends Component {
    render() {
        return (
            <div className="backgroundPic">
                <div className="container">
                    <LocalTime></LocalTime>
                    <TrafficHour></TrafficHour>
                    <TrafficStreet></TrafficStreet>
                    <ComparisonMenu></ComparisonMenu>
                    <Map></Map>

                </div>
            </div>
        );
    }
}

export default App;
```

Ilustración 4.32: Archivo App.js

Gracias a poder llamar a cada uno de los componentes por separado, si uno de estos componentes es actualizado sólo se volvería a renderizar el componente, no la página entera. A continuación, se va a explicar uno por uno cada uno de estos componentes:

4.5.3 Componentes de la página web

- Componente “LocalTime”: Este componente es informativo. Utiliza la hora actual del ordenador y la transforma a la zona horaria en la que se encuentra Nueva York. Además, busca la fecha y hora local de Nueva York dentro de los valores del JSON de datos. Si la fecha y hora no se encuentra en el JSON, se renderizará un mensaje diciendo “sin información”. Por otro lado, si la fecha y hora se encuentran dentro del JSON, calculará el número de valores que se encuentren por encima y por debajo de 0.5. Si el número de valores inferiores a 0.5 es superior al número de valores mayores que 0.5, significará que para esa hora el tráfico se encuentra en un mal estado, por lo que renderizará un mensaje para indicarlo. El componente se muestra de la siguiente manera:

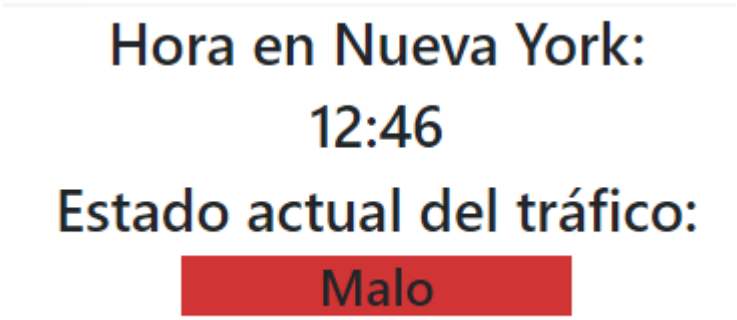

Ilustración 4.33: Componente "LocalTime"

- Componente “TrafficHour”: Se realizó este componente con el propósito de mostrar un menú todas las horas que apareciesen en el archivo JSON con los datos pasados y las predicciones. Para almacenar todos estos valores, se hace un mapeo del JSON y se va almacenando el valor de la hora array. Este array es después utilizado para mostrar todas las posibles horas. A continuación, se muestra visualmente este componente:

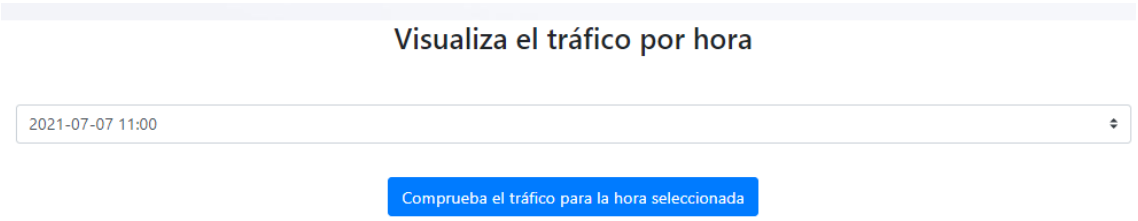

Ilustración 4.34: Componente "TrafficHour"

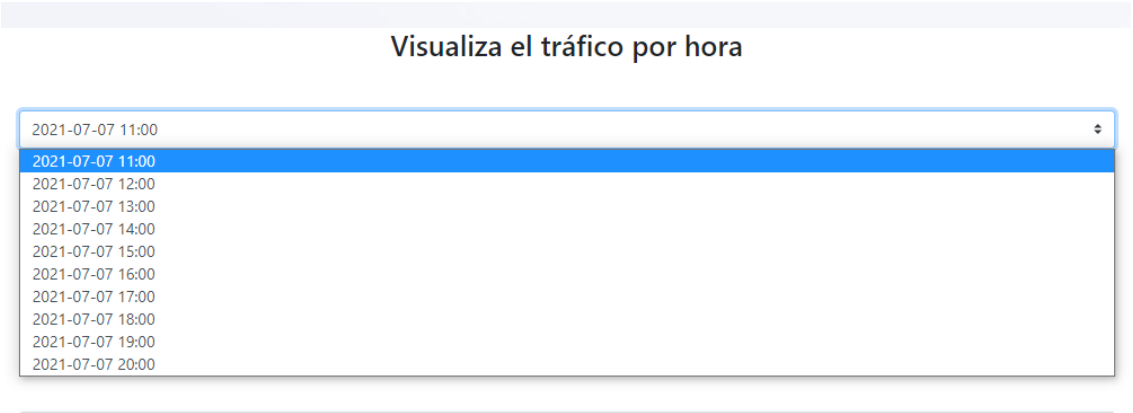

Ilustración 4.35: Menú de selección de fechas

Una vez seleccionada una hora, se puede hacer click en el botón para cargar el componente que generará una gráfica con los datos para esa hora. Si no se selecciona ninguna hora y se hace click en el botón, se mostrará por defecto la primera hora que aparezca en el menú. El componente que se renderiza no se encuentra en el archivo App.js, ya que su renderizado depende de que el usuario presione el botón. El componente en cuestión es el siguiente:

- Componente “HourChart.js”: Este componente recibe el valor del menú cuando el botón es presionado y genera un gráfico de barras con el valor del estado del tráfico (velocidad relativa) para cada una de las calles en la hora seleccionada. Para generar el gráfico se realiza un mapeo del archivo JSON y se almacena el valor de cada una de las calles para la hora seleccionada. El color de las barras de la gráfica depende del valor de la columna. Es decir, si el valor para la calle es inferior a 0.5, la barra aparecerá en color rojo. Si el valor es superior a 0.5, el color será verde. Esto sirve como indicación para determinar fácilmente si el estado del tráfico en una calle es bueno o malo. Se consideró que una velocidad relativa de 0.5 era un buen indicador para determinar si el estado del tráfico en una calle era bueno o malo.

