



Predicción de intensidades de tráfico rodado en la ciudad de Madrid a partir de Big Data

Alejandro Álvarez Tenedor

Rubén Colinas Merino

José Fidel González Alfonso

María Martínez Alonso

Eva Rodríguez Borrajo

Uxía Taboada Nieto

Septiembre 2023

Índice general

| | |
|---|-----------|
| 1. Introducción: Planteamiento del problema y presentación de los datos. | 1 |
| 2. Tratamiento de la base de datos | 5 |
| 2.1. Limpieza | 5 |
| 2.2. Análisis exploratorio de los datos | 8 |
| 2.3. Creación de nuevas variables calculadas | 8 |
| 2.4. Transformación de variables | 9 |
| 3. Selección del modelo predictivo | 11 |
| 3.1. Comparación de modelos sencillos | 11 |
| 3.2. Grid search de los modelos con mejor puntuación | 12 |
| 3.3. Modelo final. Ajustes, validación y evaluación | 14 |
| 4. Productivización del modelo | 15 |
| 4.1. Servicio Backend | 15 |
| 4.2. Interfaz de usuario | 16 |
| 5. Visualización de resultados | 19 |
| 5.1. Proyecto piloto App móvil | 21 |
| 6. Conclusiones | 23 |
| Código | 25 |
| .1. Script de limpieza de la base de datos | 25 |
| .2. Gráficos del EDA | 40 |
| .2.1. Gráfico de correlaciones | 40 |
| .3. Script de construcción de nuevas variables calculadas | 40 |
| .4. Script de transformación de variables | 60 |
| .5. Script de PCA | 68 |

| | | |
|-------|---|------------|
| .6. | Scripts de comparación de modelos sencillos | 73 |
| .6.1. | Script comparación modelos sencillos dataset normal | 73 |
| .6.2. | Script comparación modelos sencillos dataset transformado | 78 |
| .7. | Scripts de grid searches | 83 |
| .7.1. | Bagging dataset original | 83 |
| .7.2. | Bagging dataset transformado | 93 |
| .7.3. | Decision Tree dataset original | 103 |
| .7.4. | Decision Tree dataset transformado | 175 |
| .7.5. | XGBoost dataset original | 248 |
| .7.6. | XGBoost dataset transformado | 256 |
| .8. | Script del modelo final | 263 |
| .9. | Scripts de la productivización del modelo | 276 |
| .9.1. | transformaciones.py | 276 |
| .9.2. | servicio_modelo.py | 278 |
| .9.3. | app.py | 280 |
| | Bibliografía | 287 |

Capítulo 1

Introducción: Planteamiento del problema y presentación de los datos.

En este TFM se plantea un problema de predicción propuesto por la empresa **MRC Consultants and Transaction Advisers**, consistente en predecir las intensidades de tráfico rodado en grandes ciudades a partir del Big Data. Dicha empresa proporcionó al grupo encargado del trabajo el artículo *Big Data for Traffic Estimation and Prediction: A Survey of Data and Tools* [1], que contiene una amplia colección de fuentes de datos públicas sobre tráfico rodado y del que se obtuvo el dataset a tratar, en concreto la base de datos de tráfico correspondiente al Ayuntamiento de Madrid [2].

El problema de predicción planteado consiste en, partiendo de la base de datos históricos de tráfico de Madrid, aplicar algoritmos de Machine Learning para predecir las intensidades de tráfico rodado.

En la web del Ayuntamiento de Madrid encontramos datos históricos de los puntos de medida de tráfico desde 2013 (cada mes se van incorporando los datos del mes anterior); así pues, para un mismo punto de medida es posible estudiar la evolución del tráfico a lo largo del tiempo. Tras comprobar la volumetría de los datasets y teniendo en cuenta la reciente pandemia del COVID-19, se decidió tomar datos desde enero de 2021 a junio del 2023.

En el dataset tenemos datos registrados e integrados sobre períodos de 15 minutos que los diversos sistemas de control de tráfico de la ciudad de Madrid proporcionan periódicamente y de forma automática. Debido a la gran volumetría de los datos (por el número de sensores de medida y que hay un registro cada 15 minutos por sensor) se decidió hacer dos reducciones del dataset, una primera temporal, en la que se hizo la media de las 4 medidas de cada hora y quedarnos con un solo registro por hora y punto de medida. Tras esta

primera reducción, se hizo una segunda en la que nos quedamos únicamente dos puntos de medida por distrito de la ciudad, ya que conocíamos la posición de cada punto, registrada en [3]. Con estas reducciones conseguimos pasar de un dataset inicial de 88528889 registros, a uno final de 840918.

El dataset consta de las siguientes variables:

- id (Entero). Identificación del punto de medida. Es de tipo secuencial, además de ser único e invariable.
- fecha (Fecha). Fecha y hora oficiales de Madrid con formato dd/mm/yyyy hh:mi:ss.
- tipo_elem (Texto). Nombre del tipo de punto de medida: Urbano o M30.
- intensidad (Entero). Número de vehículos en el periodo de 15 minutos, expresada en vehículos/hora. El valor efectivo de vehículos que han circulado en ese periodo se obtiene dividiendo entre cuatro. Un valor negativo implica la ausencia de datos.
- ocupacion (Entero). Porcentaje de tiempo que está un detector de tráfico ocupado por un vehículo. Por ejemplo, una ocupación del 50 % en un periodo de 15 minutos significa que ha habido vehículos situados sobre el detector durante 7 minutos y 30 segundos. Un valor negativo implica la ausencia de datos.
- carga (Entero). Parámetro de carga del vial en el periodo de 15 minutos. Representa una estimación del grado de congestión, calculado a partir de un algoritmo que usa como variables la intensidad y ocupación, con ciertos factores de corrección. Establece el grado de uso de la vía en un rango de 0 (vacía) a 100 (colapso). Un valor negativo implica la ausencia de datos. En [2] encontramos de forma detallada cómo se calcula este parámetro.
- vmed (Entero). Velocidad media de los vehículos en el periodo de 15 minutos (Km./h). Solo para puntos de medida interurbanos M30. Un valor negativo implica la ausencia de datos.
- error (Texto). Indicación de si ha habido al menos una muestra errónea o sustituida en el periodo de 15 minutos. N: no ha habido errores ni sustituciones E: los parámetros de calidad de alguna de las muestras integradas no son óptimos. S: alguna de las muestras recibidas era totalmente errónea y no se ha integrado.
- periodo_integracion (Entero). Número de muestras recibidas y consideradas para el periodo de integración.

Nuestra variable objetivo sería así la intensidad. Como vemos, las variables carga y ocupación son variables calculadas a partir de la intensidad, por lo que no las introduciremos en nuestro modelo predictivo.

Para tener más variables que ayudasen en la predicción del tráfico, se buscaron otros dos datasets, uno con información sobre festivos en Madrid y otro con información meteorológica en Madrid, que se mergearon con este original por fecha.

En el dataset de festivos en Madrid [4] encontramos las fiestas laborales determinadas en el Decreto de Consejo de Gobierno de la Comunidad de Madrid para nuestra comunidad y los días de fiesta locales en la ciudad de Madrid aprobados por Acuerdo del Pleno del Ayuntamiento. Sus variables son las siguientes:

- fecha (Fecha). Día del año de forma consecutiva en formato dd/mm/aaaa.
- dia_semana (Fecha). Día de la semana correspondiente a la fecha de columna.
- laborable/festivo/domingo festivo (Texto). Para el día del año y el día de la semana señalados en las columnas 1 y 2 se indicará si es laborable, festivo o domingo que coincida con día festivo.
- tipo de festivo (Texto). Se indicará si se trata de festivo nacional, de la Comunidad de Madrid o local (ciudad de Madrid).
- festividad (Texto). Nombre dado al día festivo por la legislación de nuestro país

A este dataset se le hicieron las transformaciones convenientes, de forma que nos quedamos con las fechas que necesitábamos y los siguientes campos calculados a partir de los que había:

- dia (Fecha). Fecha con formato coincidente con el del dataset de tráfico, desglosada por horas.
- fin_de_semana (Binario). Variable binaria que indica si el día es laborable (0) o sábado/domingo (1).
- festivo (Binario). Variable binaria que indica si el día es festivo.
- tipo_de_festivo (Texto). Variable categórica que indica el tipo de festivo: Local, Nacional, Comunidad de Madrid, No festivo.

En el dataset de Meteostat [5] encontramos datos meteorológicos en Madrid. Sus variables son:

- time (Double). Fecha y hora.
- temp (Double). Temperatura media en grados celsius.
- dwpt (Double). Punto de Rocío.
- rhum (Double). Humedad relativa.
- prcp (Double). Precipitación total.
- snow (Double). Profundidad de la nieve.
- wdir (Double). Dirección del viento.
- wspd (Double). Velocidad del viento.
- wpgt (Double). Ráfaga del viento.
- pres (Double). Presión del aire.
- tsun (Double). Duración del sol.
- coco (Double). Código de condición meteorológica.

Una vez mergeados estos tres datasets, nuestro dataset final tiene 23 columnas y 840918 registros.

Capítulo 2

Tratamiento de la base de datos

En este capítulo se trata el procesamiento de la base de datos, su limpieza, análisis exploratorio y posterior construcción de variables calculadas a partir de las originales. Cabe decir que este análisis exploratorio y la posterior limpieza del dataset se hicieron después de la primera reducción de datos (la temporal en la que nos quedamos con un registro por hora) y antes de la segunda reducción del mismo, en la que nos quedamos con dos puntos de medida por distrito. Tras haber hecho la limpieza se comprobó que los cambios en la naturaleza de los datos y su estructura eran ínfimos tras su reducción, por lo que todo lo dicho en este EDA es aplicable al dataset reducido.

2.1. Limpieza

Todo lo que se explica en esta sección viene del script de limpieza del anexo .1.

Antes de empezar con la depuración de los datos y para que nos sirva de apoyo en esta fase, hacemos una exploración inicial.

Vemos un summary inicial de cada variable:

| summary | id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion | error |
|---------|--------------------|-----------|--------------------|-------------------|--------------------|--------------------|---------------------|----------|
| count | 90092529 | 90092529 | 90092529 | 90040204 | 90092529 | 90066389 | 90092529 | 89745434 |
| mean | 6357.065334385274 | null | 343.30340818826386 | 4.893551984844459 | 16.298286720311737 | 4.713893492499183 | 13.557078190134945 | null |
| stddev | 163037.29669535195 | null | 561.7943751728709 | 8.939960537523644 | 16.749513237023628 | 17.660497251793988 | 2.851153857375059 | null |
| min | 31 | M30 | 0 | 0 | 0 | 0 | 1 | N |
| max | 1140850688 | URB | 99999 | 100 | 100 | 182 | 26 | N |

| fin_de_semana | festivo | tipo_de_festivo | temp | dwpt | rhum | prcp |
|--------------------|----------------------|---------------------|--------------------|-------------------|-------------------|----------------------|
| 88557303 | 88557303 | 88557303 | 90052373 | 90052373 | 90052373 | 90040248 |
| 0.2811510869973084 | 0.038680209129675056 | null | 15.081899976157331 | 4.972165917274387 | 58.02114935938445 | 0.050336300717096154 |
| 0.4495610698889969 | 0.19283166485628764 | null | 9.019192798414469 | 5.179011799645601 | 24.74248794687838 | 0.2960679320829629 |
| 0 | 0 | Comunidad de Madrid | -10.2 | -18.2 | 5.0 | 0.0 |
| 1 | 1 | No Festivo | 41.5 | 17.5 | 100.0 | 7.5 |

| wdir | wspd | pres | coco | anno | mes | dia | hora |
|-------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| 90052373 | 90052373 | 90052373 | 38526970 | 90092529 | 90092529 | 90092529 | 90092529 |
| 150.5898034802481 | 10.78386248629307 | 1018.2986879302208 | 2.5806564336619258 | 2021.8088031583618 | 5.960264385518582 | 15.720775448539134 | 11.542515950462441 |
| 106.4319149970997 | 5.6898772128816315 | 6.289763523819925 | 2.2303045744435837 | 0.750741275531618 | 3.4032201001296514 | 8.791355210332863 | 6.916008166283234 |
| 0.0 | 0.0 | 997.7 | 1 | 2021 | 1 | 1 | 0 |
| 360.0 | 41.4 | 1040.0 | 9 | 2023 | 12 | 31 | 23 |

Comenzamos con el tratamiento de las variables numéricas.

Vemos que para intensidad el valor máximo es 99999, es decir, tenemos outliers. Haciendo un count vemos que solamente hay 5 registros con este valor de la intensidad, por lo que al suponer un porcentaje tan pequeño sobre el total, los eliminamos. Hacemos un nuevo summary para ver cómo queda la variable.

| summary | intensidad |
|---------|-------------------|
| count | 90092524 |
| mean | 343.2978774465237 |
| stddev | 561.3036322120458 |
| min | 0 |
| max | 93771 |

Como vemos, hay más valores atípicos además del 99999. Vamos a calcular la media de la variable intensidad y su desviación estándar. Consideraremos outliers todos aquellos que superen el umbral de $+/- 3$ veces la desviación estándar.

```
# Calcular la desviacion estandar y el promedio de la variable numerica
stats = df.select(stddev Samp("intensidad").alias("stddev"),
                  avg("intensidad").alias("avg")).collect()[0]
stddev = stats["stddev"]
avg_value = stats["avg"]

# Definir un umbral (por ejemplo, 3 desviaciones estandar)
threshold = 3 * stddev

# Filtrar los valores que NO son outliers y calcular la media
imputed_value = df.filter(~((col("intensidad") < avg_value - threshold) |
                             (col("intensidad") > avg_value + threshold)) \ 
                           .select(avg("intensidad").alias("media_imputada")) \ 
                           .collect()[0]["media_imputada"])

# Imputar los outliers en la columna original
df = df.withColumn("intensidad",
                    when(((col("intensidad") < avg_value - threshold) |
                          (col("intensidad") > avg_value + threshold)),
                          imputed_value).otherwise(col("intensidad")))
```

Hacemos de nuevo un summary para comprobar que la variable está correcta.

```
+-----+-----+
|summary|      intensidad|
+-----+-----+
|  count|    90092529|
|  mean|283.95715588321235|
| stddev| 344.7513996174725|
|  min|        0.0|
|  max|    2028.0|
+-----+-----+
```

Vemos que para las variables temp, dwpt, rhum, wdir, wspd, pres hay 40113 nulls y comprobamos que los registros con temp, dwpt, rhum, wdir, wspd y pres a null son los mismos, por lo que al suponer un 0,04 % del total, se eliminan. Por último, el porcentaje de registros con prcp a nulls es un 0,01 % del total, por lo que los eliminamos.

Tratamos a continuación las categóricas.

```
+-----+
|error|  count|
+-----+
|  null|  347021|
|     N|89614805|
+-----+
```

Vemos que error, que debería tener las categorías N,E,S, solamente tiene N y nulls. Esto se debe a algún problema de lectura del dataset en el que las categorías relacionadas con error han sido leídas como nulls, por lo que se decide recategorizar todos los nulls como una nueva categoría, E/S, que agrupa a las dos categorías perdidas y que sigue siendo informativa, ya que de esta manera tenemos en una categoría las mediciones con error y en otra las que no tienen error.

```
+-----+
|coco|  count|
+-----+
|   7|  660364|
|   3|16370051|
|   8|  852868|
|null|51472356|
|   5|  777443|
|  18|  68480|
|  17|  340179|
|   9|  513591|
|  1|14568486|
|   2|  4338008|
+-----+
```

Vemos que la variable coco tiene más de la mitad de registros nulos, por lo que se elimina esta variable.

Por último, observamos que en las variables del dataset de festivos hay el mismo número de nulls, lo que lleva a pensar que puedan ser los mismos registros. Como podemos ver en .1, los missings se corresponden a la segunda quincena del mes de junio de 2023. Esto es

debido a que se hizo un left join entre el dataset de tráfico, el cual tenía datos hasta el 30 de junio de 2023, y el de festividades, cuya fecha máxima era el 15 de junio de 2023. Dado que en el mes de junio de 2023 no hubo festivos en Madrid, se puede actualizar dicha información manualmente de forma sencilla en nuestro dataframe (ver .1).

Terminamos con esto la limpieza del dataset.

2.2. Análisis exploratorio de los datos

Veamos cómo quedan las estadísticas de las variables numéricas tras la limpieza de los datos.

| | summary | id | intensidad | ocupacion | carga | vmed | periodo_integracion | |
|--------------------|---------------------|-------------------|-------------------|----------------------|--------------------|--------------------|---------------------|--------------------|
| count | 89961826 | 89961826 | 89961826 | 89961826 | 89961826 | 89961826 | 89961826 | |
| mean | 6358.025357422158 | 284.1029339780394 | 4.892596633154156 | 16.308691066364084 | 4.682590346709948 | 13.561624238263017 | 13.561624238263017 | |
| stddev | 163155.65954771815 | 344.8126710539194 | 8.934827606247323 | 16.747439240650376 | 17.612828013228278 | 2.8448382350580435 | 2.8448382350580435 | |
| min | 31 | 0.0 | 0 | 0 | 0 | 1 | 1 | |
| max | 1140850688 | 2028.0 | 100 | 100 | 182 | 26 | 26 | |
| <hr/> | | | | | | | | |
| | periodo_integracion | temp | dwpf | rhum | prcp | wdir | wspd | pres |
| 89961826 | 89961826 | 89961826 | 89961826 | 89961826 | 89961826 | 89961826 | 89961826 | 89961826 |
| 13.561624238263017 | 15.08255104676259 | 4.971606560101921 | 58.01820641123936 | 0.050338491350424425 | 150.588899407371 | 10.783987604885018 | 1018.2985520643715 | 1018.2985520643715 |
| 2.8448382350580435 | 9.019974136236394 | 5.179395015565291 | 24.74368207762321 | 0.296085631467945 | 106.42874639803726 | 5.690216297853146 | 6.29028727520111 | 6.29028727520111 |
| 1 | -10.2 | -18.2 | 5.0 | 0.0 | 0.0 | 0.0 | 997.7 | 997.7 |
| 26 | 41.5 | 17.5 | 100.0 | 7.5 | 360.0 | 41.4 | 1040.0 | 1040.0 |

Así, en el apartado de limpieza ya se han comprobado los límites de las variables cuantitativas, las categorías de las variables cualitativas y la representatividad de las categorías minoritarias de las variables cualitativas.

Para analizar la correlación entre las variables, tenemos un gráfico en el anexo de EDA .2. Como podemos observar en el gráfico de correlaciones .2.1, claramente las variables intensidad, carga y ocupación están muy correlacionadas, ya que son variables calculadas unas a partir de las otras. Similarmente pasa con vmed. Vemos también que, como es lógico, condición adversa está fuertemente inversamente relacionada con intensidad de tráfico.

2.3. Creación de nuevas variables calculadas

Una vez tenemos nuestro dataset limpio y hemos analizado qué variables nos pueden ser de más interés a la hora de predecir la intensidad, vamos a construir nuevas variables a partir de los datos que ya tenemos que puedan aportar información y valor predictivo al modelo.