Además, el componente renderiza una pequeña descripción para comprender el gráfico mostrado. El resultado es el siguiente:

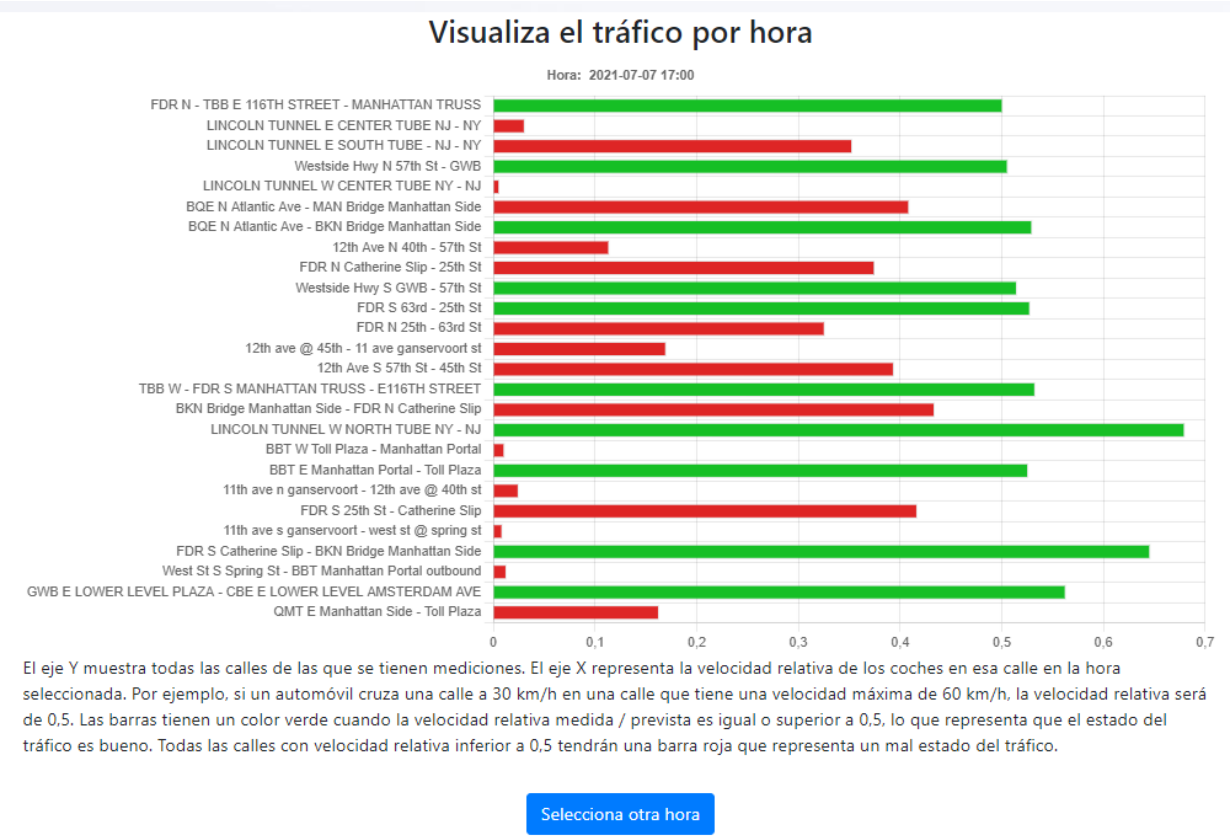

Ilustración 4.36: Gráfica del tráfico por hora

Al pasar el ratón por la barra aparecerá el valor de la velocidad relativa para esa calle en ese momento.

- Componente “TrafficStreet”: El funcionamiento de este componente es muy similar al componente de “TrafficHour”. En este caso, se muestra un menú con todas las calles disponibles:

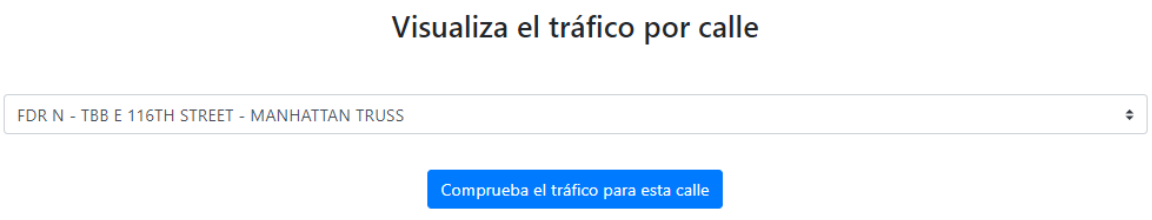

Ilustración 4.37: Componente “TrafficStreet”

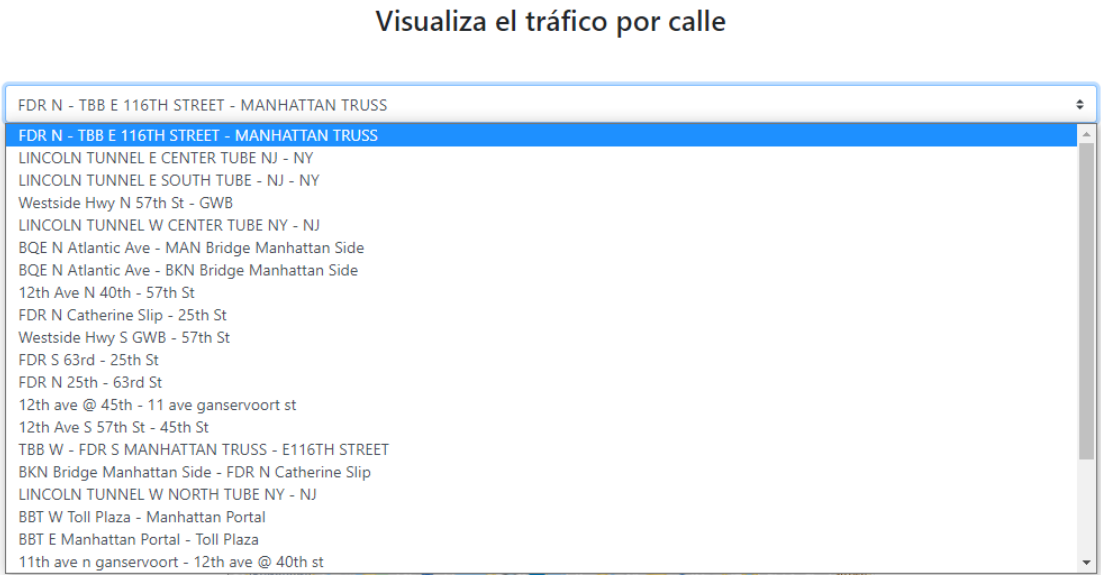

Ilustración 4.38: Menú con la selección de calles

Nuevamente, se podrá hacer click en el botón para comprobar el tráfico para la calle seleccionada. Si no se selecciona ninguna calle y se presiona el botón, aparecerá la gráfica para la primera opción que aparezca en el menú. En este caso, al presionar el botón se renderizará un componente nuevo llamado “StreetLineChart”:

- Componente “StreetLineChart”: Este componente recibe el valor seleccionado de la calle en el momento de presionar el botón y genera una gráfica de líneas para representar el estado del tráfico a lo largo de las horas. Los 6 primeros valores son los valores reales medidos mientras que los 4 valores siguientes son las predicciones que ha realizado el modelo. Para acceder a estos datos, se mapea nuevamente el contenido del JSON para obtener todos aquellos datos cuya calle sea la que fue seleccionada en el menú. Esta gráfica es generada utilizando react-chartjs-2 y mdbreact. Adicionalmente, se proporciona un párrafo informativo sobre la gráfica. El resultado es el siguiente:

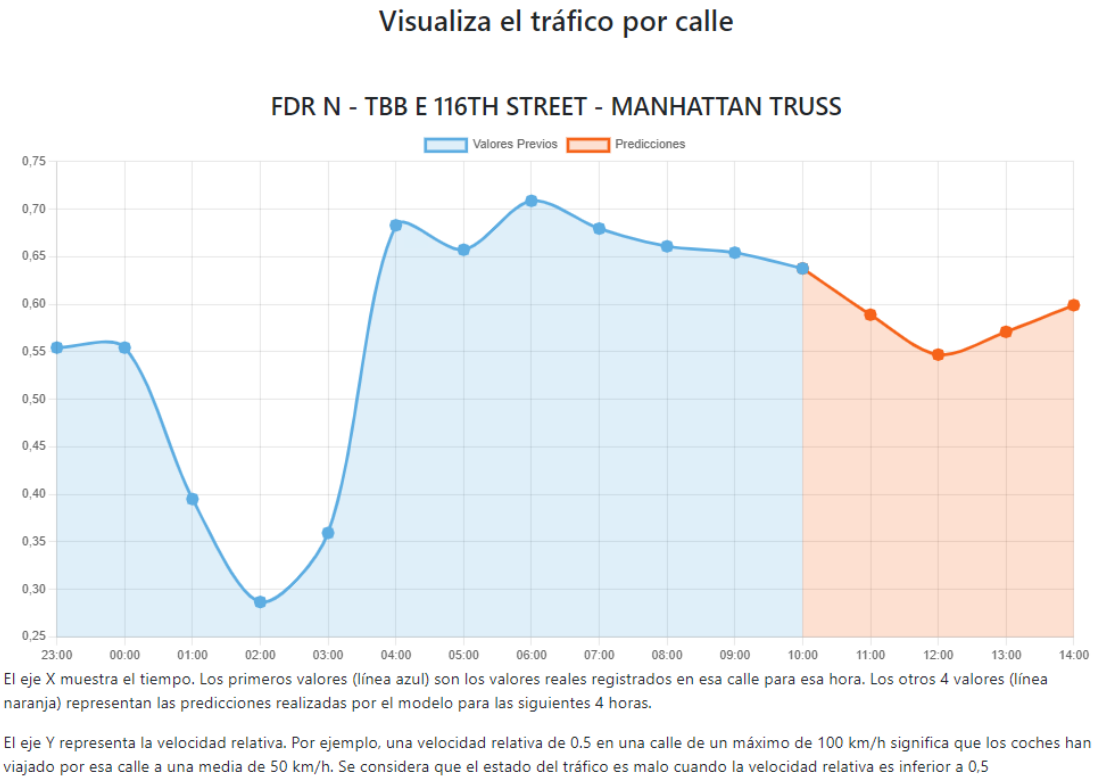

Ilustración 4.39: Gráfica del tráfico por calle

Al pasar el ratón por cada una de las horas se mostrará el valor de la velocidad relativa para esa hora en la calle seleccionada.

- Componente “ComparisonMenu”: Este componente es idéntico al componente “TrafficStreet”. Es un menú para seleccionar la calle que se desee. Al seleccionar una calle y presionar el botón, se generará un gráfico creado en el componente “ComparisonMap”.
- Componente “ComparisonMap”: Una vez seleccionada una calle y presionado el botón, se cargará este componente, que tiene 2 gráficos. El primero de ellos es un gráfico que muestra los valores reales del tráfico en una línea y los valores de las predicciones en otra. De esta forma, se pueden realizar comparaciones para ver cómo de acertadas estaban las predicciones. El segundo gráfico muestra el error en valor absoluto para cada una de las horas entre los valores reales y las predicciones. El componente se vería de la siguiente manera:

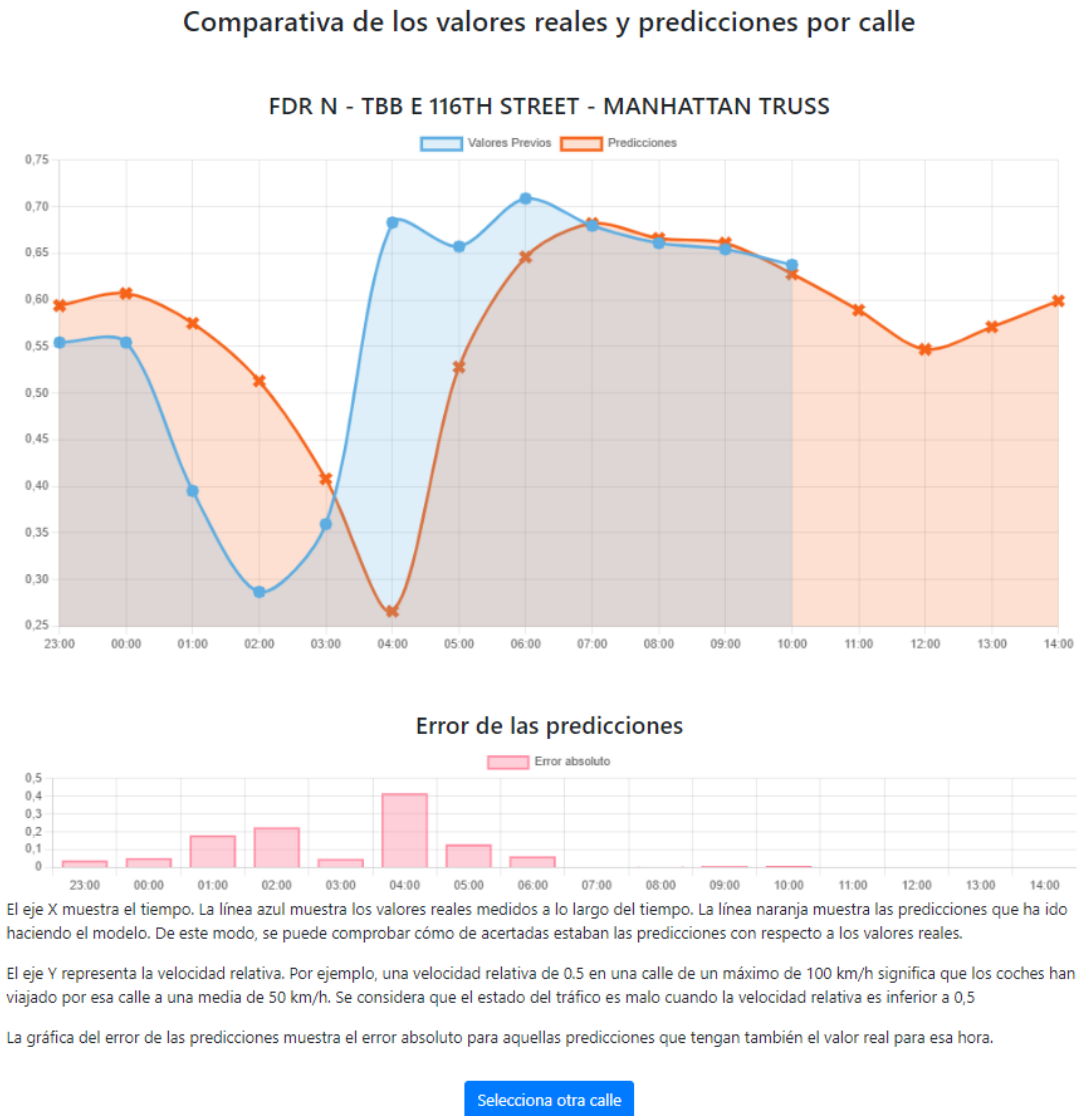

Ilustración 4.40: Gráfica comparativa de los valores reales y predicciones por calle y el error entre ambos