Todo lo que se explica en esta sección viene del script de limpieza del anexo .3.

El primer bloque de variables calculadas es a partir de la fecha. Vemos que puede ser

útil tener una variable que indique la estación del año (**estacion**) y otra que indique qué dia de la semana es (**dia_semana**).

Creamos también la variable **hora_punta**. Para crear esta variable vamos a tener en cuenta varios factores. Se establece primero que la hora punta sólo se aplicará para días laborables, es decir, de lunes a viernes. También se asume que la hora punta cambiará en los meses de verano puesto que muchas empresas tienen jornada intensiva. Hacemos un estudio de qué ocurre con las intensidades medias por meses (ver .3) y vemos que claramente hay una diferencia grande entre los meses de verano y el resto del año, se observa inclusive una diferencia entre los propios julio y agosto. De esta manera, decidimos dividir los meses en tres bloques: septiembre a junio, julio y agosto. Una vez hecho esto, estudiamos cuáles son las horas del día en las que hay más tráfico para cada bloque y creamos la variable **hora_punta** con los resultados, de forma que las horas puntas para los días laborables son:

- sept - jul: 9:00, 14:00-15:00, 19:00
- agosto: 12:00-15:00, 19:00

Por su parte, las horas puntas los fines de semana son:

- sept - jul: 12:00-14:00, 19:00-21:00
- agosto: 12:00-14:00, 20:00-21:00

Se decide crear la variable **condicion_adversa**, considerando como condición adversa una temperatura demasiado alta o demasiado baja (teniendo en cuenta la media) y una cantidad de precipitación considerable.

Otra variable que se considera importante es **reduccion_tp**, una binaria que indica si se está en horario de reducción de transporte público (entre la 1 y las 6 de la madrugada) o no.

Relacionadas con las vacaciones, se crean las variables binarias **puente** y **periodo_vacacional**, considerándose puente aquellos días en los que hay tres o más días no laborables seguidos, y periodo vacacional a semana santa, verano (15-31 de julio y agosto) y navidad. Asimismo se crea la variable binaria **operacion_salida**, considerándose un día como operación salida si es el día anterior a un puente o periodo vacacional o el último día de uno de estos.

2.4. Transformación de variables

Para contar con datos que pudiesen predecir mejor en nuestros modelos, se decidió crear un dataset con las variables numéricas del original transformadas, de forma que estas nuevas

variables transformadas maximizaran la correlación con la variable numérica objetivo. Para esto se creó una función que eligiese de entre las posibles transformaciones (logaritmo, exponencial, potencia, raiz) la que maximizaba la correlación. Pasamos así a tener un nuevo dataset, que llamamos dataset_transformado, en el que las variables numéricas son resultado de haber sufrido la mejor transformación en terminos de correlación con la variable objetivo. Todo este proceso está hecho en el script de transformación de variables .4.

Así mismo, se hizo otro script en el que se hace un análisis de componentes principales y guardamos un nuevo dataset con las pca calculadas. Ver .5.

De esta forma, contamos con tres datasets para hacer una primera búsqueda de nuestro mejor modelo.

Capítulo 3

Selección del modelo predictivo

Una vez tenemos nuestro dataset listo y hemos analizado las variables que mejor pueden ir en nuestro modelo, buscamos el modelo ideal para nuestro problema de predicción.

3.1. Comparación de modelos sencillos

Los resultados discutidos en esta sección se tratan en .6.

Tras hacer una investigación sobre cuálesara hacer una primera criba de modelos, se entrenaron de manera sencilla para cada uno de los tres datasets anteriormente mencionados los siguientes modelos: Regresión Lineal, Lasso, regresión Ridge, Decision Tree, KNN, Elastic net, MLP, Random Forest, XGBoost, Gradient Boosting y Bagging, del paquete sklearn de python. Los resultados para el dataset de PCA no eran mejores que los de los datasets normal y filtrado, por lo que sumando el coste computacional que tuvo hacer el PCA, decidimos descartar este dataset y seguir probando los mejores modelos de los anteriores con los datasets normal y filtrado.

Se volvió a hacer un entrenamiento sencillo de modelos para estos dos datasets, con los modelos Decision Tree, KNN, Random forest, XGBoost, Gradient Boosting y Bagging para el dataset normal, que habían sido los de mayor puntuación en la primera tirada (Ver script .6.1). Se obtuvieron los siguientes resultados:

| Modelo | R2 Score | RMSE |
|--------|----------|----------|
| DTR | 0.8276 | 150.4674 |
| KNNR | 0.6501 | 214.3609 |
| RFR | 0.9073 | 110.3665 |
| XGB | 0.8538 | 138.5486 |
| GBR | 0.6451 | 215.8913 |
| BAG | 0.8969 | 116.3740 |

Para el dataset transformado los resultados obtenidos fueron:

| | R2 Score | RMSE |
|--------|----------|----------|
| Modelo | R2_Score | RMSE |
| LR | 0.2479 | 314.2910 |
| DTR | 0.8291 | 149.8236 |
| Lasso | 0.2450 | 314.9030 |
| Ridge | 0.2479 | 314.2910 |
| RFR | 0.9064 | 110.8903 |
| EN | 0.1922 | 325.7185 |
| XGB | 0.8634 | 133.9448 |
| GBR | 0.6464 | 215.4918 |
| BAG | 0.8958 | 116.9588 |

Así pues, decidimos hacer un grid search para mejorar los tres mejores modelos: XGBoost, Decision Tree y Bagging (Omitimos Random forest por ser un modelo propenso al sobreajuste y dar unos resultados tan buenos en una primera tirada).

3.2. Grid search de los modelos con mejor puntuación

Los resultados discutidos en esta sección se tratan en .7.

Se hizo un grid search para los modelos XGBoost, Decision Tree y Bagging con los datasets normal y filtrado. Los parámetros y resultados de cada uno se pueden ver en .7. Los resultados de la evaluación de cada modelo son los siguientes:

| DATASET | Modelo | RMSE CrossValidation | R2 CV Train | Desviación std CV | Precisión CV | R2 CV Test | Desviación std CV | Precisión CV |
|--------------|----------------|----------------------|-------------|-------------------|--------------|------------|-------------------|--------------|
| Filtrado | Bagging | 132,01 | 0,8641 | 0,001 | 0,86+-0,00 | 0,8473 | 0,004 | 0,85+-0,00 |
| | Decission Tree | 138,73 | 0,8484 | 0,001 | 0,85+-0,00 | 0,8169 | 0,005 | 0,82+-0,01 |
| | XGB | 126,67 | 0,8744 | 0,001 | 0,87+-0,00 | 0,8676 | 0,005 | 0,87+-0,01 |
| Transformado | Bagging | 118,20 | 0,8926 | 0,002 | 0,89+-0,00 | 0,8738 | 0,003 | 0,87+-0,01 |
| | Decission Tree | 137,85 | 0,8492 | 0,001 | 0,85+-0,00 | 0,8176 | 0,004 | 0,82+-0,01 |
| | XGB | 126,36 | 0,8743 | 0,002 | 0,86- 0,00 | 0,8676 | 0,004 | 0,85+-0,01 |

Obtenemos también la curva de aprendizaje de cada modelo, que nos permite ver cómo evolucionan las accuracies en entrenamiento y test y así comprobar si hay sobreentrenamiento. 3.1

Si ambas curvas convergen a un valor similar y alto, el modelo está generalizando bien y no hay overfitting. Si la curva de entrenamiento está por encima de la curva de prueba y no convergen, podría haber overfitting. Si ambas curvas están por debajo de un valor deseado, podría haber underfitting.

Vemos que los resultados para los modelos con el dataset original son ligeramente mejores que los del dataset transformado, y las curvas de aprendizaje son prácticamente

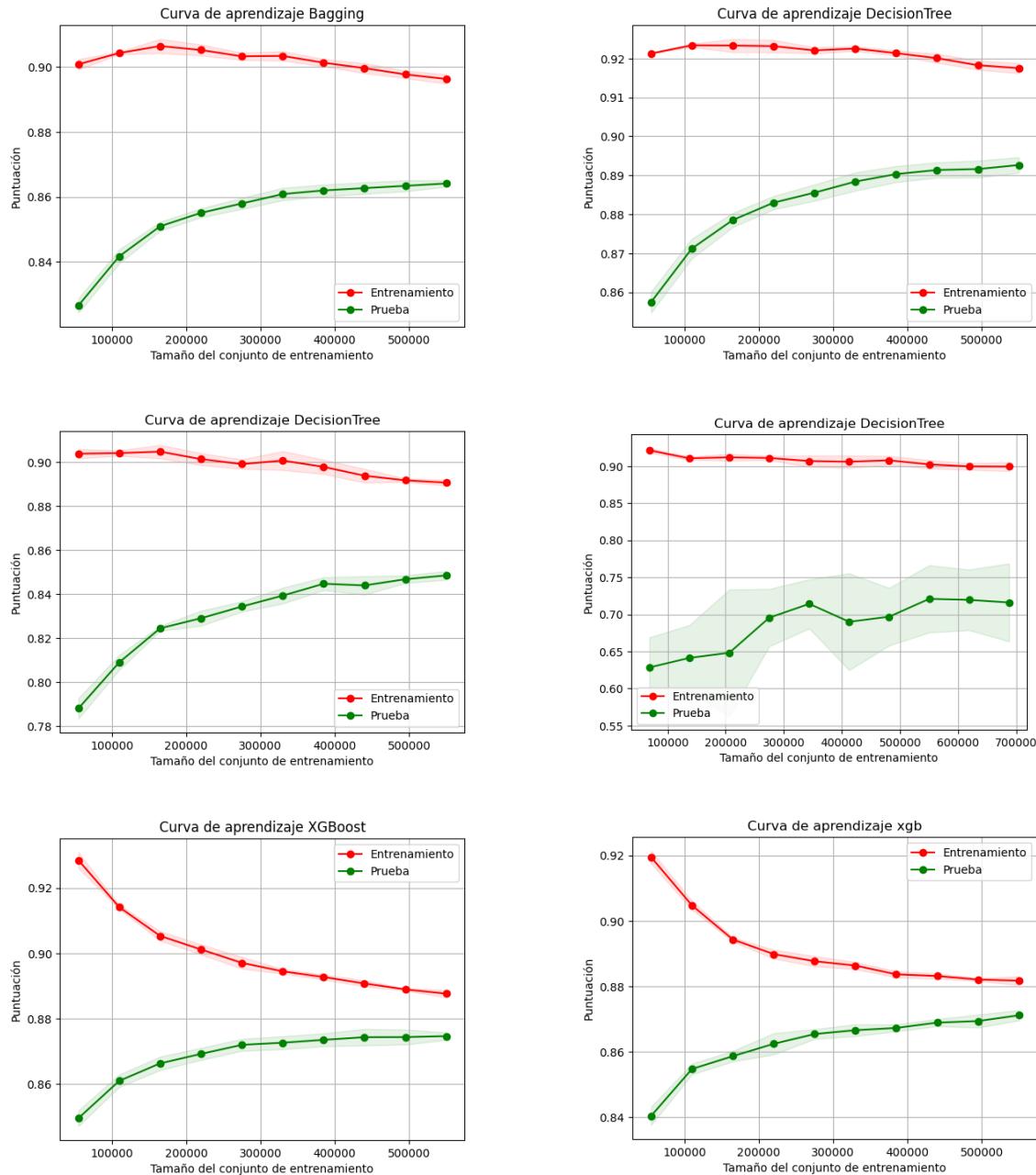


Figura 3.1: Curvas de aprendizaje, a la izquierda modelos con el dataset original y a la derecha modelos con el dataset con variables transformadas.

iguales. Por esto y por ahorrar el coste computacional de transformar todo el dataset, decidimos dejar como finalistas los modelos Bagging, Decision Tree y XGBoost con el dataset original.

3.3. Modelo final. Ajustes, validación y evaluación

Los resultados discutidos en esta sección se tratan en .8.

En los scripts del apartado anterior, se hizo un gráfico para cada modelo de la importancia de las variables (ver .7), así que para elegir al modelo ganador, volvimos a entrenar los modelos finalistas, Bagging, Decision Tree y XGBoost con el dataset original, solamente con las variables más significativas.

Las variables con las que se ha entrenado cada uno de estos modelos son:

- **Bagging:** id, hora, tipo_elem, fin_de_semana, mes, vacacional, anno, dia, temp, pres, dwpt, wdir, rhum, wspd.
- **XGBoost:** id, hora, tipo_elem, fin_de_semana, festivo, dia_semana, hora_punta, estacion, vacacional, reduccion_tp, anno, mes.
- **Decision Tree:** id, hora, tipo_elem, mes, fin_de_semana, anno, vacacional, dia, temp, festivo, pres, dwpt, wdir, wspd, rhum.

Así, las métricas de evaluación de estos tres modelos finalistas pueden verse en el siguiente gráfico: 3.2

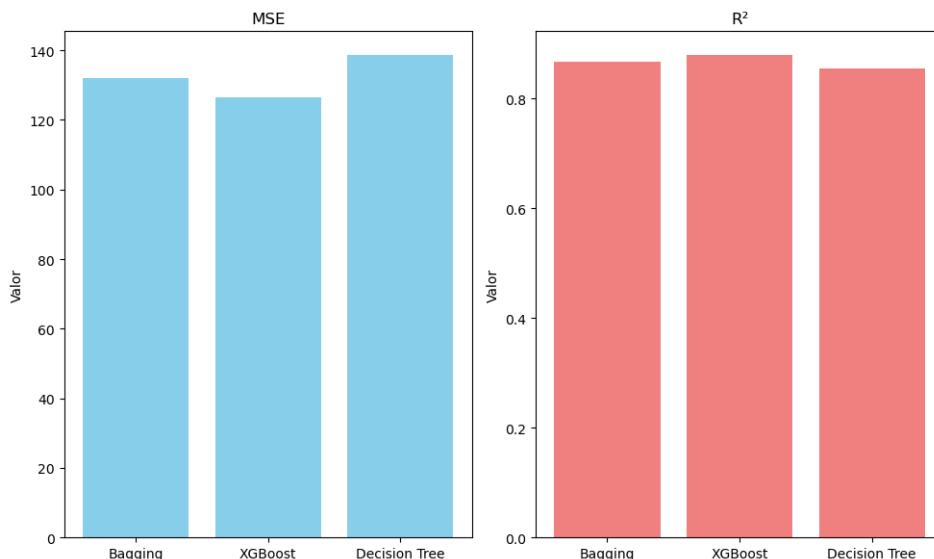


Figura 3.2: Métricas MSE y R^2 para los modelos Bagging, XGBoost y Decision Tree

Vemos que el modelo con mejores MSE y R^2 es XGBoost para el dataset original con las variables anteriormente mencionadas, y por tanto, es nuestro modelo final.

Capítulo 4

Productivización del modelo

Una vez determinado el mejor modelo que, en nuestro caso es el XGBoost entrenado con el dataset original, y determinar los hiperparámetros que maximizan la precisión de este, el último paso será productivizar dicho modelo.

En este capítulo vamos a tratar en detalle la puesta en producción de nuestro modelo, detallando los diferentes módulos que conforman dicha puesta en producción así como diferentes aplicaciones de esta.

Se ha implementado una interfaz de usuario utilizando Streamlit para facilitar la interacción con el modelo de predicción. Además, se ha creado un servicio backend con Flask que sirve como punto de entrada para realizar predicciones en producción.

A continuación, explicamos de forma detallada cada uno estos módulos:

4.1. Servicio Backend

El servicio backend será el encargado de realizar las predicciones de los datos insertados mediante la interfaz de usuario. Dicho servicio comenzará leyendo el fichero .pkl en el que se encuentra alojado nuestro modelo XGBoost preentrenado.

Tras esto, se define una función *predict* la cual recibe como parámetro de entrada un json que contiene todas las variables con las que fue entrenado el modelo. Con dichos datos, se el modelo XGBoost cargado realizará la predicción para posteriormente ser devuelta por dicha función *predict*.

Definida dicha función para realizar la predicción, crearemos un endpoint haciendo uso de la librería Flask, mediante el cual dicho servicio estará escuchando peticiones POST a la ruta '/predict'. Todas las peticiones POST externas vendrán acompañadas de un json con las variables de entrada de nuestro modelo. Dicho json será la entrada de la función *predict* explicada en el párrafo anterior, y mediante la cual se realizarán las predicciones.

Con el objetivo de proporcionar portabilidad, aislamiento y reproducibilidad a nuestro servicio backend se ha empaquetado la aplicación y todas sus dependencias en un contenedor Docker, lo que facilita su ejecución en diferentes entornos sin problemas de compatibilidad. Además, mediante la dockerización conseguimos una gestión más eficiente de versiones, escalabilidad, despliegue y colaboración así como la optimización del uso de recursos en todos los niveles de desarrollo nuestro proyecto.

4.2. Interfaz de usuario

Definido el backend, el siguiente paso fue desarrollar una interfaz gráfica desde la que el usuario pudiese ingresar los datos a partir de los cuales desea obtener una predicción de la intensidad del tráfico. Para el desarrollo de dicha interfaz se ha utilizado la librería de Python Streamlit la cual destaca por su simplicidad, interactividad y una buena integración con modelos de Machine Learning, permitiendo mostrar los resultados de forma sencilla.

Como se indicó previamente, el modelo XGBoost productivizado requiere como variables de entrada el id, el tipo de elemento (M30 o URB), la fecha, la hora y el resto de nuevas variables creadas a partir de estas últimas como son festivo, fin de semana, día de la semana, estación, etc. Es por este motivo por lo que, como se muestra en la figura 4.1, en la interfaz web, únicamente será necesario introducir el distrito, la fecha y la hora en la que se quiere predecir la intensidad de tráfico.

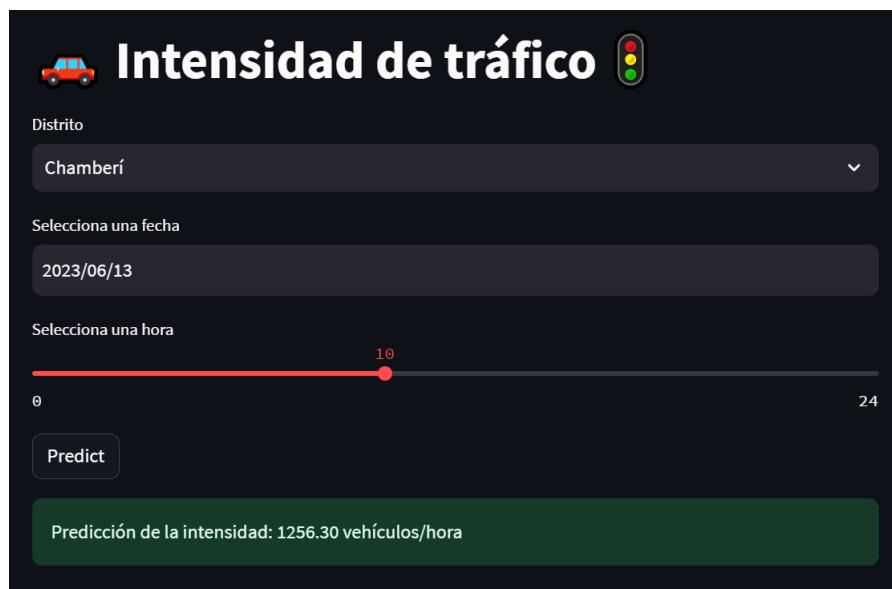


Figura 4.1: Interfaz de usuario para la predicción de la intensidad de tráfico

Una vez introducidas las variables y antes de realizar la petición post enviando el json con las variables de entrada de nuestro modelo se deberán realizar dos operaciones en el propio cliente:

1. Creación de las nuevas variables: a partir de la fecha y la hora será necesario crear las variables adicionales de entrada con las que el modelo fue entrenado.
2. Dummificar las variables categóricas: creadas dichas variables, será necesario dummificar las categóricas para disponer así, de la misma naturaleza de variables con las que el modelo fue entrenado.

Tras el proceso anteriormente mencionado, ya se podrá pasar a realizar la predicción. Cuando el usuario presiona el botón de predicción, se realizará una llamada POST al método `'/predict'` pasando como argumento las variables de entrada en formato json. Posteriormente, se realizará la predicción en el backend, siendo devuelta al frontal desde donde se analizará si la respuesta es correcta y, en caso de serlo, se mostrará la predicción realizada. .9.

Capítulo 5

Visualización de resultados

Como paso final, hemos desarrollado un dashboard interactivo en el que podemos visualizar los resultados reales así como los predichos por nuestro modelo

Enlace al dashboard.

Para eso contamos con varias vistas en las que los valores de la intensidad aparecen desglosados por horas, días y distritos. 5.2 - 5.3 5.1

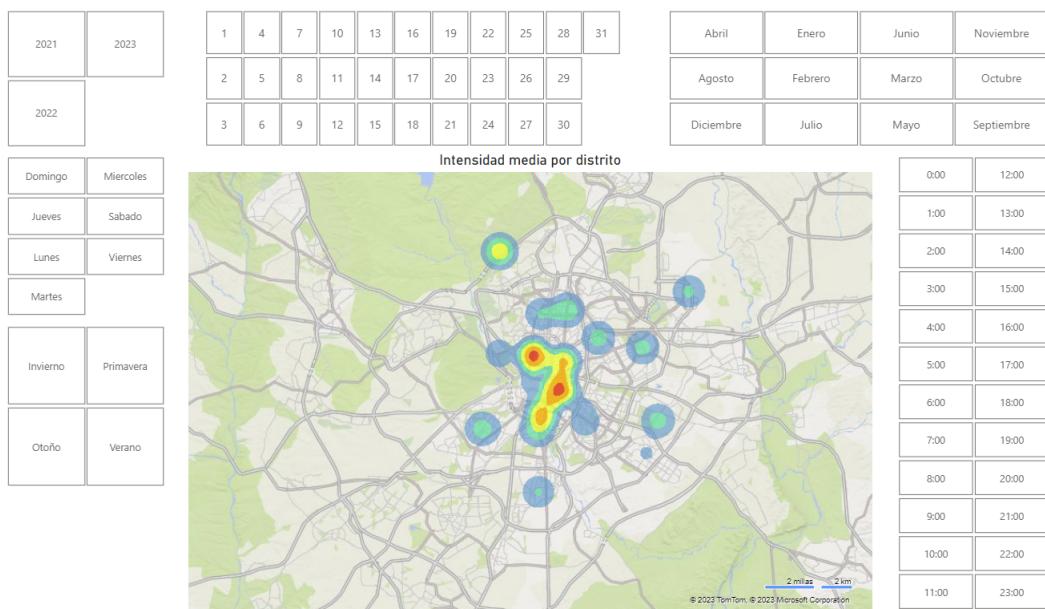


Figura 5.1: Intensidad media por distritos

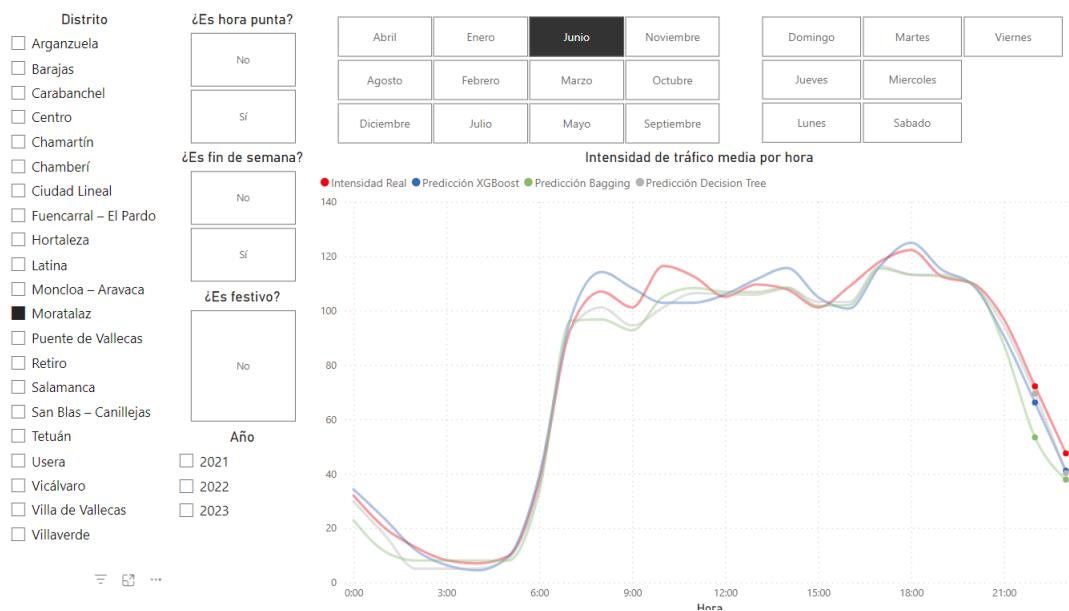


Figura 5.2: Intensidad media por hora vs predicciones

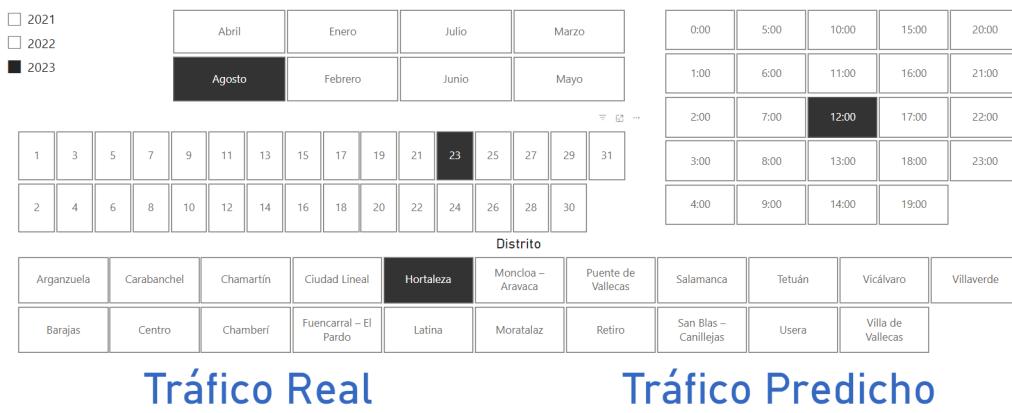
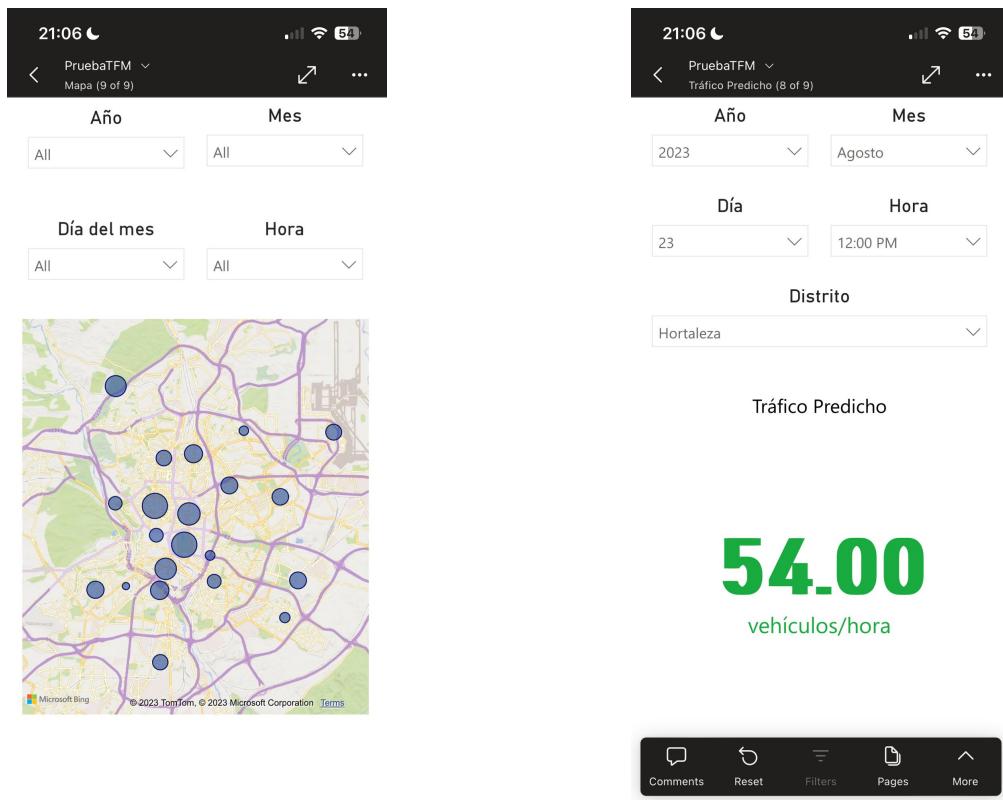


Figura 5.3: KPI de intensidad real vs predicha

Los datos están entrenados con observaciones entre enero de 2021 hasta junio de 2023. En el tiempo que hemos desarrollado el modelo, los datos de julio y agosto fueron subidos a la página de la DGT, por lo que nos pareció interesante añadirlo a nuestro modelo para ver como de precisas eran las predicciones de datos que no estaban en el modelo entrenado. Vemos que salvo contados ejemplos, las predicciones de nuestro modelo se aproximan mucho a los valores reales de la intensidad también en los meses que no están en el modelo.

5.1. Proyecto piloto App móvil

Viendo la precisión del modelo, se nos ocurrió llevarlo más allá y empezar un piloto de App móvil que podría ser útil para que la gente pudiera anticiparse a situaciones desfavorables de tráfico y que, por ejemplo, pudiera visualizar mediante un mapa de calor los distritos que podrían evitarse para saltarse un tráfico intenso o cuando deberían o no optar por transporte público para evitar un atasco.



Capítulo 6

Conclusiones

En este proyecto, hemos abordado el problema de predicción de intensidades de tráfico rodado en grandes ciudades utilizando técnicas de Machine Learning y Big Data. El problema que hemos abordado supone un caso de gran relevancia en la actualidad, ya que la gestión del tráfico en las grandes ciudades es un desafío crítico que afecta tanto a la movilidad de los ciudadanos como a la eficiencia de las ciudades. La capacidad de predecir con precisión las intensidades de tráfico puede ayudar a tomar decisiones informadas para abordar problemas de congestión y mejorar la planificación urbana.

La calidad de los datos es fundamental para el éxito de cualquier modelo predictivo. Durante este proyecto, se realizó un extenso trabajo de limpieza y preprocesamiento de los datos para garantizar que fueran adecuados para su uso en el entrenamiento de modelos. Después de comparar varios modelos de Machine Learning, se seleccionó el modelo XGBoost como el más adecuado para nuestro problema. Este modelo demostró un buen equilibrio entre rendimiento y capacidad de generalización.

La productivización del modelo es un componente crítico en cualquier proyecto de Machine Learning, especialmente cuando se trata de aplicaciones del mundo real como la predicción de intensidades de tráfico en ciudades. Es por esto que le dimos gran peso a este apartado en nuestro proyecto, ya que nuestro objetivo ha sido buscar una solución con aplicabilidad práctica.

La solución desarrollada en este proyecto tiene aplicaciones potenciales en la gestión del tráfico urbano, permitiendo a las autoridades tomar decisiones más informadas sobre la asignación de recursos y la planificación de infraestructuras.

Código

.1. Script de limpieza de la base de datos

```
In [ ]: from pyspark.sql.functions import *
```

```
In [ ]: from pyspark.sql import SparkSession

# Crea una instancia de SparkSession
spark = SparkSession.builder \
    .appName("script2") \
    .getOrCreate()
```

```
In [ ]: df = spark.read.option("header", "true").parquet("gs://bucket_traffic_est")
df.show()
```

| id | fecha | tipo_elem | intensidad | ocupacion | carga | vm |
|-------|---------------------|-----------|-------------------|-----------|-------|----|
| 16493 | 2023-02-15 21:00:00 | URB | 53.5 | 0.25 | 16.75 | 0 |
| 16493 | 2023-02-15 22:00:00 | URB | 36.75 | 0.0 | 8.75 | 0 |
| 16493 | 2023-02-15 23:00:00 | URB | 22.5 | 0.25 | 5.25 | 0 |
| 16493 | 2023-02-16 00:00:00 | URB | 17.0 | 0.0 | 4.75 | 0 |
| 16493 | 2023-02-16 01:00:00 | URB | 9.25 | 0.0 | 3.25 | 0 |
| 16493 | 2023-02-16 02:00:00 | URB | 1.25 | 0.0 | 0.75 | 0 |
| 16493 | 2023-02-16 03:00:00 | URB | 4.25 | 0.0 | 1.25 | 0 |
| 16493 | 2023-02-16 04:00:00 | URB | 1.25 | 0.0 | 1.25 | 0 |
| 16493 | 2023-02-16 05:00:00 | URB | 1.25 | 0.0 | 0.75 | 0 |
| 16493 | 2023-02-16 06:00:00 | URB | 16.33333333333332 | 0.0 | 4.0 | 0 |
| 16493 | 2023-02-16 07:00:00 | URB | 33.5 | 0.0 | 7.25 | 0 |
| 16493 | 2023-02-16 08:00:00 | URB | 70.0 | 0.25 | 21.25 | 0 |
| 16493 | 2023-02-16 09:00:00 | URB | 98.0 | 1.5 | 30.75 | 0 |
| 16493 | 2023-02-16 10:00:00 | URB | 53.25 | 0.25 | 16.75 | 0 |
| 16493 | 2023-02-16 11:00:00 | URB | 72.5 | 0.5 | 20.5 | 0 |
| 16493 | 2023-02-16 12:00:00 | URB | 60.5 | 1.5 | 17.5 | 0 |
| 16493 | 2023-02-16 13:00:00 | URB | 60.0 | 0.0 | 18.0 | 0 |
| 16493 | 2023-02-16 14:00:00 | URB | 94.25 | 0.75 | 30.25 | 0 |
| 16493 | 2023-02-16 15:00:00 | URB | 107.75 | 0.75 | 34.0 | 0 |
| 16493 | 2023-02-16 16:00:00 | URB | 105.5 | 1.25 | 35.5 | 0 |

only showing top 20 rows

```
In [ ]: df.select("id").distinct().count()
```

```
Out[ ]: 4645
```

```
In [ ]: df.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- fecha: string (nullable = true)
 |-- tipo_elem: string (nullable = true)
 |-- intensidad: string (nullable = true)
 |-- ocupacion: string (nullable = true)
 |-- carga: string (nullable = true)
 |-- vmed: string (nullable = true)
 |-- periodo_integracion: string (nullable = true)
 |-- error: string (nullable = true)
 |-- fin_de_semana: string (nullable = true)
 |-- festivo: string (nullable = true)
 |-- tipo_de_festivo: string (nullable = true)
 |-- temp: string (nullable = true)
 |-- dwpt: string (nullable = true)
```

```

|--- rhum: string (nullable = true)
|--- prcp: string (nullable = true)
|--- wdir: string (nullable = true)
|--- wspd: string (nullable = true)
|--- pres: string (nullable = true)
|--- coco: string (nullable = true)

```

Paso las variables al tipo que tienen que tener (timestamp, int...)

```
In [ ]: df = df.withColumn('id', col('id').cast('int')) \
    .withColumn("fecha", to_timestamp('fecha')) \
    .withColumn('intensidad', col('intensidad').cast('int')) \
    .withColumn('ocupacion', col('ocupacion').cast('int')) \
    .withColumn('carga', col('carga').cast('int')) \
    .withColumn('vmed', col('vmed').cast('int')) \
    .withColumn('periodo_integracion', col('periodo_integracion').cast('in')) \
    .withColumn('fin_de_semana', col('fin_de_semana').cast('int')) \
    .withColumn('festivo', col('festivo').cast('int')) \
    .withColumn('temp', col('temp').cast('double')) \
    .withColumn('dwpt', col('dwpt').cast('double')) \
    .withColumn('rhum', col('rhum').cast('double')) \
    .withColumn('prcp', col('prcp').cast('double')) \
    .withColumn('wdir', col('wdir').cast('double')) \
    .withColumn('wspd', col('wspd').cast('double')) \
    .withColumn('pres', col('pres').cast('double'))
```

```
In [ ]: df.printSchema()
```

```

root
|-- id: integer (nullable = true)
|-- fecha: timestamp (nullable = true)
|-- tipo_elem: string (nullable = true)
|-- intensidad: integer (nullable = true)
|-- ocupacion: integer (nullable = true)
|-- carga: integer (nullable = true)
|-- vmed: integer (nullable = true)
|-- periodo_integracion: integer (nullable = true)
|-- error: string (nullable = true)
|-- fin_de_semana: integer (nullable = true)
|-- festivo: integer (nullable = true)
|-- tipo_de_festivo: string (nullable = true)
|-- temp: double (nullable = true)
|-- dwpt: double (nullable = true)
|-- rhum: double (nullable = true)
|-- prcp: double (nullable = true)
|-- wdir: double (nullable = true)
|-- wspd: double (nullable = true)
|-- pres: double (nullable = true)
|-- coco: string (nullable = true)