- Componente “Map”: Este componente es simplemente informativo y proporciona un mapa creado en Google My Maps con la localización de las calles de las que se realizan mediciones. Está adaptado, además, para cambiar su tamaño dependiendo del tamaño de la pantalla. El componente sería el siguiente:

Ilustración 4.41: Componente "Map"

4.5.4 Vista general de la página web

Una vez mostrados los principales componentes de la página web queda por visualizar la página web por completo:

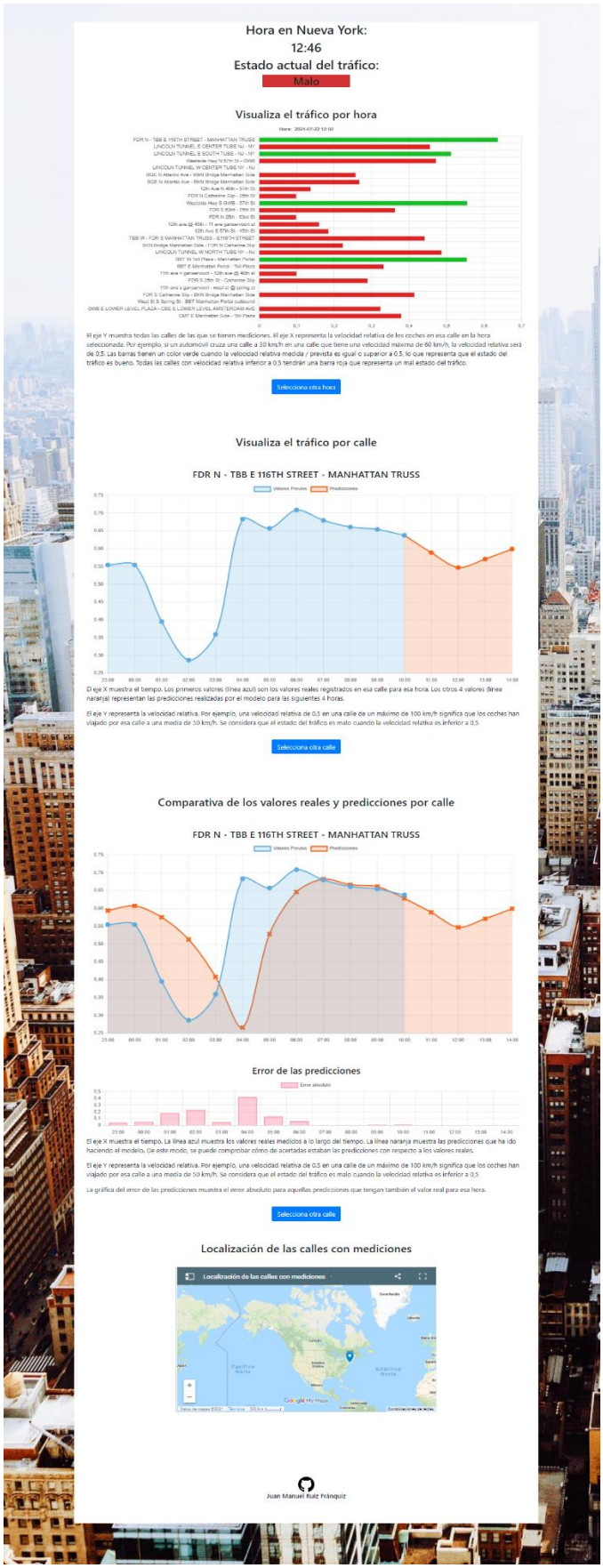

Ilustración 4.42: Vista general de la página web

Con esto concluiría la realización de la página web para representar los resultados del proyecto.

4.6 Integración de todas las fases

El desarrollo de las fases anteriores fue realizado individualmente, pero una vez hechas todas las fases de las que se compone el proyecto, para que éste sea funcional, se debe establecer una relación que las una a todas entre sí. Es por ello por lo que el proyecto se dividió en dos bloques. Por un lado, se encuentra todo el código relacionado con la recolección de datos históricos y en tiempo real y la generación de modelos para realizar predicciones. Por otro lado, se encuentra todo el código relacionado con la página web y la representación de las predicciones y de los datos. Para unir todas las partes entre sí, se realizaron las siguientes tareas:

- Unificar la ejecución de la recolección de datos en tiempo real con la generación de predicciones para los siguientes datos.
- Actualizar las nuevas predicciones en la página web.