```

```
In [ ]: df = df.withColumn("anno", year("fecha")) \
    .withColumn("mes", month("fecha")) \
    .withColumn("dia", dayofmonth("fecha")) \
    .withColumn("hora", hour("fecha"))
```

```
In [ ]: df = df.drop('fecha')
df.show()
```

```

+---+---+---+---+---+---+---+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+---+---+---+---+---+---+---+
```

```

| 6493 | URB | 53 | 0 | 16 | 0 | 14 | N
| 6493 | URB | 36 | 0 | 8 | 0 | 10 | N
| 6493 | URB | 22 | 0 | 5 | 0 | 14 | N
| 6493 | URB | 17 | 0 | 4 | 0 | 15 | N
| 6493 | URB | 9 | 0 | 3 | 0 | 15 | N
| 6493 | URB | 1 | 0 | 0 | 0 | 15 | N
| 6493 | URB | 4 | 0 | 1 | 0 | 15 | N
| 6493 | URB | 1 | 0 | 1 | 0 | 15 | N
| 6493 | URB | 1 | 0 | 0 | 0 | 15 | N
| 6493 | URB | 16 | 0 | 4 | 0 | 9 | N
| 6493 | URB | 33 | 0 | 7 | 0 | 10 | N
| 6493 | URB | 70 | 0 | 21 | 0 | 14 | N
| 6493 | URB | 98 | 1 | 30 | 0 | 14 | N
| 6493 | URB | 53 | 0 | 16 | 0 | 13 | N
| 6493 | URB | 72 | 0 | 20 | 0 | 14 | N
| 6493 | URB | 60 | 1 | 17 | 0 | 13 | N
| 6493 | URB | 60 | 0 | 18 | 0 | 14 | N
| 6493 | URB | 94 | 0 | 30 | 0 | 15 | N
| 6493 | URB | 107 | 0 | 34 | 0 | 15 | N
| 6493 | URB | 105 | 1 | 35 | 0 | 15 | N
+---+---+---+---+---+---+---+---+
only showing top 20 rows

```

In []: df.printSchema()

```

root
|-- id: integer (nullable = true)
|-- tipo_elem: string (nullable = true)
|-- intensidad: integer (nullable = true)
|-- ocupacion: integer (nullable = true)
|-- carga: integer (nullable = true)
|-- vmed: integer (nullable = true)
|-- periodo_integracion: integer (nullable = true)
|-- error: string (nullable = true)
|-- fin_de_semana: integer (nullable = true)
|-- festivo: integer (nullable = true)
|-- tipo_de_festivo: string (nullable = true)
|-- temp: double (nullable = true)
|-- dwpt: double (nullable = true)
|-- rhum: double (nullable = true)
|-- prcp: double (nullable = true)
|-- wdir: double (nullable = true)
|-- wspd: double (nullable = true)
|-- pres: double (nullable = true)
|-- coco: string (nullable = true)
|-- anno: integer (nullable = true)
|-- mes: integer (nullable = true)
|-- dia: integer (nullable = true)
|-- hora: integer (nullable = true)

```

LIMPIEZA

In []:

```

# Contamos filas y columnas
cols = df.columns
print("Los datos tienen {} columnas".format(len(cols)))

rows = df.count()
print("Los datos tienen {} filas".format(rows))

```

```
Los datos tienen 23 columnas  
[Stage 3:=====]> (6 +  
Los datos tienen 90092529 filas
```

Numéricas

```
In [ ]: from IPython.core.display import HTML  
display(HTML("<style>pre { white-space: pre !important; }</style>"))  
df.describe().show()
```

```
[Stage 211:> (0 +  
+-----+-----+-----+-----+  
|summary| id|tipo_elem| intensidad| ocupacio  
+-----+-----+-----+-----+  
| count | 90092529 | 90092529 | 90092529 | 9004020  
| mean | 6357.065334385274 | null | 343.30340818826386 | 4.89355198484445  
| stddev | 163037.29669535195 | null | 561.7943751728709 | 8.93996053752364  
| min | 31 | M30 | 0 |  
| max | 1140850688 | URB | 99999 | 10  
+-----+-----+-----+-----+
```

Tratamiento de las numéricas

1. Tratamiento de los nulls de intensidad (registros con intensidad = 99999)

```
In [ ]: df.where('intensidad = 99999').count()
```

```
Out[ ]: 5
```

Como vemos, sólamente 5 registros tienen el valor del outlier 99999. Al ser un % tan pequeño sobre el total, podemos eliminar dichos registros.

```
In [ ]: df_filtrado_outliers = df.filter(df["intensidad"] != 99999)
```

```
In [ ]: # Calcular estadísticos descriptivos de la intensidad tras eliminar  
intensidad_stats = df_filtrado_outliers.describe("intensidad")  
  
# Mostrar las estadísticas descriptivas  
intensidad_stats.show()
```

```
[Stage 9:=====]> (6 +  
+-----+  
|summary| intensidad|  
+-----+  
| count | 90092524 |  
| mean | 343.2978774465237 |  
| stddev | 561.3036322120458 |  
| min | 0 |  
| max | 93771 |  
+-----+
```

Como vemos, hay más valores atípicos además del 99999. Vamos a calcular la media de la variable intensidad y su desviación estándar. Consideraremos outliers todos aquellos que superen el umbral de + - 3 veces la desviación estándar.

```
In [ ]: # Calcular la desviación estándar y el promedio de la variable numérica  
stats = df.select(stddev Samp("intensidad").alias("stddev"), avg("intensidad"))  
stddev = stats["stddev"]  
avg_value = stats["avg"]  
  
# Definir un umbral (por ejemplo, 3 desviaciones estándar)  
threshold = 3 * stddev
```

```
In [ ]: # Filtrar los valores que NO son outliers y calcular la media  
imputed_value = df.filter(~((col("intensidad") < avg_value - threshold) |  
                           .select(avg("intensidad").alias("media_imputada")) \  
                           .collect() [0] ["media_imputada"])
```

```
In [ ]: # Imputar los outliers en la columna original  
df = df.withColumn("intensidad",  
                    when(((col("intensidad") < avg_value - threshold) |  
                          imputed_value).otherwise(col("intensidad")))
```

Vemos de nuevo las estadísticas de dicha variable intensidad:

```
In [ ]: # Calcular estadísticos descriptivos de la intensidad tras eliminar outliers  
intensidad_stats_imputada = df.describe("intensidad")  
  
# Mostrar las estadísticas descriptivas  
intensidad_stats_imputada.show()
```

```
[Stage 18:=====]  
+-----+  
|summary|      intensidad|  
+-----+  
|  count|      90092529|  
|  mean| 283.95715588321235|  
| stddev| 344.7513996174725|  
|   min|        0.0|  
|   max|      2028.0|  
+-----+
```

Podemos probar a suavizar también mediante una transformación logarítmica y comparar distribuciones. Al ser dicha variable la variable objetivo debemos analizarla con detenimiento.

```
In [ ]: df.describe("ocupacion").show()  
df.describe("vmed").show()
```

```
+-----+  
|summary|      ocupacion|  
+-----+
```

```

| count | 90040204 |
| mean | 4.893551984844459 |
| stddev | 8.939960537523644 |
| min | 0 |
| max | 100 |
+-----+

```

```

[Stage 39:=====] (6 +)
+-----+
| summary | vmed |
+-----+
| count | 90066389 |
| mean | 4.713893492499183 |
| stddev | 17.660497251793984 |
| min | 0 |
| max | 182 |
+-----+

```

2. En ocupación el count da 90040204, pero el total de registros es 90092529, dónde están los que faltan?? Algo parecido pasa para vmed, hay 90066389, tb para festivo, tipo_de_festivo, fin_de_semana, y las variables del tiempo (igual estos campos aparecen vacíos en los registros, mirarlo)

Eran nulls, así que vamos a tratarlos

2.1 nulls de ocupacion

```
In [ ]: registros_faltantes = df.filter(df.ocupacion.isNull() | (df.ocupacion == registros_faltantes.show())
[Stage 21:> (0 +)
+-----+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+-----+-----+-----+-----+-----+-----+
| 6827| M30 | 108.0 | null | 0 | 69 | 5 | N
| 6827| M30 | 51.0 | null | 0 | 51 | 5 | N
| 6827| M30 | 117.0 | null | 0 | 70 | 5 | N
| 6827| M30 | 102.0 | null | 0 | 44 | 5 | N
| 6827| M30 | 99.0 | null | 0 | 60 | 5 | N
| 6827| M30 | 105.0 | null | 0 | 61 | 5 | N
| 6827| M30 | 93.0 | null | 0 | 60 | 5 | N
| 6827| M30 | 60.0 | null | 0 | 46 | 5 | N
| 6827| M30 | 84.0 | null | 0 | 49 | 5 | N
| 6827| M30 | 57.0 | null | 0 | 55 | 5 | N
| 6827| M30 | 195.0 | null | 0 | 65 | 5 | N
| 6827| M30 | 63.0 | null | 0 | 55 | 4 | N
| 6827| M30 | 30.0 | null | 0 | 36 | 5 | N
| 6827| M30 | 135.0 | null | 0 | 63 | 5 | N
| 6827| M30 | 84.0 | null | 0 | 57 | 5 | N
| 6827| M30 | 75.0 | null | 0 | 55 | 5 | N
| 6827| M30 | 117.0 | null | 0 | 62 | 5 | N
| 6827| M30 | 84.0 | null | 0 | 52 | 5 | N
| 6827| M30 | 66.0 | null | 0 | 58 | 5 | N
| 6827| M30 | 27.0 | null | 0 | 27 | 5 | N
+-----+
only showing top 20 rows
```

En el documento de la estructura del dataset de tráfico histórico dice que para la variable ocupación, un valor negativo implica la ausencia de datos. Vamos a contar cuantos registros sin ocupación tenemos

```
In [ ]: nulls = df.where(df.ocupacion.isNull()).count()
print(nulls)
print(f"porcentaje del total que representan estos nulls: {nulls*100/df.c
52325
[Stage 25:=====>
porcentaje del total que representan estos nulls: 0.058079177686309595
```

Como el porcentaje es muy pequeño, eliminamos los registros con ocupación a null.

```
In [ ]: df = df.dropna(subset=["ocupacion"])
df.count()
```



```
Out[ ]: 90040204
```

2.2 nulls de vmed

```
In [ ]: nulls = df.where(df.vmed.isNull()).count()
print(nulls)
print(f"porcentaje del total que representan estos nulls: {nulls*100/df.c
26140
[Stage 34:=====
porcentaje del total que representan estos nulls: 0.02903147576164976
```

De nuevo el porcentaje de registros con nulls es insignificante, por lo que los eliminamos

```
In [ ]: df = df.dropna(subset=["vmed"])
print(df.where(df.vmed.isNull()).count())
print(df.count())
```



```
0
[Stage 40:=====
90014064
```

2.3 nulls de temp, dwpt, rhum, prcp, wdir, wspd, pres

```
In [ ]: tiempo = ["temp", "dwpt", "rhum", "prcp", "wdir", "wspd", "pres"]

for variable in tiempo:
    nulls = df.where(col(variable).isNull()).count()
    print(f"{variable}: nulls: {nulls}, porcentaje del total que represen
```



```
temp: nulls: 40113, porcentaje del total que representan estos nulls: 0.0
```

```
[Stage 52:=====] (5 +)
dwpt: nulls: 40113, porcentaje del total que representan estos nulls: 0.0

rhum: nulls: 40113, porcentaje del total que representan estos nulls: 0.0

prcp: nulls: 52238, porcentaje del total que representan estos nulls: 0.0

wdir: nulls: 40113, porcentaje del total que representan estos nulls: 0.0

wspd: nulls: 40113, porcentaje del total que representan estos nulls: 0.0

[Stage 82:=====] (6 +)
pres: nulls: 40113, porcentaje del total que representan estos nulls: 0.0
```

Seguramente, los registros con temp, dwpt, rhum, wdir, wspd y pres a null sean los mismos, vamos a comprobarlo

```
In [ ]: registros_temp_null = df.where(col("temp").isNull())
registros_dwpt_null = df.where(col("dwpt").isNull())
registros_rhum_null = df.where(col("rhum").isNull())
registros_wdir_null = df.where(col("wdir").isNull())
registros_wspd_null = df.where(col("wspd").isNull())
registros_pres_null = df.where(col("pres").isNull())

# Compara los DataFrames resultantes para verificar si son los mismos
son_iguales = (
    registros_temp_null
    .exceptAll(registros_dwpt_null)
    .exceptAll(registros_rhum_null)
    .exceptAll(registros_wdir_null)
    .exceptAll(registros_wspd_null)
    .exceptAll(registros_pres_null)
    .count() == 0
)

if son_iguales:
    print("Los registros con valores nulls en temp, dwpt, rhum, wdir, wsp")
else:
    print("Los registros con valores nulls en temp, dwpt, rhum, wdir, wsp")
```

```
[Stage 99:=====] (206 + 1)
Los registros con valores nulls en temp, dwpt, rhum, wdir, wspd y pres so
```

Efectivamente son los mismos registros que tienen todas esas variables a nulls, y como suponen solamente un 0.05% del total, los eliminamos.

```
In [ ]: df = df.dropna(subset=["temp", "dwpt", "rhum", "wdir", "wspd", "pres"])
print(df.where(df.temp.isNull()).count())
print(df.count())
```

```
0
```

```
[Stage 116:=====] (6 +)
89973951
```

Por último, tratamos los nulls de prcp

```
In [ ]: nulls = df.where(df.prcp.isNull()).count()
print(nulls)
print(f"porcentaje del total que representan estos nulls: {nulls*100/df.c
12125
[Stage 122:=====] (5 +)
porcentaje del total que representan estos nulls: 0.013476122661324499
```

De nuevo, el porcentaje de nulls es insignificante, por lo que eliminamos estos registros

```
In [ ]: df = df.dropna(subset=["prcp"])
print(df.where(df.prcp.isNull()).count())
print(df.count())
0
[Stage 128:=====] (6 +)
89961826
```

Hemos terminado la limpieza de las variables numéricas, veamos ahora cómo quedan las estadísticas

```
In [ ]: df.describe(['id', 'intensidad', 'ocupacion', 'carga', 'vmed', 'periodo_i
[Stage 131:=====] (6 +)
+-----+-----+-----+-----+
|summary|      id|  intensidad|     ocupacion|
+-----+-----+-----+-----+
|  count|  89961826|    89961826|   89961826|
|  mean| 6358.025357422158|284.1029339780394|4.892596633154156|16.308691
| stddev|163155.65954771815|344.8126710539194|8.934827606247323|16.747439
|  min|          31|        0.0|         0|
|  max| 1140850688|    2028.0|        100|
+-----+-----+-----+-----+
```

Categóricas

Vemos las categorías de las categóricas

```
In [ ]: # tipo_elem
df.groupBy('tipo_elem').count().show()
[Stage 134:=====] (6 +)
+-----+-----+
|tipo_elem|  count|
+-----+-----+
|      URB|83070801|
|      M30| 6891025|
+-----+-----+
```

```
# error
```

```
In [ ]: df.groupBy('error').count().show()
```

```
[Stage 139:=====] (6 +)
+---+-----+
|error| count|
+---+-----+
| null| 347021|
|     N|89614805|
+---+-----+
```

```
In [ ]: # festivos
df.groupBy('tipo_de_festivo').count().show()
```

```
[Stage 144:=====] (6 +)
+-----+-----+
| tipo_de_festivo| count|
+-----+-----+
|         Nacional| 2321271|
|             null| 1535191|
|        No Festivo|85111901|
|           Local| 502542|
|Comunidad de Madrid| 490921|
+-----+-----+
```

```
In [ ]: # coco?
df.groupBy('coco').count().show()
```

```
[Stage 149:=====] (6 +)
+---+-----+
|coco| count|
+---+-----+
|    7| 660364|
|    3|16370051|
|    8| 852868|
|null|51472356|
|    5| 777443|
|   18| 68480|
|   17| 340179|
|    9| 513591|
|   11|14568486|
|    2| 4338008|
+---+-----+
```

Tratamiento de las categóricas

1. Recategorizamos los nulls de error como E/S (de esto en el informe ni mu xq es un peazo de falseo)

Recategorizamos asignando de forma aleatoria un 50/50 de los datos nulos para cada una de las dos categorías. Utilizamos random.randint(0, 1) para decidir entre "E" y "S" para los valores nulos. Esto asegura una recategorización aleatoria y equilibrada de los valores nulos de la columna "error". Los valores no nulos permanecerán sin cambios.

Versión 2: Recategorizamos todos los nulls como una nueva categoría llamada E/S, así no es tan falso y no afecta tanto a las predicciones, los nulls iban a ser seguramente o E o S pero no sabemos cuál era cada uno, para hacer esto de manera más acertada podríamos hacer clustering pero nos va a llevar la vida, así mejor metemos esta categoría nueva, decimos que los datos venían así y palante.

```
In [ ]: # Recategorizar los valores nulos como "E/S"
df = df.withColumn("error", when(col("error").isNull(), "E/S").otherwise(
```

Comprobamos que se haya creado la nueva categoria correctamente

```
In [ ]: df.groupBy('error').count().show()
```

```
+----+-----+
|error|  count |
+----+-----+
|    N|89614805|
|   E/S| 347021|
+----+-----+
```

1. Eliminamos la variable coco porque tiene más de la mitad de los registros a null

```
In [ ]: df = df.drop('coco')
```

1. Vemos qué pasa con los festivo, tipo_festivo nulos y fin_de_semana

3.1 Análisis de valores nulos

```
In [ ]: temporal = ['festivo', 'tipo_festivo', 'fin_de_semana']

for variable in temporal:
    nulos = df.filter(col("festivo").isNull()).count()
    print(f'{variable} contiene un total de {nulos} valores nulos')
```

```
festivo contiene un total de 1535191 valores nulos
```

```
tipo_festivo contiene un total de 1535191 valores nulos
```

```
[Stage 167:=====] (6 +)
fin_de_semana contiene un total de 1535191 valores nulos
```

Como vemos, tenemos el mismo número de valores nulos para las tres variables por lo que posiblemente se correspondan con los mismos registros.

Vamos a comprobar a qué años, meses y días se corresponden estos missings:

```
In [ ]: df.filter(col("festivo").isNull()).select('anno').distinct().show()
```

```
[Stage 170:=====] (6 +
+---+
|anno|
+---+
```

```
| 2023 |  
+---+
```

```
In [ ]: df.filter(col("festivo").isNull()).select('mes').distinct().show()
```

```
[Stage 173:=====] (6 +  
+---+  
| mes |  
+---+  
|   6 |  
+---+
```

```
In [ ]: df.filter(col("festivo").isNull()).select('dia').distinct().orderBy('dia')
```

```
[Stage 178:=====] (6 +  
+---+  
| dia |  
+---+  
|  16 |  
|  17 |  
|  18 |  
|  19 |  
|  20 |  
|  21 |  
|  22 |  
|  23 |  
|  24 |  
|  25 |  
|  26 |  
|  27 |  
|  28 |  
|  29 |  
|  30 |  
+---+
```

Como podemos ver, los missings se corresponden a la segunda quincena del mes de junio de 2023. Esto es debido a que se hizo un left join entre el dataset de tráfico, el cual tenía datos hasta el 30 de junio de 2023, y el de festividades, cuya fecha máxima era el 15 de junio de 2023.

Dado que en el mes de junio de 2023 no hubo festivos en Madrid, se puede actualizar dicha información de forma sencilla en nuestro dataframe (fines de semana los días 17,18,24 y 25):

```
In [ ]: # Actualizamos el campo festivo  
df = df.withColumn("festivo",  
    when(((col("anno") == 2023) & (col("mes") == 6) & (col("dia") >= 14))  
        "0").otherwise(col("festivo"))  
)
```

```
In [ ]: # Actualizamos el campo tipo_de_festivo  
df = df.withColumn("tipo_de_festivo",  
    when(((col("anno") == 2023) & (col("mes") == 6) & (col("dia") >= 14))
```

```
        "No Festivo") .otherwise(col("tipo_de_festivo"))
    )
```

```
In [ ]: # Actualizamos el campo fin_de_semana

fines_semana = [3,4,10,11,17,18,24,25]

# Fin de semana
df = df.withColumn("fin_de_semana",
    when(((col("anno") == 2023) & (col("mes") == 6) & (col("dia").isin(fines_semana)))
        .otherwise(col("fin_de_semana")))
)

# No fin de semana
df = df.withColumn("fin_de_semana",
    when(((col("anno") == 2023) & (col("mes") == 6) & (~col("dia").isin(fines_semana)))
        .otherwise(col("fin_de_semana")))
)
```

Volvemos a comprobar que ya no queden nulos:

```
In [ ]: temporal = ['festivo', 'tipo_festivo', 'fin_de_semana']

for variable in temporal:
    nulos = df.filter(col(variable).isNull()).count()
    print(f'{variable} contiene un total de {nulos} valores nulos')
```

```
[Stage 190:=====] (6 +)
festivo contiene un total de 0 valores nulos
```

```
tipo_festivo contiene un total de 0 valores nulos
```

```
[Stage 196:=====] (6 +)
fin_de_semana contiene un total de 0 valores nulos
```

```
In [ ]: df.write.parquet("gs://bucket_traffic_estimation/Datasets/total_limpio.parquet")
```

```
In [ ]: df.show()
```

```
[Stage 202:> (0 +)
+---+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+---+-----+-----+-----+-----+-----+
| 6493|    URB|      53.0|        0|     16|     0|          14|     N
| 6493|    URB|      36.0|        0|      8|     0|          10|     N
| 6493|    URB|      22.0|        0|      5|     0|          14|     N
| 6493|    URB|      17.0|        0|      4|     0|          15|     N
| 6493|    URB|       9.0|        0|      3|     0|          15|     N
| 6493|    URB|       1.0|        0|      0|     0|          15|     N
| 6493|    URB|       4.0|        0|      1|     0|          15|     N
| 6493|    URB|       1.0|        0|      1|     0|          15|     N
| 6493|    URB|       1.0|        0|      0|     0|          15|     N
| 6493|    URB|      16.0|        0|      4|     0|          9|     N
| 6493|    URB|      33.0|        0|      7|     0|          10|     N
| 6493|    URB|      70.0|        0|     21|     0|          14|     N
| 6493|    URB|      98.0|        1|     30|     0|          14|     N
| 6493|    URB|      53.0|        0|     16|     0|          13|     N
| 6493|    URB|      72.0|        0|     20|     0|          14|     N
| 6493|    URB|      60.0|        1|     17|     0|          13|     N
```

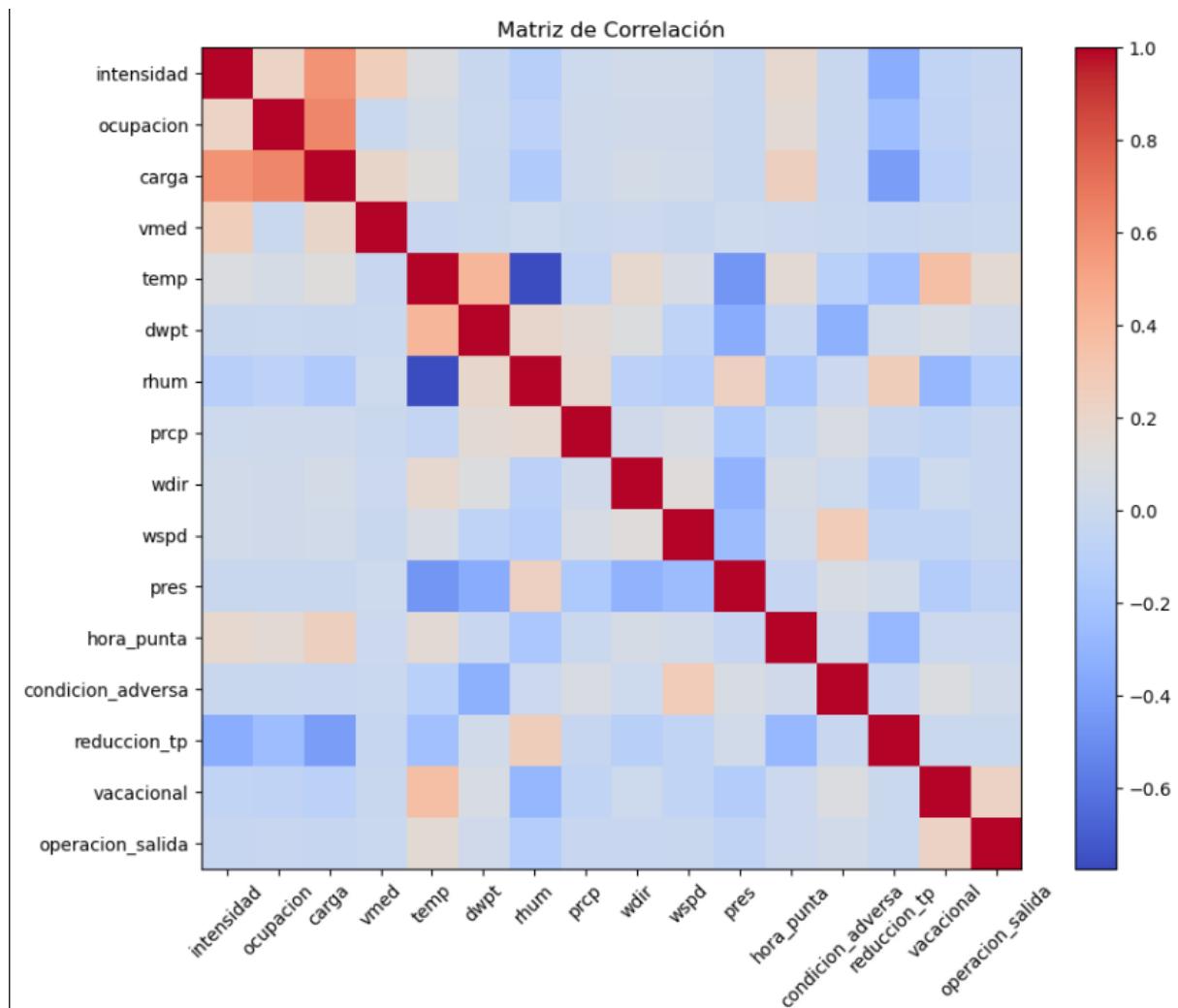
```
| 6493 |      URB |      60.0 |      0 |     18 |      0 |          14 |      N  
| 6493 |      URB |      94.0 |      0 |     30 |      0 |          15 |      N  
| 6493 |      URB |     107.0 |      0 |     34 |      0 |          15 |      N  
| 6493 |      URB |     105.0 |      1 |     35 |      0 |          15 |      N  
+---+---+---+---+---+---+---+---+  
only showing top 20 rows
```

```
In [ ]: df.count()
```

```
Out[ ]: 89961826
```

.2. Gráficos del EDA

.2.1. Gráfico de correlaciones



.3. Script de construcción de nuevas variables calculadas

```
In [ ]: from pyspark.sql.functions import *
import matplotlib.pyplot as plt
```

```
In [ ]: from pyspark.sql import SparkSession

# Crea una instancia de SparkSession
spark = SparkSession.builder \
    .appName("script3") \
    .getOrCreate()
```

```
In [ ]: df = spark.read.option("header", "true").parquet("gs://bucket_traffic_est")
df.show()
```

| id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion | error |
|-------|-----------|------------|-----------|-------|------|---------------------|-------|
| 39191 | URB | 787.0 | 5 | 25 | 0 | 15 | N |
| 39191 | URB | 686.0 | 3 | 22 | 0 | 15 | N |
| 39191 | URB | 485.0 | 1 | 14 | 0 | 14 | N |
| 39191 | URB | 572.0 | 2 | 17 | 0 | 15 | N |
| 39191 | URB | 728.0 | 3 | 22 | 0 | 15 | N |
| 39191 | URB | 533.0 | 2 | 17 | 0 | 15 | N |
| 39191 | URB | 540.0 | 2 | 16 | 0 | 15 | N |
| 39191 | URB | 432.0 | 2 | 13 | 0 | 15 | N |
| 39191 | URB | 295.0 | 1 | 9 | 0 | 15 | N |
| 39191 | URB | 187.0 | 0 | 4 | 0 | 15 | N |
| 39191 | URB | 142.0 | 0 | 4 | 0 | 15 | N |
| 39191 | URB | 76.0 | 0 | 2 | 0 | 15 | N |
| 39191 | URB | 26.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 28.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 22.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 41.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 127.0 | 0 | 2 | 0 | 14 | N |
| 39191 | URB | 271.0 | 0 | 6 | 0 | 15 | N |
| 39191 | URB | 439.0 | 2 | 13 | 0 | 15 | N |
| 39191 | URB | 571.0 | 2 | 16 | 0 | 15 | N |

only showing top 20 rows

Creación de variables

1. Creamos la variable estaciones del año.

Se utilizan las variables día y mes para poder extraer la estación del año que es.

```
In [ ]: df = df.withColumn('estacion',
    when(
        ((col('mes') == 3) & (col('dia') >= 21)) | ((col('mes') >= 4) & (
            'Primavera'
        ))
        .when(
            ((col('mes') == 6) & (col('dia') >= 21)) | ((col('mes') >= 7) & (
                'Verano'
            ))
        )
        .when(
            ((col('mes') == 9) & (col('dia') >= 21)) | ((col('mes') >= 10) &
```

```

        'Otono'
    )
.otherwise('Inviero')
)

```

Realizamos la comprobación para ver que las estaciones están bien, tomando un año aleatorio.

```
In [ ]: anho = 2022
filtro = df.filter(col('anno') == anho)
comprobacion = filtro.groupBy('estacion', 'mes').agg(collect_set('dia')).a
comprobacion.show(truncate=False)
```

```
[Stage 8:=====] (2 +)
+-----+
|estacion |mes|dia
+-----+
|Primavera|5 |[30, 9, 1, 31, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14
|Verano   |7 |[30, 9, 1, 31, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14
|Inviero  |1 |[30, 9, 1, 31, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14
|Otono    |10|[30, 9, 1, 31, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14
|Verano   |8 |[30, 9, 1, 31, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14
|Otono    |12|[15, 9, 1, 16, 2, 17, 3, 18, 10, 4, 11, 12, 19, 13, 5, 20,
|Otono    |9 |[30, 27, 24, 21, 28, 25, 22, 29, 26, 23]
|Inviero  |3 |[15, 9, 1, 16, 2, 17, 3, 18, 10, 4, 11, 12, 19, 13, 5, 20,
|Verano   |9 |[15, 9, 1, 16, 2, 17, 3, 18, 10, 4, 11, 12, 19, 13, 5, 20,
|Otono    |11|[30, 9, 1, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14, 23
|Verano   |6 |[30, 27, 24, 21, 28, 25, 22, 29, 26, 23]
|Primavera|6 |[15, 9, 1, 16, 2, 17, 3, 18, 10, 4, 11, 12, 19, 13, 5, 20,
|Primavera|4 |[30, 9, 1, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14, 23
|Inviero  |12|[30, 27, 31, 24, 21, 28, 25, 22, 29, 26, 23]
|Inviero  |2 |[9, 1, 2, 24, 3, 25, 4, 26, 27, 19, 20, 21, 22, 14, 23, 15
|Primavera|3 |[30, 27, 31, 24, 21, 28, 25, 22, 29, 26, 23]
+-----+
```

2. Creamos la variable días de la semana.

```
In [ ]: df = df.withColumn('dia_semana',
date_format(concat_ws('-', col('anno')).cast('string'), col('mes')).cas
```

Realizamos la comprobación para ver que los días están bien, tomando un mes y un año aleatorios.

```
In [ ]: anho = 2023
mes = 5
filtro = df.filter((col('anno') == anho) & (col('mes') == mes))

comprobacion = filtro.groupBy('dia_semana').agg(collect_set('dia').alias(
comprobacion.show(truncate=False))
```

```
+-----+
|dia_semana|dia
+-----+
|Wednesday |[31, 17, 24, 3, 10]|
|Tuesday   |[30, 9, 16, 2, 23] |
|Friday    |[12, 19, 5, 26] |
|Thursday  |[18, 25, 4, 11] |
```

```

| Saturday | [27, 13, 20, 6]    |
| Monday   | [15, 1, 22, 29, 8] |
| Sunday   | [21, 28, 7, 14]    |
+-----+-----+

```

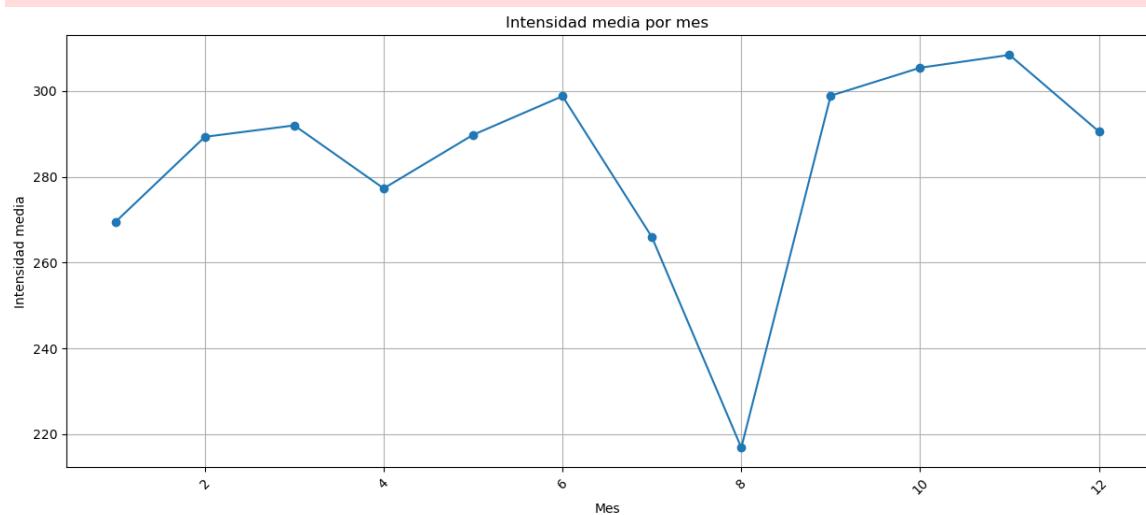
3. Creamos la variable hora punta.

Para crear esta variable vamos a tener en cuenta varios factores. Se establece primero que la hora punta sólo se aplicará para días laborables, es decir, de lunes a viernes. También se asume que la hora punta cambiará en los meses de verano puesto que muchas empresas tienen jornada intensiva.

Empezamos estudiando qué ocurre con las intensidades medias por meses, para ver si existe algún mes donde la intensidad media se reduzca.

```
In [ ]: intensidad_diarria = df.groupBy("mes").agg(avg("intensidad")).alias("avg_intensity")
intensidad_diarria_pandas = intensidad_diarria.toPandas()

plt.figure(figsize=(15, 6))
plt.plot(intensidad_diarria_pandas["mes"], intensidad_diarria_pandas["avg_intensity"])
plt.xlabel("Mes")
plt.ylabel("Intensidad media")
plt.title("Intensidad media por mes")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



Se ve claramente que hay una diferencia grande entre los meses de verano y el resto del año. Además, se observa una diferencia también entre julio y agosto. Así que, haremos la siguiente división en relación a los meses: Sept - Jun, Julio y Agosto.

Estudiamos las horas del día donde hay más tráfico para los meses de Sept - Jun en días laborables.

```
In [ ]: meses_filtros = [9, 10, 11, 12, 1, 2, 3, 4, 5, 6]
curso_laboral = df.filter((col('mes').isin(meses_filtros)) & (df.fin_de_semana == False))
intensidad_diarria = curso_laboral.groupby("hora").agg(avg("intensidad")).alias("avg_intensity")
```

```

intensidad_diaria_pandas = intensidad_diaria.toPandas()

plt.figure(figsize=(15, 6))
plt.plot(intensidad_diaria_pandas["hora"], intensidad_diaria_pandas["avg_intensidad"])
plt.xlabel("Hora")
plt.ylabel("Intensidad media")
plt.title("Intensidad media en los meses de septiembre a junio en días laborables")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

```



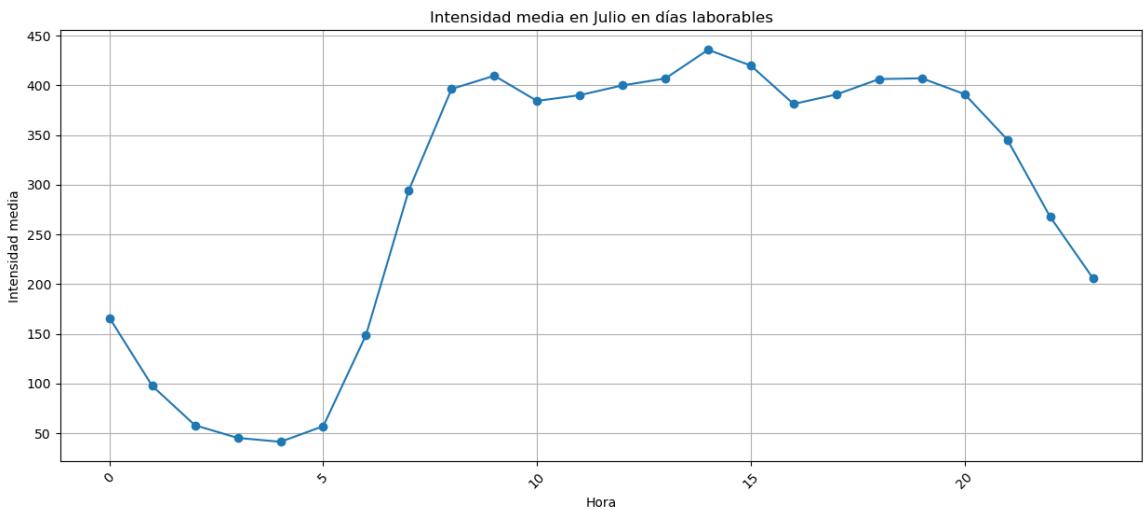
Estudiamos ahora las horas del día donde hay más tráfico para el mes de Julio y luego para el mes de Agosto, en días laborables.