- Crear un método principal que controle el flujo de toda la lógica para las predicciones y los datos en tiempo real, la creación de los modelos y la descarga de datos históricos.
- Reentrenar el modelo en tiempo real

4.6.1 Unión de los datos en tiempo real con la generación de predicciones

La primera de las tareas involucraba dos de los bloques realizados por separado durante el desarrollo del proyecto: la recolección de datos en tiempo real y la generación de predicciones utilizando el modelo escogido (véase apartado 4.4.13). Para unificar estos dos bloques se realizó una serie de cambios en los métodos encargados de recolectar datos en tiempo real.

La lógica que se implementó fue añadir una condición en el momento en el que se recibiese un nuevo valor para el clima o para la calidad del aire. Siguiendo la lógica del funcionamiento de la recolección de datos en tiempo real, si se recibe un nuevo valor para el clima o la calidad del aire significa que ya se recibieron todos los datos del tráfico para la hora anterior. Por tanto, este bloque de datos quedaría finalizado para esa hora.

El siguiente paso sería comprobar si también existe un valor para el otro atributo. Es decir, si se recibió un nuevo valor de clima, se comprueba que haya un valor también para esa hora de calidad del aire y viceversa. Si esta condición se cumple, significa que ya se han registrado todos los valores posibles de esa hora y, por tanto, se pueden extraer todos los datos para la hora. Las predicciones se harán siempre a partir de esta hora, ya que sería la última hora del conjunto de datos de la se cuenten todos los valores de todas las fuentes de datos. Estos valores son modificados para cumplir con los campos requeridos para realizar las predicciones. Este procesamiento incluye eliminar aquellas columnas que no sean utilizadas, añadir las nuevas columnas como el día de la semana o la velocidad relativa y transformar las columnas categóricas a sus valores numéricos y agrupar los valores por calle y hora.

Tras realizar todo el procesamiento de los datos, éstos están listos para ser aplicados a los modelos, por lo que se puede ejecutar el método que realice las predicciones. Este método se encarga de normalizar todos los datos, dividir los datos por calle y realizar las predicciones para cada una de las calles. Finalmente, el método desnormaliza los datos correspondientes a las predicciones que realiza (velocidad relativa) y genera el archivo JSON para que estas predicciones puedan ser interpretadas y utilizadas en la página web.

Aunque pudiese parecer que el proceso está listo, hay un matiz que tuvo que tomarse en consideración. Supongamos que se ejecuta por primera vez el programa para generar datos en tiempo real. Puesto que la ventana de datos utilizada para entrenar el modelo y realizar predicciones era de 12 unidades, es necesario esperar un total de 12 horas para empezar a ver las nuevas predicciones. Esto es muy problemático, ya que en este caso la página web tardaría 12 horas en mostrar los resultados. Esta es la cantidad de valores que necesitaría el modelo para empezar a hacer predicciones.

Para atajar este problema, se añadió al código una condición adicional. Al momento de ejecutar el programa, éste se asegura de tener las horas que necesita el modelo para empezar a hacer predicciones en el momento en el que se ejecuta por primera vez. Para ello, realiza una pequeña descarga de datos históricos. Este histórico de datos contendrá los valores de las últimas horas, por lo que se podrán generar predicciones. Para el tamaño del histórico de datos a descargar hay que tener en cuenta nuevamente la frecuencia a la que se actualizan los datos de las distintas API (véase 4.2.2). Como se mencionó anteriormente, la API de los datos de tráfico actualiza sus valores en intervalos irregulares de tiempo. Puede haber un desfase de varias horas a la hora de

actualizar los datos en la API. Este factor debe ser tomado en cuenta, ya que puede darse el caso en el que los datos no se hayan actualizado y, por tanto, la descarga de ese pequeño histórico de datos no sirva para generar predicciones. Para explicar este concepto, se va a realizar un ejemplo:

Supongamos que el programa se ejecuta por primera vez a las 15:00. Ya que el modelo necesita 12 valores para realizar predicciones, el programa procede a descargar los datos de las últimas 12 horas, es decir, desde las 03:00 hasta las 15:00. A continuación, supongamos que la API de tráfico no ha actualizado sus datos desde las 12:00, es decir, lleva un desfase de 3 horas. Los datos descargados de la API serían desde las 03:00 hasta las 12:00, que es el último registro de datos de tráfico. Esto es un total de 9 horas. El modelo necesita 12 valores, por lo que no podría generar predicciones hasta que la API de tráfico actualizase sus valores para 3 horas más.

Para lidiar con este problema, se aumentó el tamaño de la descarga de datos históricos. En este caso, para asegurar que se puedan realizar predicciones con desfases muy elevados, se añadió un margen de 4 horas adicionales. De esta manera, aunque la última actualización de los datos de tráfico fuese hace 4 horas, el modelo generaría predicciones para la hora actual (puesto que el modelo realiza predicciones a 4 horas).

4.6.2 Actualización de las predicciones en la página web

Este apartado de integración resultó ser más sencillo de implementar. El problema reside en que los datos que utiliza la página web para generar sus gráficas se encuentran dentro del propio directorio en el que está el código de la página web, mientras que el archivo JSON generado por las predicciones se creaba y almacenaba en un módulo distinto del proyecto. Para integrar ambos módulos se cambió el directorio en el que se guardaba el archivo JSON para estar en el mismo directorio utilizado por la página web para generar las gráficas.

Cada vez que se generen nuevas predicciones, el JSON generado reemplaza al archivo JSON que se encontrase anteriormente en el directorio. Gracias a que la página web genera las gráficas y todo su contenido de forma dinámica, en el momento en el que se produzca un cambio en el JSON, toda la información de la página web (horas y valores de predicción, gráficas…) será actualizada para mostrar los nuevos valores.

4.6.3 Método principal para el control de flujo de la lógica del proyecto

El siguiente punto para la integración del proyecto fue crear un archivo único que controlase la lógica del módulo del proyecto encargado de descargar los datos históricos, traer datos en tiempo real, crear los modelos y realizar las predicciones.

Este método es necesario por los siguientes dos motivos:

- Debido a que el proyecto cuenta con una gran cantidad de datos como son los datos históricos y los datos en tiempo real, el proyecto puede llegar a ocupar más de 2.5gb. Esto hace que el proyecto pueda resultar en tiempos de descarga elevados. También esto puede aumentar el tiempo a la hora de exportar el proyecto de un sitio a otro.