```

In [ ]: julio_semana = df.filter((df.mes == 7) & (df.fin_de_semana == 0))
intensidad_diaria = julio_semana.groupBy("hora").agg(avg("intensidad")).alias("avg_intensidad")
intensidad_diaria_pandas = intensidad_diaria.toPandas()

plt.figure(figsize=(15, 6))
plt.plot(intensidad_diaria_pandas["hora"], intensidad_diaria_pandas["avg_intensidad"])
plt.xlabel("Hora")
plt.ylabel("Intensidad media")
plt.title("Intensidad media en Julio en días laborables")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

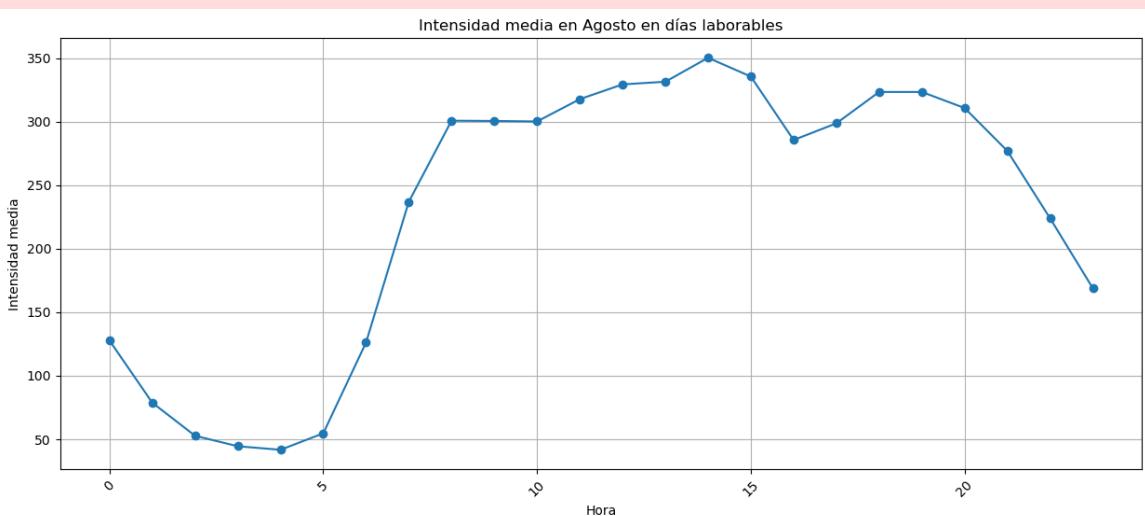
```



```
In [ ]: agosto_laboral = df.filter((df.mes == 8) & (df.fin_de_semana == 0))
intensidad_diaria = agosto_laboral.groupBy("hora").agg(avg("intensidad"))

intensidad_diaria_pandas = intensidad_diaria.toPandas()

plt.figure(figsize=(15, 6))
plt.plot(intensidad_diaria_pandas["hora"], intensidad_diaria_pandas["avg"])
plt.xlabel("Hora")
plt.ylabel("Intensidad media")
plt.title("Intensidad media en Agosto en días laborables")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



Estudiamos ahora qué ocurre con las intensidades medias por horas los fines de semana, con la misma división mensual que antes.

```
In [ ]: meses_filtros = [9, 10, 11, 12, 1, 2, 3, 4, 5, 6]
curso_fin = df.filter((col('mes')).isin(meses_filtros)) & (df.fin_de_seman
intensidad_diaria = curso_fin.groupBy("hora").agg(avg("intensidad")).alias

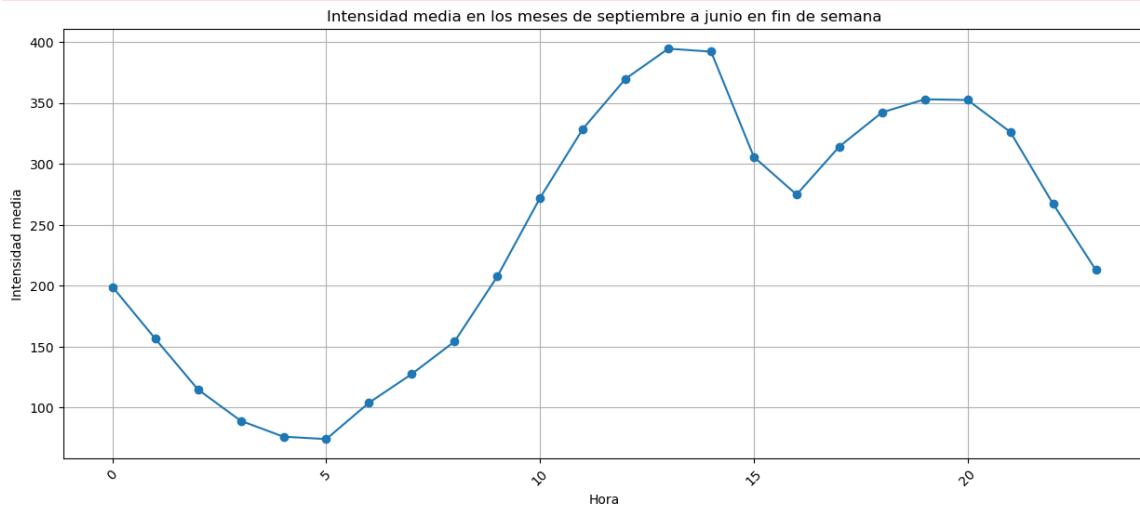
intensidad_diaria_pandas = intensidad_diaria.toPandas()

plt.figure(figsize=(15, 6))
plt.plot(intensidad_diaria_pandas["hora"], intensidad_diaria_pandas["avg"])
plt.xlabel("Hora")
plt.ylabel("Intensidad media")
```

```

plt.title("Intensidad media en los meses de septiembre a junio en fin de semana")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

```

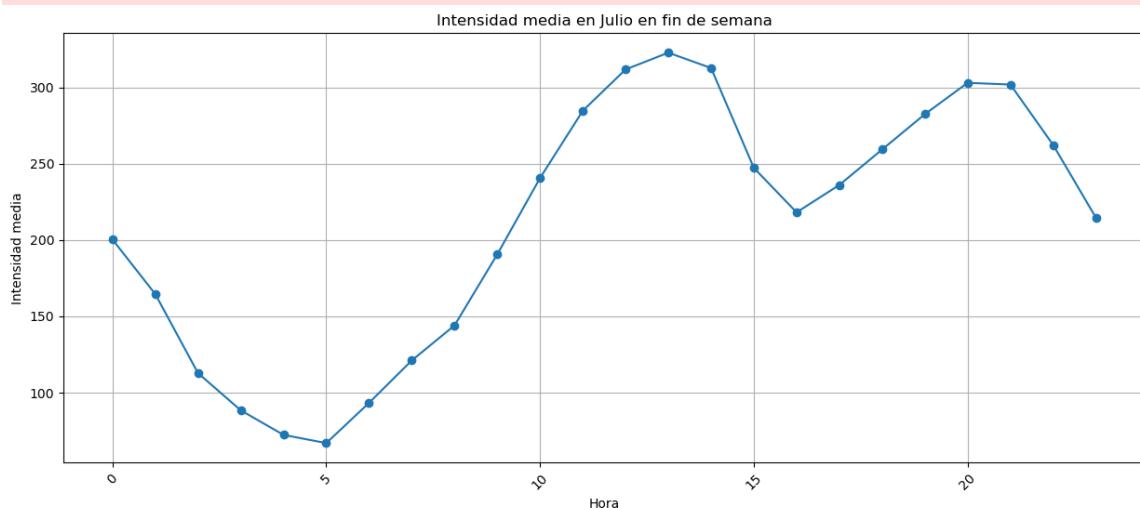


```

In [ ]: julio_fin = df.filter((df.mes == 7) & (df.fin_de_semana == 1))
intensidad_diaria = julio_fin.groupBy("hora").agg(avg("intensidad")).alias("intensidad_diaria")
intensidad_diaria_pandas = intensidad_diaria.toPandas()

plt.figure(figsize=(15, 6))
plt.plot(intensidad_diaria_pandas["hora"], intensidad_diaria_pandas["avg"])
plt.xlabel("Hora")
plt.ylabel("Intensidad media")
plt.title("Intensidad media en Julio en fin de semana")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

```



```

In [ ]: agosto_fin = df.filter((df.mes == 8) & (df.fin_de_semana == 1))
intensidad_diaria = agosto_fin.groupBy("hora").agg(avg("intensidad")).alias("intensidad_diaria")
intensidad_diaria_pandas = intensidad_diaria.toPandas()

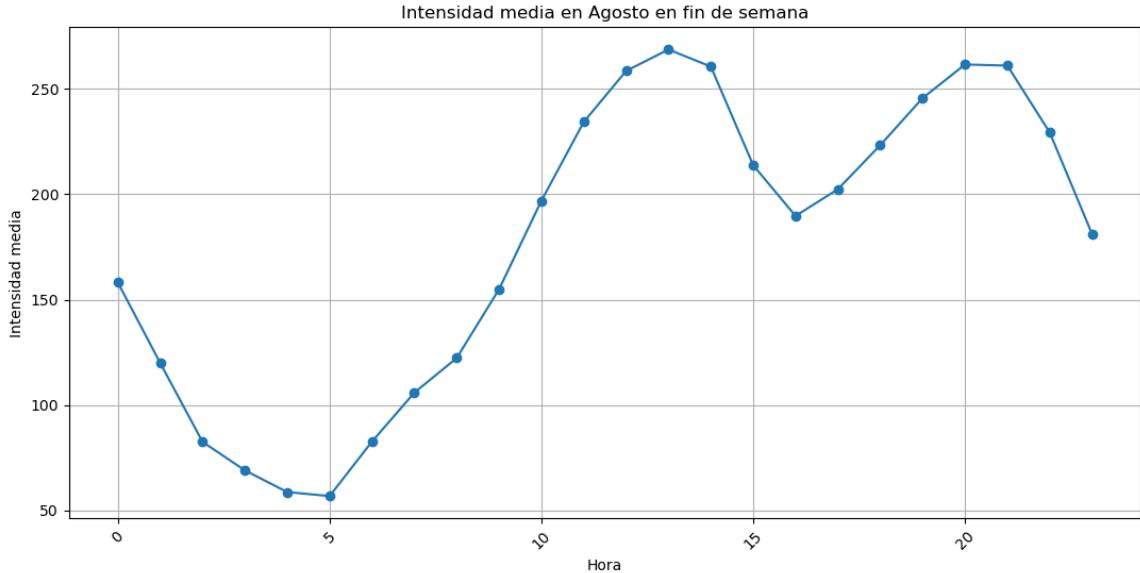
plt.figure(figsize=(13, 6))
plt.plot(intensidad_diaria_pandas["hora"], intensidad_diaria_pandas["avg"])
plt.xlabel("Hora")

```

```

plt.ylabel("Intensidad media")
plt.title("Intensidad media en Agosto en fin de semana")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()

```



Por tanto, observando los gráficos anteriores, se puede deducir que las horas puntuas para los días laborables son:

- Sept - Jul: 9:00, 14:00-15:00, 19:00
- Agosto: 12:00-15:00, 19:00

Por su parte, las horas puntuas los fines de semana son:

- Sept - Jul: 12:00-14:00, 19:00-21:00
- Agosto: 12:00-14:00, 20:00-21:00

Creamos entonces la nueva variable de hora punta en función a las horas puntuas que hemos observado para cada división mensual.

```

In [ ]: # Agosto
condiciones_hora_punta_verano = [
    (
        (col('dia_semana')).isin(['Monday', 'Tuesday', 'Wednesday', 'Thursday'])
        & ((col('hora')).between(12, 15)) | (col('hora')).between(19, 19))
    ) |
    (
        (col('dia_semana')).isin(['Saturday', 'Sunday'])) &
        ((col('hora')).between(12, 14)) | (col('hora')).between(20, 21))
    ),
]

# Resto de meses
condiciones_hora_punta_noverano = [
    (
        (col('dia_semana')).isin(['Monday', 'Tuesday', 'Wednesday', 'Thursday'])
        & ((col('hora')).between(9, 9)) | (col('hora')).between(14, 15)) | (col('hora')).between(20, 21))
    ) |
    (
        (col('dia_semana')).isin(['Saturday', 'Sunday'])) &
        ((col('hora')).between(12, 14)) | (col('hora')).between(19, 19))
    )
]

```

```

        ((col('hora').between(12, 14)) | (col('hora').between(19, 21)))
    ),
]

df = df.withColumn('hora_punta',
when(((col('mes') == 8) & (condiciones_hora_punta_verano[0])) | ((col
.otherwise(0)
)
)

```

In []: df.show()

| [Stage 72:> (0 + |
|--|
| id tipo_elem intensidad ocupacion carga vmed periodo_integracion error |
| -----+-----+-----+-----+-----+-----+-----+-----+ |
| 3919 URB 787.0 5 25 0 15 N |
| 3919 URB 686.0 3 22 0 15 N |
| 3919 URB 485.0 1 14 0 14 N |
| 3919 URB 572.0 2 17 0 15 N |
| 3919 URB 728.0 3 22 0 15 N |
| 3919 URB 533.0 2 17 0 15 N |
| 3919 URB 540.0 2 16 0 15 N |
| 3919 URB 432.0 2 13 0 15 N |
| 3919 URB 295.0 1 9 0 15 N |
| 3919 URB 187.0 0 4 0 15 N |
| 3919 URB 142.0 0 4 0 15 N |
| 3919 URB 76.0 0 2 0 15 N |
| 3919 URB 26.0 0 0 0 15 N |
| 3919 URB 28.0 0 0 0 15 N |
| 3919 URB 22.0 0 0 0 15 N |
| 3919 URB 41.0 0 0 0 15 N |
| 3919 URB 127.0 0 2 0 14 N |
| 3919 URB 271.0 0 6 0 15 N |
| 3919 URB 439.0 2 13 0 15 N |
| 3919 URB 571.0 2 16 0 15 N |
| -----+-----+-----+-----+-----+-----+-----+-----+ |
| only showing top 20 rows |

4. Creación de la variable condiciones adversas.

In []: print(df.agg({"temp": "max"}).collect()[0][0])
print(df.agg({"temp": "min"}).collect()[0][0])

41.5

[Stage 76:=====> (3 +
-10.2

In []: print(df.agg({"prcp": "max"}).collect()[0][0])
print(df.agg({"prcp": "min"}).collect()[0][0])

7.5

[Stage 82:=====> (6 +
0.0

In []: print(df.agg({"wspd": "max"}).collect()[0][0])

```
print(df.agg({"wspd": "min"}).collect()[0][0])
```

```
41.4
```

```
[Stage 88:=====] (6 + 0.0)
```

```
In [ ]: low_temp_condition = (col('temp').between(-10.2, 5))
high_temp_condition = (col('temp').between(30, 41.5))
moderate_precip_condition = (col('prcp').between(2.1, 5))
high_precip_condition = (col('prcp') > 5)
high_wspd_condition = (col('wspd') > 20)

df = df.withColumn('condicion_adversa',
    when((low_temp_condition) | (high_temp_condition) | (moderate_precip_
        .otherwise(0)
    )

from IPython.core.display import HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))
df.show()
```

```
23/09/01 18:19:06 WARN org.apache.spark.sql.catalyst.util.package: Truncating data at Stage 91: (0 +
```

| id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion | error |
|-------|-----------|------------|-----------|-------|------|---------------------|-------|
| 39191 | URB | 787.0 | 5 | 25 | 0 | 15 | N |
| 39191 | URB | 686.0 | 3 | 22 | 0 | 15 | N |
| 39191 | URB | 485.0 | 1 | 14 | 0 | 14 | N |
| 39191 | URB | 572.0 | 2 | 17 | 0 | 15 | N |
| 39191 | URB | 728.0 | 3 | 22 | 0 | 15 | N |
| 39191 | URB | 533.0 | 2 | 17 | 0 | 15 | N |
| 39191 | URB | 540.0 | 2 | 16 | 0 | 15 | N |
| 39191 | URB | 432.0 | 2 | 13 | 0 | 15 | N |
| 39191 | URB | 295.0 | 1 | 9 | 0 | 15 | N |
| 39191 | URB | 187.0 | 0 | 4 | 0 | 15 | N |
| 39191 | URB | 142.0 | 0 | 4 | 0 | 15 | N |
| 39191 | URB | 76.0 | 0 | 2 | 0 | 15 | N |
| 39191 | URB | 26.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 28.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 22.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 41.0 | 0 | 0 | 0 | 15 | N |
| 39191 | URB | 127.0 | 0 | 2 | 0 | 14 | N |
| 39191 | URB | 271.0 | 0 | 6 | 0 | 15 | N |
| 39191 | URB | 439.0 | 2 | 13 | 0 | 15 | N |
| 39191 | URB | 571.0 | 2 | 16 | 0 | 15 | N |

```
only showing top 20 rows
```

5. Creación de la variable reducción de transporte público

Esta variable es una binaria que representa si se está en horario de transporte público reducido o no (1:30-6:00, tomaremos 1:00-6:00)

```
In [ ]: df = df.withColumn('reduccion_tp',
    when(col("hora").between(1, 6), 1)
```

```

    .otherwise(0)
)

display(HTML("<style>pre { white-space: pre !important; }</style>"))
df.show()

```

| [Stage 92:> (0 +) | | | | | | | | |
|--|----|-----------|------------|-----------|-------|------|---------------------|-------|
| | id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion | error |
| 3919 URB 787.0 5 25 0 15 N | | | | | | | | |
| 3919 URB 686.0 3 22 0 15 N | | | | | | | | |
| 3919 URB 485.0 1 14 0 14 N | | | | | | | | |
| 3919 URB 572.0 2 17 0 15 N | | | | | | | | |
| 3919 URB 728.0 3 22 0 15 N | | | | | | | | |
| 3919 URB 533.0 2 17 0 15 N | | | | | | | | |
| 3919 URB 540.0 2 16 0 15 N | | | | | | | | |
| 3919 URB 432.0 2 13 0 15 N | | | | | | | | |
| 3919 URB 295.0 1 9 0 15 N | | | | | | | | |
| 3919 URB 187.0 0 4 0 15 N | | | | | | | | |
| 3919 URB 142.0 0 4 0 15 N | | | | | | | | |
| 3919 URB 76.0 0 2 0 15 N | | | | | | | | |
| 3919 URB 26.0 0 0 0 15 N | | | | | | | | |
| 3919 URB 28.0 0 0 0 15 N | | | | | | | | |
| 3919 URB 22.0 0 0 0 15 N | | | | | | | | |
| 3919 URB 41.0 0 0 0 15 N | | | | | | | | |
| 3919 URB 127.0 0 2 0 14 N | | | | | | | | |
| 3919 URB 271.0 0 6 0 15 N | | | | | | | | |
| 3919 URB 439.0 2 13 0 15 N | | | | | | | | |
| 3919 URB 571.0 2 16 0 15 N | | | | | | | | |

only showing top 20 rows

6. Creación de la variable puente

Consideramos puente cuando hay más de dos días festivos/findesemana seguidos

```

In [ ]: import pyspark.sql.functions as F
from pyspark.sql.window import Window

# Creo un df auxiliar en el que no hay horas para buscar los puentes
agg_df = df.groupBy(["anno", "mes", "dia"]).agg(
    F.max('festivo').alias('festivo'),
    F.max('fin_de_semana').alias('fin_de_semana')
)

# Calcular la columna auxiliar "finde_festivo"
agg_df = agg_df.withColumn("finde_festivo", (F.col("festivo") + F.col("fi
window_spec = Window.orderBy("anno", "mes", "dia")

# Usamos lag() y lead() para obtener el valor de "festivo" en los registr
agg_df = agg_df.withColumn("festivo_anterior_1", lag("finde_festivo").ove
agg_df = agg_df.withColumn("festivo_anterior_2", lag("finde_festivo", 2).o
agg_df = agg_df.withColumn("festivo_siguiente_1", lead("finde_festivo").o
agg_df = agg_df.withColumn("festivo_siguiente_2", lead("finde_festivo", 2

# Creamos la columna "puente"
agg_df = agg_df.withColumn("puente", when(

```

```

        (agg_df["finde_festivo"] == 1) & (
        ((agg_df["festivo_anterior_1"] == 1) & (agg_df["festivo_anterior_2"])
        ((agg_df["festivo_siguiente_1"] == 1) & (agg_df["festivo_siguiente_2"])
        ((agg_df["festivo_anterior_1"] == 1) & (agg_df["festivo_siguiente_1"])
        ),
        1
    ) .otherwise(0))

# Eliminamos las columnas auxiliares que ya no necesitamos
agg_df = agg_df.drop("festivo_anterior_1", "festivo_anterior_2", "festivo

# agg_df.show()

# Hago un left join con el df
agg_df = agg_df.withColumnRenamed("anno", "anno_agg")
agg_df = agg_df.withColumnRenamed("mes", "mes_agg")
agg_df = agg_df.withColumnRenamed("dia", "dia_agg")

join_condition = (df["mes"] == agg_df["mes_agg"]) & (df["dia"] == agg_df[
df = df.join(agg_df, join_condition, "left")
df = df.drop("anno_agg", "mes_agg", "dia_agg")

display(HTML("<style>pre { white-space: pre !important; }</style>"))
df.show()

```

```

23/09/01 18:19:09 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:21:33 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:21:48 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:21:48 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:21:48 WARN org.apache.spark.sql.execution.window.WindowExec:

+-----+-----+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+-----+-----+-----+-----+-----+-----+-----+
| 3920|    URB|      65.0|       1|     7|     0|        15|     N
| 3920|    URB|      31.0|       0|     2|     0|        15|     N
| 3920|    URB|      14.0|       0|     1|     0|        15|     N
| 3920|    URB|      13.0|       0|     1|     0|        15|     N
| 3920|    URB|       3.0|       0|     0|     0|        14|     N
| 3920|    URB|      20.0|       0|     1|     0|        15|     N
| 3920|    URB|      43.0|       0|     3|     0|        13|     N
| 3920|    URB|     143.0|       2|    12|     0|        14|     N
| 3920|    URB|     226.0|       4|    26|     0|        15|     N
| 3920|    URB|     259.0|       5|    28|     0|        15|     N
| 3920|    URB|     252.0|       5|    28|     0|        15|     N
| 3920|    URB|     261.0|       6|    29|     0|        15|     N
| 3920|    URB|     267.0|       6|    30|     0|        15|     N
| 3920|    URB|     283.0|       5|    30|     0|        15|     N
| 3920|    URB|     297.0|       5|    34|     0|        15|     N
| 3920|    URB|     223.0|       3|    24|     0|        15|     N
| 3920|    URB|     198.0|       3|    23|     0|        15|     N
| 3920|    URB|     213.0|       4|    23|     0|        15|     N
| 3920|    URB|     224.0|       3|    25|     0|        15|     N
| 3920|    URB|     219.0|       4|    24|     0|        15|     N
+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

```

In []: # comprobación

```

display(HTML("<style>pre { white-space: pre !important; }</style>"))
df.where((col("puente") == 1) & (col("hora") == 0)).show()

```

```

23/09/01 18:21:50 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:22:11 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:22:34 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:22:35 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:22:35 WARN org.apache.spark.sql.execution.window.WindowExec:

+-----+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+-----+-----+-----+-----+-----+-----+
| 4164|    URB|    743.0|      2|     15|     0|          15|     N
| 4165|    URB|    182.0|      1|      9|     0|          13|     N
| 4166|    URB|    556.0|      5|     25|     0|          15|     N
| 4167|    URB|    332.0|      5|     22|     0|          15|     N
| 4168|    URB|    210.0|      3|     25|     0|          15|     N
| 4169|    URB|    274.0|      4|     23|     0|          15|     N
| 4170|    URB|    318.0|      4|     15|     0|          15|     N
| 4172|    URB|     89.0|      2|     10|     0|          15|     N
| 4173|    URB|    910.0|      3|     21|     0|          15|     N
| 4174|    URB|    732.0|      5|     27|     0|          15|     N
| 4175|    URB|    558.0|      4|     31|     0|          15|     N
| 4176|    URB|     41.0|      8|     10|     0|          11|     N
| 4177|    URB|     32.0|      1|      5|     0|          14|     N
| 4178|    URB|     78.0|      1|      9|     0|          15|     N
| 4179|    URB|    187.0|      1|     12|     0|          12|     N
| 4180|    URB|    444.0|      1|     11|     0|          14|     N
| 4181|    URB|    170.0|      1|      6|     0|          14|     N
| 4182|    URB|    442.0|      6|     19|     0|          15|     N
| 4183|    URB|    152.0|      2|     10|     0|          15|     N
| 4184|    URB|    579.0|      1|     12|     0|          15|     N
+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

```

```
In [ ]: # Comprobamos si merece la pena quitar semana santa o dejarlo como puente
from pyspark.sql.functions import *
display(HTML("<style>pre { white-space: pre !important; }</style>"))
dfpuentes = df.where((col('puente') == 1) & (col('anno') == 2022)) \
    .groupBy('anno', 'mes', 'dia') \
    .agg(avg("intensidad").alias("avg_intensity")) \
    .orderBy('mes', 'dia')

dfpuentes.show()
```

```

23/09/01 18:22:36 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:22:43 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:23:03 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:23:03 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:23:03 WARN org.apache.spark.sql.execution.window.WindowExec:
[Stage 119:=====] (6 + 1)
+-----+
|anno|mes|dial|      avg_intensity|
+-----+
|2022| 4| 14| 167.3275679108101|
|2022| 4| 15|155.78392785728425|
|2022| 4| 16|177.18839692194828|
|2022| 4| 17|183.84634920564727|
|2022| 4| 30|239.64466845932017|
|2022| 5|  1|216.12673237333533|
|2022| 5|  2|195.6391066091033|
|2022| 5| 14|239.0432578020917|
|2022| 5| 15|213.55149765195054|
|2022| 5| 16|204.52475098976575|
|2022| 7| 23|188.83416692549187|

```

```
|2022| 7| 24| 163.1728518987654|
|2022| 7| 25|170.76722444928288|
|2022| 8| 13|155.94490527229905|
|2022| 8| 14| 141.2921325185329|
|2022| 8| 15|143.99485899769135|
|2022| 12| 24| 241.9599076755512|
|2022| 12| 25|211.12465791174844|
|2022| 12| 26|212.91687343563117|
+---+---+-----+
```

Vemos que no merece la pena, así que dejamos la variable puente como estaba

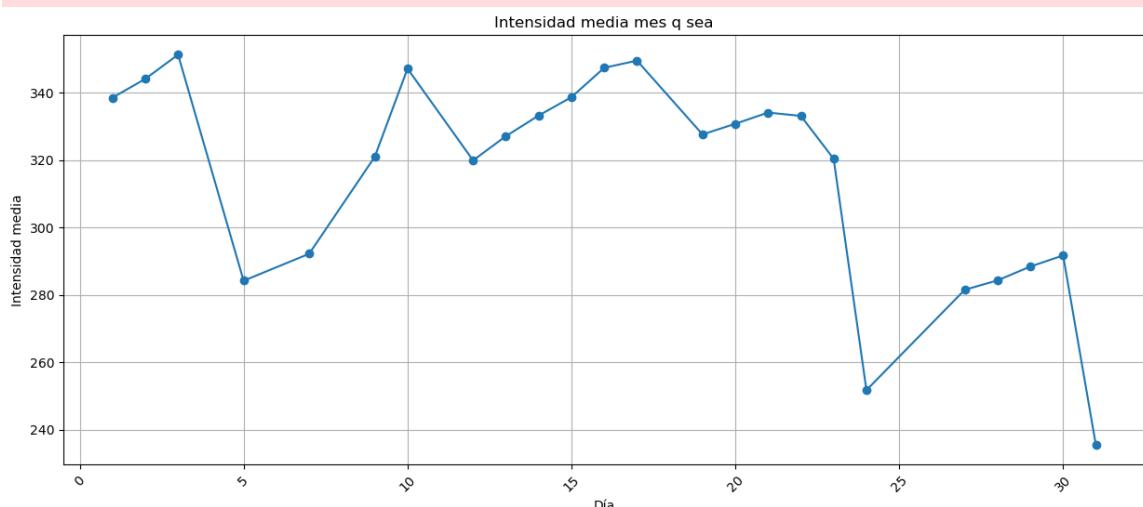
7. Creación de la variable periodo vacacional

seleccionamos los rangos de los periodos vacacionales. Se considerarán semana santa, verano (15-31 de julio y agosto) porque baja la intensidad media considerablemente a partir del 15 (se podría incluir julio entero la vdd), y navidades

```
In [ ]: intensidad_mensual = df.filter(
    (col('mes') == 12) &
    (col('fin_de_semana') == 0) &
    (col('festivo') == 0)
).groupBy("dia").agg(avg("intensidad").alias("avg_intensity")).orderBy("d

intensidad_mensual_pandas = intensidad_mensual.toPandas()

plt.figure(figsize=(15, 6))
plt.plot(intensidad_mensual_pandas["dia"], intensidad_mensual_pandas["avg
plt.xlabel("Día")
plt.ylabel("Intensidad media")
plt.title("Intensidad media mes q sea")
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



Crear col semanas santa

```
In [ ]: from pyspark.sql.types import IntegerType
from datetime import date, timedelta
from dateutil.relativedelta import relativedelta
```

```

from dateutil.easter import easter

def is_holiday(anno, mes, dia):
    easter_date = easter(anno)
    christmas_date = date(anno, 12, 25)
    kings_day_date = date(anno, 1, 6)
    summer_start = date(anno, 7, 1) # July 1
    summer_end = date(anno, 8, 31) # August 31

    # Calculate the start and end dates of the week before Easter
    week_before_easter_start = easter_date - timedelta(days=7)
    week_before_easter_end = easter_date - timedelta(days=1)

    input_date = date(anno, mes, dia)

    # Check if the date is a holiday
    return int(
        input_date == easter_date or
        (christmas_date - relativedelta(days=3)) <= input_date <= date(an-
        (date(anno, 1, 1) <= input_date <= kings_day_date) or
        (week_before_easter_start <= input_date <= week_before_easter_end
        (summer_start <= input_date <= summer_end)
    )

# Create a UDF for the is_holiday function
is_holiday_udf = udf(is_holiday, IntegerType())

# Add a new column 'is_holiday' to the DataFrame
df = df.withColumn('vacacional', is_holiday_udf(col('anno'), col('mes')))

df.show()

```

```

23/09/01 18:23:14 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:25:37 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:25:39 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:25:39 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:25:39 WARN org.apache.spark.sql.execution.window.WindowExec:
[Stage 144:> (0 +
+-----+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+-----+-----+-----+-----+-----+-----+
| 3920|    URB|     65.0|      1|     7|     0|          15|     N
| 3920|    URB|     31.0|      0|     2|     0|          15|     N
| 3920|    URB|     14.0|      0|     1|     0|          15|     N
| 3920|    URB|     13.0|      0|     1|     0|          15|     N
| 3920|    URB|      3.0|      0|     0|     0|          14|     N
| 3920|    URB|     20.0|      0|     1|     0|          15|     N
| 3920|    URB|     43.0|      0|     3|     0|          13|     N
| 3920|    URB|    143.0|      2|    12|     0|          14|     N
| 3920|    URB|    226.0|      4|    26|     0|          15|     N
| 3920|    URB|    259.0|      5|    28|     0|          15|     N
| 3920|    URB|    252.0|      5|    28|     0|          15|     N
| 3920|    URB|    261.0|      6|    29|     0|          15|     N
| 3920|    URB|    267.0|      6|    30|     0|          15|     N
| 3920|    URB|    283.0|      5|    30|     0|          15|     N
| 3920|    URB|    297.0|      5|    34|     0|          15|     N
| 3920|    URB|    223.0|      3|    24|     0|          15|     N
| 3920|    URB|    198.0|      3|    23|     0|          15|     N
| 3920|    URB|    213.0|      4|    23|     0|          15|     N
| 3920|    URB|    224.0|      3|    25|     0|          15|     N
| 3920|    URB|    219.0|      4|    24|     0|          15|     N
+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

```

```
In [ ]: filtered_df = df.filter((col('vacacional') == 1) & (col('anno') == 2021))

filtered_df.show()
```

```
23/09/01 18:25:43 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:25:52 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:12 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:12 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:12 WARN org.apache.spark.sql.execution.window.WindowExec:
[Stage 155:=====] (2 +)
+---+-----+-----+-----+-----+-----+
| id|tipo_elem| intensidad|ocupacion|carga|vmed|periodo_integraci
+---+-----+-----+-----+-----+-----+
|1001| M30 | 210.0| 1| 0| 48|
|1001| M30 | 90.0| 0| 0| 47|
|1001| M30 | 168.0| 1| 0| 66|
|1001| M30 | 99.0| 0| 0| 52|
|1001| M30 | 420.0| 1| 0| 59|
|1001| M30 | 450.0| 1| 0| 55|
|1001| M30 | 627.0| 1| 0| 62|
|1001| M30 | 789.0| 2| 0| 60|
|1001| M30 | 1110.0| 3| 0| 61|
|1001| M30 | 1497.0| 4| 0| 60|
|1001| M30 | 1743.0| 5| 0| 60|
|1001| M30 | 283.95715588406466| 8| 0| 55|
|1001| M30 | 283.95715588406466| 8| 0| 59|
|1001| M30 | 283.95715588406466| 11| 0| 58|
|1001| M30 | 283.95715588406466| 10| 0| 58|
|1001| M30 | 1965.0| 5| 0| 57|
+---+-----+-----+-----+-----+-----+
only showing top 20 rows
```

```
In [ ]: filtered_df = df.filter((col('vacacional') == 1) & (col('anno') == 2022))

filtered_df.show()
```

```
23/09/01 18:26:14 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:50 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:50 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:50 WARN org.apache.spark.sql.execution.window.WindowExec:
[Stage 166:=====] (3 +)
+---+-----+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+---+-----+-----+-----+-----+-----+-----+
|6273| URB | 3.0| 1| 2| 0| 15| N
|6273| URB | 1.0| 1| 2| 0| 15| N
|6273| URB | 1.0| 0| 0| 0| 15| N
|6273| URB | 0.0| 0| 0| 0| 15| N
|6273| URB | 1.0| 0| 0| 0| 15| N
|6273| URB | 0.0| 0| 0| 0| 15| N
|6273| URB | 16.0| 4| 7| 0| 6| N
|6273| URB | 20.0| 0| 6| 0| 3| N
|6273| URB | 22.0| 14| 11| 0| 6| N
|6273| URB | 25.0| 9| 10| 0| 7| N
|6273| URB | 30.0| 11| 13| 0| 8| N
+---+-----+-----+-----+-----+-----+-----+
```

```

| 6273 | URB | 32.0 | 12 | 13 | 0 | 12 | N
| 6273 | URB | 25.0 | 8 | 11 | 0 | 12 | N
| 6273 | URB | 29.0 | 10 | 11 | 0 | 11 | N
| 6273 | URB | 33.0 | 12 | 14 | 0 | 13 | N
| 6273 | URB | 29.0 | 9 | 11 | 0 | 8 | N
| 6273 | URB | 30.0 | 7 | 11 | 0 | 11 | N
| 6273 | URB | 24.0 | 6 | 10 | 0 | 11 | N
| 6273 | URB | 32.0 | 16 | 14 | 0 | 9 | N
| 6273 | URB | 33.0 | 12 | 14 | 0 | 9 | N
+---+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

```

8. Creación de la variable operación salida

Consideramos operación salida al día anterior a un puente o periodo vacacional y al ultimo dia de un puente o periodo vacacional

```
In [ ]: # Ordena el DataFrame por una columna de fecha adecuada
df = df.orderBy('anno','mes','dia')

# Define una ventana para obtener el valor de 'puente' del día siguiente
window = Window.orderBy('anno','mes','dia')

# Obtiene el valor de 'puente' del día siguiente usando lag
next_puente = lag(col('puente')).over(window)

# Crea la columna 'operacion_salida' basada en las condiciones especificadas
operacion_salida_condition = (
    ((col('puente') == 0) & (next_puente == 1)) |
    ((col('puente') == 1) & (next_puente == 0)) |
    ((col('mes') == 7) & (col('dia').isin(14, 15, 31))) |
    ((col('mes') == 8) & (col('dia').isin(1, 14, 15)))
)
df = df.withColumn('operacion_salida', when(operacion_salida_condition, 1
df = df.drop('next_puente')

df.show()
```

```

23/09/01 18:26:55 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:55 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:26:55 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:29:19 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:29:19 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 18:29:28 WARN org.apache.spark.sql.execution.window.WindowExec:
[Stage 177:=====] (37 + 6

```

```
In [ ]: # Comprobación
df.where((col('anno') == 2022) & (col('mes') == 4) & (col('hora') == 0)).show()

23/09/01 19:27:27 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 19:27:58 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 19:27:58 WARN org.apache.spark.sql.execution.window.WindowExec:
[Stage 202:]
+---+-----+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error

```

| 6273 | URB | 6.0 | 3 | 4 | 0 | | 15 | N |
|------|-----|-------|----|----|---|--|----|---|
| 6069 | URB | 115.0 | 0 | 2 | 0 | | 15 | N |
| 6274 | URB | 33.0 | 0 | 1 | 0 | | 15 | N |
| 6070 | URB | 95.0 | 0 | 3 | 0 | | 15 | N |
| 6275 | URB | 33.0 | 5 | 12 | 0 | | 15 | N |
| 6071 | URB | 157.0 | 0 | 7 | 0 | | 15 | N |
| 6276 | URB | 8.0 | 1 | 5 | 0 | | 15 | N |
| 6072 | URB | 261.0 | 0 | 13 | 0 | | 15 | N |
| 6277 | URB | 12.0 | 1 | 3 | 0 | | 15 | N |
| 6073 | URB | 16.0 | 0 | 3 | 0 | | 15 | N |
| 6278 | URB | 13.0 | 2 | 3 | 0 | | 15 | N |
| 6074 | URB | 48.0 | 0 | 5 | 0 | | 15 | N |
| 6279 | URB | 46.0 | 15 | 12 | 0 | | 15 | N |
| 6075 | URB | 58.0 | 2 | 8 | 0 | | 15 | N |
| 6280 | URB | 10.0 | 2 | 2 | 0 | | 15 | N |
| 6076 | URB | 19.0 | 0 | 4 | 0 | | 15 | N |
| 6281 | URB | 26.0 | 4 | 9 | 0 | | 15 | N |
| 6077 | URB | 34.0 | 0 | 2 | 0 | | 15 | N |
| 6282 | URB | 5.0 | 0 | 2 | 0 | | 15 | N |
| 6078 | URB | 35.0 | 0 | 5 | 0 | | 15 | N |

only showing top 20 rows

OTRAS OPCIONES que no se han tenido en cuenta a la hora de exportar el dataset final

Opcion 2 con holiday y puentes.

Consideraremos los dias de operación salida:

- Los días previos a los periodos vacacionales y puentes
- Los últimos días de periodos vacacionales y de puentes
- 14,15,31 julio
- 1, 14,15 agosto

Opción 3

Poner días específicos de la DGT;

Para la DGT las operaciones salida son:

-Fin de semana anterior semana santa -Jueves santo y miercoles anterior -Ultimo fin de semana Junio -Ultimo fin de semana Julio -Fin de semana del puente del 15 -Fin de semana semana 51 -Fin de semana semana 52

Mientras que las operaciones retorno son:

-Domingo de ramos+Lunes de pascua -Ultimo fin de semana agosto