- El proyecto cuenta con muchas fases que pueden resultar confusas a la hora de trabajar por primera vez con él. Por ejemplo, si se cuenta con un proyecto sin datos, el usuario tendría que descargar manualmente los datos históricos, procesarlos y adaptarlos para crear el modelo y una vez hecho, ejecutar el programa para recoger los datos en tiempo real.

Resulta conveniente crear un archivo que controle todo el flujo de ejecución. De esta forma, alguien que vaya a trabajar con el proyecto puede empezar a hacerlo simplemente ejecutando un archivo llamado “main.py”

El archivo en cuestión debe tener una lógica en cascada que vaya comprobando si se tienen las distintas fases del proyecto. Por ejemplo, se puede dar el caso en el que no se tengan los archivos con el estado de los modelos predictivos. Si esto ocurre, debe comprobarse si el proyecto contiene la unión del histórico de datos para poder generar los modelos. En el caso que no sea así, comprobará que los datos hayan sido descargados (sin unirlos), y así sucesivamente. Para poder comprender en mayor medida la lógica de este archivo, se ha creado un diagrama de flujo:

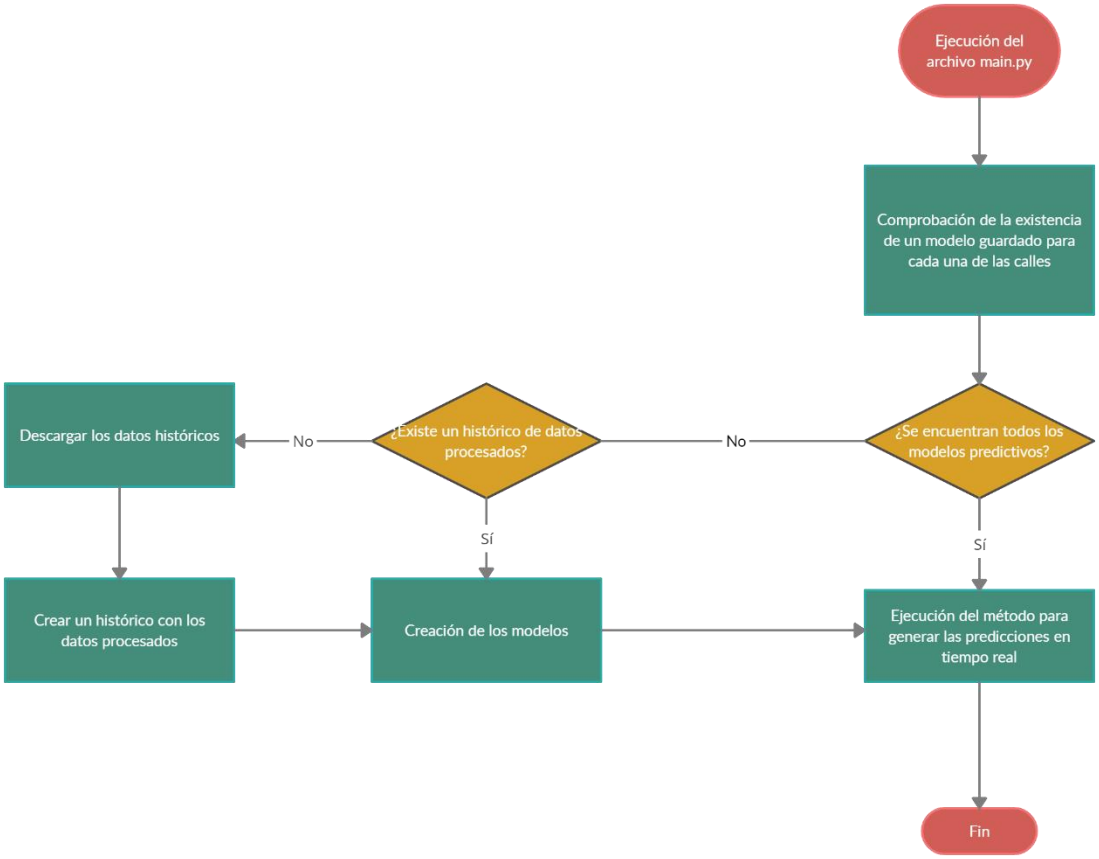

Ilustración 4.434: Lógica del archivo main.py

Gracias a la implementación de esta lógica es posible exportar el proyecto sin necesidad de contar con todos los datos históricos, los datos procesados o los modelos. Esto reduce exponencialmente el tamaño del proyecto. Además, esta lógica aporta un menor grado de complejidad al proyecto a la hora de trabajar con él por primera vez, ya que el código está adaptado para satisfacer las distintas situaciones posibles en las que se pueda encontrar el usuario al ejecutar el proyecto.

4.6.4 Reentreno de los modelos en tiempo real

Aunque los modelos han sido entrenados con un histórico de datos, resulta conveniente reentrenar los modelos en tiempo real con los nuevos datos que se obtengan, para asegurar que el modelo siempre vaya mejorando sus predicciones. Para ello se añadió que, una vez lleguen nuevos valores en tiempo real, antes de realizar las nuevas predicciones, se cargue el modelo y se reentrene con los últimos datos obtenidos de las API en tiempo real.

4.6.5 Visión general del proyecto

Una vez realizada la integración de las distintas fases, se puede considerar que el proyecto está preparado para ser utilizado. Por tanto, el proyecto estaría compuesto por dos módulos que deberán ser ejecutados por separado.

- Módulo 1 – Predicciones en tiempo real: Este módulo contiene toda la lógica para descargar los datos históricos, procesarlos y realizar los modelos predictivos y generar las predicciones en tiempo real mediante el uso de hilos. Para ejecutar este módulo deberá ejecutarse el archivo main.py
- Módulo 2 – Página web: Este módulo utiliza las predicciones generadas en tiempo real del módulo 1 para actualizar las gráficas y la información de la página web de forma dinámica. Para visualizar la página deberá inicializarse un servidor en local con la ejecución de la página web (comando a utilizar: npm start).

4.7 Ajuste a la planificación inicialmente prevista y modificación de los objetivos planteados

En el apartado 2.2 de este documento se especificaron aquellos objetivos planteados inicialmente para el desarrollo de este objetivo. Para poder realizar un ajuste sobre la planificación inicial, se irá mencionando cada uno de los objetivos planteados y la forma en la que fueron realizados.

- Realizar un estudio de las diferentes fuentes de información disponibles. La realización de este objetivo fue fundamental para poder proceder con el trabajo y fue el primer objetivo en ser realizado. Para ello, se hizo una búsqueda de todas las fuentes de información, bases de datos o API disponibles que pudiesen traer valor o ser de utilidad para las siguientes fases del proyecto. Una vez buscadas todas las fuentes de información, se realizó una comparativa entre ellas para descartar aquellas que por una razón u otra no fuesen de gran utilidad.
- Identificar las diferentes tecnologías de big data e inteligencia artificial que sean convenientes para el estudio a realizar: En la actualidad existe una gran variedad de tecnologías que ayudan a realizar las distintas fases del proyecto. Para la decisión sobre qué tecnologías utilizar se tuvo en cuenta las ventajas que éstas podrían traer a la hora de realizar las tareas del proyecto, la documentación disponible y comunidad detrás de cada tecnología, entre otros factores. Finalmente, se obtuvo un listado de herramientas que fueron las utilizadas para el desarrollo del proyecto.

- Efectuar la ingestión y fusión de datos dentro de una arquitectura big data. La ingestión y el procesamiento de datos se realizó en la primera fase del desarrollo del proyecto. La arquitectura creada está preparada para adquirir y procesar grandes cantidades de datos, tanto históricos como en tiempo real.

- Desarrollar el modelo predictivo del estado del tráfico en la ciudad de Nueva York: Para desarrollar este objetivo se partió de una serie de algoritmos que a priori podían ser de utilidad para resolver el problema en cuestión. Una vez implementados, los algoritmos fueron descartados según su utilidad o la calidad de los resultados obtenidos.

- Crear una página web para mostrar los resultados del análisis. La página web fue realizada para poder mostrar en tiempo real el estado del tráfico y las predicciones para las siguientes horas mediante el uso de gráficas generadas de forma dinámica. La página web creada actualiza su información a medida que se generen nuevas predicciones.

- Evaluar los resultados obtenidos: Evaluar los resultados fue un proceso fundamental a la hora de seleccionar el mejor modelo para la predicción del estado del tráfico. Los resultados obtenidos fueron evaluados para determinar qué modelos proporcionaban una mayor fiabilidad a la hora de generar predicciones.

Tras repasar los objetivos planteados inicialmente, se puede concluir que el proyecto realizado siguió la planificación inicial. No se realizó ninguna modificación con respecto a los objetivos inicialmente planteados, ya que todos eran necesarios para tener un proyecto funcional.

Capítulo 5
Conclusiones y trabajo futuro

El trabajo realizado ha consistido en la creación de dos módulos que estuviesen conectados entre sí.

El primero de ellos es el módulo encargado de realizar la descarga y el procesamiento de un histórico de datos de distintas fuentes de información. Estas fuentes de información son el tráfico, la calidad del aire y la calidad del aire de Nueva York y tenían como objetivo generar una serie de modelos predictivos del para cada una de las calles de las que se realizan mediciones en Manhattan. Los modelos creados generarían predicciones sobre una variable creada llamada “velocidad relativa”, que es la proporción entre la velocidad a la que circula un coche sobre la velocidad máxima de la vía en la que circula. Estos modelos creados fueron utilizados para generar predicciones en tiempo real. Los datos con los que estos modelos realizaban las predicciones en tiempo real eran proporcionados por una parte del proyecto encargada de recoger y procesar todos los datos que se fuesen actualizando en las distintas API en tiempo real. El flujo de ejecución de este módulo está controlado por un archivo principal encargado de comprobar si se cuenta con todo lo necesario para poder realizar predicciones en tiempo real.

El segundo de los módulos tenía como objetivo representar toda la información y las predicciones generadas por el primer módulo en una página web. Para ayudar a mostrar los datos, la página web genera de forma dinámica gráficas con los datos obtenidos por el primer módulo. Esta información es actualizada a medida que el primer módulo actualiza sus datos y predicciones en tiempo real.

El proyecto ha conseguido todos los objetivos inicialmente propuestos y los resultados de las predicciones son buenos a pesar de la naturaleza imprevisible del tráfico. Por lo general, estas predicciones y los datos obtenidos indican el gran problema que tiene la ciudad de Nueva York en cuanto al tráfico.

Algunas de las posibles extensiones de este trabajo son las siguientes:

- Utilizar un servidor dedicado para ejecutar en todo momento la recolección de datos en tiempo real.
- Crear una API y alojarla en algún servidor para que las predicciones generadas por los modelos sean accesibles por cualquier persona.
- Hostear la página web creada y utilizar el servidor dedicado a la recolección de datos en tiempo real y la API con las predicciones para actualizar las gráficas y la información mostrada en la página web en tiempo real.

Capítulo 6
Bibliografía

Alameda, T. (09 de 09 de 2020). *'Machine learning': ¿qué es y cómo funciona?* Obtenido de BBVA NOTICIAS: https://www.scribbr.es/detector-de-plagio/generador-apa/new/webpage/

Brasseur, G. P. (01 de 2003). *The impact of road traffic on global tropospheric ozone*. Obtenido de researchgate: https://www.researchgate.net/publication/251425922_The_impact_of_road_traffic_on_global_tropospheric_ozone

Brownlee, J. (28 de 08 de 2020). *1D Convolutional Neural Network Models for Human Activity Recognition*. Obtenido de Machine Learning Mastery: https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/

Brownlee, J. (28 de 08 de 2020). *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras*. Obtenido de Machine Learning Mastery: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Calvo, D. (07 de 12 de 2018). *Función de activación – Redes neuronales*. Obtenido de Diego Calvo: https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

Chen, B. (10 de 10 de 2020). *A Practical Introduction to Keras Callbacks in TensorFlow 2*. Obtenido de Medium: https://towardsdatascience.com/a-practical-introduction-to-keras-callbacks-in-tensorflow-2-705d0c584966

colaboradores de Wikipedia. (03 de 07 de 2021). *Python*. Obtenido de Wikipedia, la enciclopedia libre: https://es.wikipedia.org/wiki/Python

colaboradores de Wikipedia. (12 de 06 de 2021). *React*. Obtenido de Wikipedia, la enciclopedia libre: https://es.wikipedia.org/wiki/React

colaboradores de Wikipedia. (03 de 05 de 2021). *Valores separados por comas*. Obtenido de Wikipedia, la enciclopedia libre: https://es.wikipedia.org/wiki/Valores_separados_por_comas

Conecta Software. (10 de 03 de 2020). *Series temporales*. Obtenido de Conecta Software: https://conectasoftware.com/analytics/series-temporales/

Equipo de Tutorialspoint. (s.f.). *TensorFlow - Optimizers - Tutorialspoint*. Obtenido de Tutorialspoint: https://www.tutorialspoint.com/tensorflow/tensorflow_optimizers.htm

Escobar, G. (12 de 07 de 2017). *¿Qué es concurrencia?* Obtenido de El Blog de Make it Real: https://blog.makeitreal.camp/concurrencia/

Escuela de Ingeniería Informática ULPGC. (s.f.). *Competencias GII*. Obtenido de EII ULPGC: https://www2.ulpgc.es/archivos/plan_estudios/4008_40/ObjetivosyCompetenciasdelGII.pdf

Identificar valores atípicos - Minitab. (s.f.). Obtenido de (C) Minitab, LLC. All rights Reserved. 2019: https://support.minitab.com/es-mx/minitab/18/help-and-how-to/statistics/basic-statistics/supporting-topics/data-concepts/identifying-outliers/

J.Cuba, E. A. (08 de 06 de 2021). *Harvard Study: Car Pollution in NYC Claims 1,400 Lives, Billions in Costs*. Obtenido de Streetsblog New York City: https://nyc.streetsblog.org/2021/06/08/harvard-study-car-pollution-in-nyc-claims-1400-lives-billions-in-costs/

Juan, C. (05 de 02 de 2019). *¿Cuáles son las 5 V's del Big Data?* Obtenido de Thinking for Innovation: https://www.iebschool.com/blog/5-vs-del-big-data/

Julián, G. (30 de 12 de 2014). *Las redes neuronales: qué son y por qué están volviendo*. Obtenido de Xataka: https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo

K. Team. (s.f.). *Keras documentation: EarlyStopping*. Obtenido de Keras: https://keras.io/api/callbacks/early_stopping/

K. Team. (s.f.). *Keras documentation: Metrics*. Obtenido de Keras: https://keras.io/api/metrics/

K. Team. (s.f.). *Keras documentation: Model training APIs*. Obtenido de Keras: https://keras.io/api/models/model_training_apis/

Kai Zhang, S. B. (15 de 04 de 2013). *Air pollution and health risks due to vehicle traffic*. Obtenido de ncbi: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4243514/

Mathworks. (s.f.). *¿Qué es la regresión lineal?* Obtenido de MATLAB & Simulink: https://es.mathworks.com/discovery/linear-regression.html

Milán Jamriska, L. M. (01 de 03 de 2008). *The effect of temperature and humidity on size segregated traffic exhaust particle emissions*. Obtenido de ScienceDirect: https://www.sciencedirect.com/science/article/abs/pii/S135223100701151X

Mohammad Hashem Askariyeh, J. Z. (01 de 03 de 2020). *Traffic contribution to PM2.5 increment in the near-road environment*. Obtenido de ScienceDirect: https://www.scribbr.es/detector-de-plagio/generador-apa/new/webpage/

Mozila. (14 de 07 de 2021). *Trabajando con JSON - Aprende sobre desarrollo web | MDN*. Obtenido de Developer Mozila: https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

Nieves, D. (s.f.). *¿Qué es el escenario de entrenamiento, validación y prueba de conjuntos de datos en aprendizaje automático? - Quora*. Obtenido de Quora: https://es.quora.com/Qu%C3%A9-es-el-escenario-de-entrenamiento-validaci%C3%B3n-y-prueba-de-conjuntos-de-datos-en-aprendizaje-autom%C3%A1tico

Ny1, S. (15 de 02 de 2019). *Nueva York es una de las 5 ciudades con más tráfico en el país*. Obtenido de Spectrum Noticias NY1: https://www.ny1noticias.com/nyc/noticias/noticias/2019/02/15/nueva-york-es-una-de-las-5-ciudades-con-mas-trafico-en-el-pais

NYC Government. (12 de 04 de 2015). *The Official Website of the City of New York*. Obtenido de Manhattan Speed Limits: https://www1.nyc.gov/html/dot/downloads/pdf/current-pre-vision-zero-speed-limit-maps.pdf

Oracle. (s.f.). *¿Qué es la inteligencia artificial (IA)?* Obtenido de Oracle: https://www.oracle.com/mx/artificial-intelligence/what-is-ai/

Pandas. (s.f.). *pandas - Python Data Analysis Library*. Obtenido de Pandas: https://pandas.pydata.org/

Pozzi, S. (02 de 04 de 2019). *Nueva York cobrará por circular en coche en Manhattan*. Obtenido de El País: https://elpais.com/sociedad/2019/03/31/actualidad/1554036405_982184.html

R.PowerData. (27 de 01 de 2017). *¿Qué es Big Data Hadoop y para qué sirve?* Obtenido de Poderdata blog: https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/que-es-big-data-hadoop-y-para-que-sirve

Ramadan, L. (08 de 01 de 2016). *What does the standard Keras model output mean? What is epoch and loss in Keras?* Obtenido de Stack Overflow: https://stackoverflow.com/questions/34673396/what-does-the-standard-keras-model-output-mean-what-is-epoch-and-loss-in-keras

Shaftoe, G. (s.f.). *What part of New York City has the most traffic?* . Obtenido de Quora.

Sierra, Y. (28 de 06 de 2019). *¿Qué es Spark y cómo revoluciona al Big Data y al Machine Learning?* Obtenido de mdcloud: https://blog.mdcloud.es/que-es-spark-big-data-y-machine-learning/

Slate, A. (8 de 7 de 2019). *Qué es una API: todo lo que necesitas saber*. Obtenido de Wrike: https://www.wrike.com/es/blog/que-es-una-api-necesitas-saber/

SmartPanel. (10 de 10 de 2019). *¿Qué es el Deep Learning?* Obtenido de SmartPanel: https://www.smartpanel.com/que-es-deep-learning/

Tensorflow Team. (17 de 06 de 2021). *Time series forecasting | TensorFlow Core*. Obtenido de TensorFlow: https://www.tensorflow.org/tutorials/structured_data/time_series

Univision. (22 de 02 de 2017). *Nueva York es una de las ciudades con el peor tráfico vehicular en el mundo, según estudio*. Obtenido de Univision: https://www.univision.com/local/nueva-york-wxtv/nueva-york-es-una-de-las-ciudades-con-el-peor-trafico-vehicular-en-el-mundo-segun-estudio

Wikipedia, c. d. (10 de 07 de 2021). *Nueva York*. Obtenido de datosmacro: https://es.wikipedia.org/wiki/Nueva_York

Yegulalp, S. (18 de 06 de 2019). *What is TensorFlow? The machine learning library explained*. Obtenido de InfoWorld: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html