```
In [ ]: Definir las condiciones para operaciones salida
operaciones_salida_conditions = [
    # Fin de semana anterior a Semana Santa
    ((col("fecha") >= "2021-03-26") & (col("fecha") <= "2021-03-28")) |
    ((col("fecha") >= "2022-04-08") & (col("fecha") <= "2022-04-10")) |
    ((col("fecha") >= "2023-03-31") & (col("fecha") <= "2023-04-02")),
```

```

# Jueves Santo y Miércoles anterior
((col("fecha") == "2021-04-01") | (col("fecha") == "2021-03-31")) |
((col("fecha") == "2022-04-14") | (col("fecha") == "2022-04-13")) |
((col("fecha") == "2023-04-06") | (col("fecha") == "2023-04-05")),
# Último fin de semana de junio
((col("fecha") >= "2021-06-25") & (col("fecha") <= "2021-06-27")) |
((col("fecha") >= "2022-06-24") & (col("fecha") <= "2022-06-26"))
# Último fin de semana de julio
((col("fecha") >= "2021-07-30") & (col("fecha") <= "2021-08-01")) |
((col("fecha") >= "2022-07-29") & (col("fecha") <= "2022-07-31"))
# Fin de semana del puente del 15
((col("fecha") >= "2021-08-14") & (col("fecha") <= "2021-08-16")) |
((col("fecha") >= "2022-08-13") & (col("fecha") <= "2022-08-15"))
# Fin de semana semana 51
((col("fecha") >= "2021-12-17") & (col("fecha") <= "2021-12-19")) |
((col("fecha") >= "2022-12-16") & (col("fecha") <= "2022-12-18"))
# Fin de semana semana 52
((col("fecha") >= "2021-12-24") & (col("fecha") <= "2021-12-26")) |
((col("fecha") >= "2022-12-23") & (col("fecha") <= "2022-12-25"))
]

```

Definir las condiciones para operaciones retorno

```

operaciones_retorno_conditions = [
    # Domingo de Ramos + Lunes de Pascua
    ((col("fecha") == "2021-03-28") | (col("fecha") == "2021-04-05")) |
    ((col("fecha") == "2022-04-10") | (col("fecha") == "2022-04-18")) |
    ((col("fecha") == "2023-04-02") | (col("fecha") == "2023-04-10")),
    # Último fin de semana de agosto
    ((col("fecha") >= "2021-08-27") & (col("fecha") <= "2021-08-29")) |
    ((col("fecha") >= "2022-08-26") & (col("fecha") <= "2022-08-28"))
]
```

Aplicar las condiciones para crear las columnas

```
df_with_operaciones = df.withColumn("operacion_salida", when(any(cond for
```

Mostrar el DataFrame resultante

```
df_with_operaciones.show()
```

Cell In[32], line 1

Definir las condiciones para operaciones salida

^

SyntaxError: invalid syntax

9. Exportación dataset final

```
In [ ]: df.write.parquet("gs://bucket_traffic_estimation/Datasets/dataset_final.p
```

```

23/09/01 20:02:52 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:02:52 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:02:52 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:04:08 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:04:08 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:05:16 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:05:59 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:31:26 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:31:58 WARN org.apache.spark.sql.execution.window.WindowExec:
23/09/01 20:31:58 WARN org.apache.spark.sql.execution.window.WindowExec:
[Stage 220:>                                         (0 +

```


.4. Script de transformación de variables

```
In [ ]: from pyspark.sql.functions import *
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
```

```
In [ ]: from pyspark.sql import SparkSession

# Crea una instancia de SparkSession
spark = SparkSession.builder \
    .appName("Transformaciones") \
    .getOrCreate()
```

```
In [ ]: from IPython.core.display import HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))
```

```
In [ ]: df = spark.read.option("header", "true").parquet("gs://bucket_traffic_est")
df.show()
```

```
23/09/06 09:07:34 WARN org.apache.spark.sql.catalyst.util.package: Trunca
+-----+-----+-----+-----+-----+-----+
| id|tipo_elem|intensidad|ocupacion|carga|vmed|periodo_integracion|error
+---+-----+-----+-----+-----+-----+-----+
|1002|      M30|     987.0|       4|     0|   65|                  5|     N
|1002|      M30|     708.0|       3|     0|   68|                  5|     N
|1002|      M30|    1254.0|       5|     0|   68|                  5|     N
|1002|      M30|    1701.0|       7|     0|   66|                  5|     N
|1002|      M30|    1968.0|       9|     0|   64|                  5|     N
|1002|      M30|    1686.0|       7|     0|   65|                  5|     N
|1002|      M30|    1269.0|       6|     0|   66|                  5|     N
|1002|      M30|     888.0|       4|     0|   66|                  5|     N
|1002|      M30|     519.0|       2|     0|   64|                  5|     N
|1014|      M30|    879.0|       3|     0|   52|                  5|     N
|1014|      M30|    474.0|       1|     0|   53|                  5|     N
|1014|      M30|    786.0|       3|     0|   51|                  5|     N
|1014|      M30|    957.0|       3|     0|   51|                  5|     N
|1014|      M30|   1176.0|       4|     0|   51|                  5|     N
|1014|      M30|    711.0|       2|     0|   51|                  5|     N
|1014|      M30|    675.0|       2|     0|   52|                  5|     N
|1014|      M30|    528.0|       2|     0|   52|                  5|     N
|1014|      M30|    462.0|       1|     0|   54|                  5|     N
|1026|      M30|   177.0|       1|     0|   46|                  5|     N
|1026|      M30|    156.0|       0|     0|   48|                  5|     N
+---+-----+-----+-----+-----+-----+-----+
only showing top 20 rows
```

```
In [ ]: df.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- tipo_elem: string (nullable = true)
 |-- intensidad: double (nullable = true)
 |-- ocupacion: integer (nullable = true)
 |-- carga: integer (nullable = true)
 |-- vmed: integer (nullable = true)
 |-- periodo_integracion: integer (nullable = true)
 |-- error: string (nullable = true)
```

```

|--- fin_de_semana: string (nullable = true)
|--- festivo: string (nullable = true)
|--- tipo_de_festivo: string (nullable = true)
|--- temp: double (nullable = true)
|--- dwpt: double (nullable = true)
|--- rhum: double (nullable = true)
|--- prcp: double (nullable = true)
|--- wdir: double (nullable = true)
|--- wspd: double (nullable = true)
|--- pres: double (nullable = true)
|--- anno: integer (nullable = true)
|--- mes: integer (nullable = true)
|--- dia: integer (nullable = true)
|--- hora: integer (nullable = true)
|--- estacion: string (nullable = true)
|--- dia_semana: string (nullable = true)
|--- hora_punta: integer (nullable = true)
|--- condicion_adversa: integer (nullable = true)
|--- reduccion_tp: integer (nullable = true)
|--- puente: integer (nullable = true)
|--- vacacional: integer (nullable = true)
|--- operacion_salida: integer (nullable = true)

```

In []: `df.count()`

Out[]: 12492676

Funcion para obtener las mejores transformaciones de cada variable

Objetivo: Maximizar la correlación entre las variables numéricas y la variable numérica objetivo. Para ello, realizamos diferentes transformaciones y analizamos la correlación con la variable intensidad.

```

In [ ]: from pyspark.sql.functions import col, log, exp, pow, sqrt, corr
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler

def mejorTransfCorr(df, variable, target):
    # Calcula la correlación entre la variable original y la variable objetivo
    correlation_original = df.select(corr(variable, target)).first()[0]
    print(f"Correlación con {variable}: {correlation_original}")

    # Escala la variable
    max_value = df.selectExpr(f"max({variable})").collect()[0][0]
    min_value = df.selectExpr(f"min({variable})").collect()[0][0]
    df = df.withColumn(variable, (col(variable) - min_value) / (max_value))

    # Aplica las transformaciones a la variable
    df_transformed = df.withColumn("log_" + variable, log(col(variable)))
        .withColumn("exp_" + variable, exp(col(variable)))
        .withColumn("square_" + variable, pow(col(variable), 2))
        .withColumn("sqrt_" + variable, sqrt(col(variable)))
        .withColumn("quartic_" + variable, pow(col(variable), 4))
        .withColumn("root4_" + variable, pow(col(variable), 0.25))

    # Inicializa variables para mantener un seguimiento de la máxima correlación
    max_correlation = -1.0

```

```

max_correlation_transformation = None

# Calcula la correlación entre cada transformación y la variable objetivo
transformations = ["log_", "exp_", "square_", "sqrt_", "quartic_", "root4_"]
for transformation in transformations:
    correlation = df_transformed.select(corr(transformation + variable))
    if correlation is not None and correlation > max_correlation:
        max_correlation = correlation
        max_correlation_transformation = transformation + variable

    # Imprime la correlación para esta transformación
    #print(f"Correlación con {transformation + variable}: {correlation}")

return max_correlation_transformation

```

Llamamos a la función para cada una de las columnas numéricas de nuestro dataframe original

```

In [ ]: from pyspark.sql.functions import col, corr, log, exp, pow, sqrt
from pyspark.sql import SparkSession

# Supongamos que 'df' es tu DataFrame original
variable_objetivo = "intensidad"

# Obtener una lista de las columnas numéricas en el DataFrame excluyendo la variable objetivo
columnas_numericas = [col_name for col_name, data_type in df.dtypes if data_type != "string"]

# Crear un diccionario para almacenar las mejores transformaciones
mejores_transformaciones = {}

# Crear un diccionario para almacenar las correlaciones de variables y transformaciones
correlaciones_variables = {}

# Iterar a través de las columnas numéricas
for columna in columnas_numericas:
    mejor_transformacion = mejorTransfCorr(df, columna, variable_objetivo)
    mejores_transformaciones[columna] = mejor_transformacion

    # Almacenar las correlaciones en el diccionario de correlaciones
    correlations = {}
    df_transformed = df.withColumn("log_" + columna, log(col(columna))) \
        .withColumn("exp_" + columna, exp(col(columna))) \
        .withColumn("square_" + columna, pow(col(columna), 2)) \
        .withColumn("sqrt_" + columna, sqrt(col(columna))) \
        .withColumn("quartic_" + columna, pow(col(columna), 4)) \
        .withColumn("root4_" + columna, pow(col(columna), 4))

    transformations = ["log_", "exp_", "square_", "sqrt_", "quartic_", "root4_"]
    for transformation in transformations:
        correlation = df_transformed.select(corr(transformation + columna))
        correlaciones[transformation + columna] = correlation
    correlaciones_variables[columna] = correlaciones

# Crear el DataFrame de mejores transformaciones
mejores_transformaciones_df = spark.createDataFrame([(variable, transformation) for variable in columnas_numericas for transformation in transformations], ["variable", "transformation"])

# Crear una lista de filas para el DataFrame de correlaciones
filas_correlaciones = []
for variable, correlaciones in correlaciones_variables.items():
    fila_correlaciones = [variable] + [correlaciones[transformation + variable] for transformation in transformations]
    filas_correlaciones.append(fila_correlaciones)

```

```

# Crear el DataFrame de correlaciones
correlaciones_columns = ["variable"] + transformations
correlaciones_df = spark.createDataFrame(filas_correlaciones, correlacion

# Mostrar los resultados
print("Mejores transformaciones:")
mejores_transformaciones_df.show(mejores_transformaciones_df.count(), tru

print("Correlaciones de variables y transformaciones:")
correlaciones_df.show()

```

Correlación con ocupacion: 0.22509685785922626

Correlación con carga: 0.6079814155775752

Correlación con vmed: 0.2605032496922503

Correlación con periodo_integracion: 0.16363738141720044

Correlación con temp: 0.08867551741462577

Correlación con dwpt: -0.008585153072514565

Correlación con rhum: -0.11238203899708532

Correlación con prcp: 0.01458735841969392

Correlación con wdir: 0.04020683896782497

Correlación con wspd: 0.03190902171938506

Correlación con pres: -0.007193192159246279

Correlación con anno: 0.009173073739287399

Correlación con mes: 0.010998071739574284

Correlación con dia: 0.007468358844709748

Correlación con hora: 0.26431805371317296

Correlación con hora_punta: 0.1842945689926841

Correlación con condicion_adversa: -0.011138779509111645

Correlación con reduccion_tp: -0.3430530940230344

Correlación con puente: -0.05397721486453232

Correlación con vacacional: -0.06600773275556757

Correlación con operacion_salida: -0.02176557867035513

Mejores transformaciones:

+-----+-----+

| variable | mejor_transformacion |
|---------------------|-----------------------------|
| ocupacion | root4_ocupacion |
| carga | log_carga |
| vmed | exp_vmed |
| periodo_integracion | quartic_periodo_integracion |
| temp | log_temp |
| dwpt | log_dwpt |
| rhum | log_rhum |
| prcp | root4_prcp |
| wdir | log_wdir |
| wspd | square_wspd |
| pres | quartic_pres |
| anno | log_anno |
| mes | quartic_mes |
| dia | sqrt_dia |
| hora | log_hora |
| hora_punta | exp_hora_punta |
| condicion_adversa | root4_condicion_adversa |
| reduccion_tp | square_reduccion_tp |
| puente | square_puente |
| vacacional | root4_vacacional |
| operacion_salida | square_operacion_salida |

Correlaciones de variables y transformaciones:

| variable | log_ | exp_ |
|---------------------|---------------------------|----------------------|
| ocupacion | 0.2608913231580283 | -0.00749341171339... |
| carga | 0.6205260290723187 | 0.02505448036834506 |
| vmed | 0.2502843748283841 | 5.832723200125049E-4 |
| periodo_integracion | 0.13560308499274817 | -0.00378217466693... |
| temp | 0.08309758564546159 | 0.003679138277390... |
| dwpt | -0.01158615662618... | -0.01194088834714... |
| rhum | -0.10299101385341808 | -0.03167853060530872 |
| prcp | -0.02827068778256694 | 0.002230718583357... |
| wdir | 0.061489815554249705 | -0.0 0.01858915 |
| wspd | 0.009372563380587905 | 0.004410598032236135 |
| pres | -0.00722854449487... | Nan -0.0071581 |
| anno | 0.009170448201574925 | Nan 0.00917569 |
| mes | 0.008788905190970022 | 0.01683437056439751 |
| dia | 0.007390458985562019 | -0.00274396455650... |
| hora | 0.3006438768685868 | -0.04576021168528311 |
| hora_punta | null 0.18429456899267993 | 0.1842945 |
| condicion_adversa | null -0.01113877950911... | -0.0111387 |
| reduccion_tp | null -0.34305309402305195 | -0.343053 |
| puente | null -0.05397721486453... | -0.0539772 |
| vacacional | null -0.06600773275557152 | -0.0660077 |

only showing top 20 rows

Añadimos una columna que indique la relación entre las variables originales y la variable objetivo intensidad

```
In [ ]: from pyspark.sql import Row

# Crear un nuevo DataFrame para almacenar las correlaciones originales
correlaciones_originales = []

# Calcular la correlación de las variables originales con 'intensidad' y
```

```

for columna in columnas_numericas:
    correlation_original = df.select(corr(columna, variable_objetivo).alias("correlation"))
    correlaciones_originales.append(Row(variable=columna, original=correlation))

# Crear un DataFrame a partir de correlaciones_originales
correlaciones_originales_df = spark.createDataFrame(correlaciones_originales)

# Unir los DataFrames por la columna "variable"
correlaciones_df = correlaciones_df.join(correlaciones_originales_df, on="variable")

# Mostrar el DataFrame de correlaciones actualizado con las correlaciones originales
print("Correlaciones de variables originales con 'intensidad':")
correlaciones_df.show()

```

| | variable | log_1 | exp_1 |
|---------------------|----------------------|----------------------|-----------------|
| carga | 0.6205260290723187 | 0.02505448036834506 | 0.475871 |
| condicion_adversa | null | -0.01113877950911... | -0.0111387 |
| dia | 0.007390458985562019 | -0.00274396455650... | 0.00654380 |
| hora | 0.3006438768685868 | -0.04576021168528311 | 0.1960851 |
| operacion_salida | null | -0.02176557867035... | -0.0217655 |
| prcp | -0.02827068778256694 | 0.002230718583357... | 0.00824282 |
| pres | -0.00722854449487... | | NaN -0.0071581 |
| reduccion_tp | null | -0.34305309402305195 | -0.343053 |
| temp | 0.08309758564546159 | 0.003679138277390... | 0.07208 |
| vacacional | null | -0.06600773275557152 | -0.0660077 |
| anno | 0.009170448201574925 | | NaN 0.00917569 |
| dwpt | -0.01158615662618... | -0.01194088834714... | -0.0151150 |
| hora_punta | null | 0.18429456899267993 | 0.1842945 |
| mes | 0.008788905190970022 | 0.01683437056439751 | 0.01441616 |
| ocupacion | 0.2608913231580283 | -0.00749341171339... | 0.0668969 |
| periodo_integracion | 0.13560308499274817 | -0.00378217466693... | 0.1840420 |
| puentel | null | -0.05397721486453... | -0.0539772 |
| vmed | 0.2502843748283841 | 5.832723200125049E-4 | 0.2512677 |
| wdir | 0.061489815554249705 | | -0.0 0.01858915 |
| wspd | 0.009372563380587905 | 0.004410598032236135 | 0.04394970 |

only showing top 20 rows

Creamos el dataframe resultante que contenga las mejores transformaciones y la variable objetivo

```

In [ ]: df_mejtrans = df

df_mejtrans = df_mejtrans.withColumn("root4_ocupacion", pow(col("ocupacion"), 4))
df_mejtrans = df_mejtrans.withColumn("log_carga", log(col("carga")))
df_mejtrans = df_mejtrans.withColumn("exp_vmed", exp(col("vmed")))
df_mejtrans = df_mejtrans.withColumn("quartic_periodo_integracion", pow(col("periodo_integracion"), 4))
df_mejtrans = df_mejtrans.withColumn("log_temp", log(col("temp")))
df_mejtrans = df_mejtrans.withColumn("log_dwpt", log(col("dwpt")))
df_mejtrans = df_mejtrans.withColumn("log_rhum", log(col("rhum")))
df_mejtrans = df_mejtrans.withColumn("root4_prcp", pow(col("prcp"), 1/4))
df_mejtrans = df_mejtrans.withColumn("log_wdir", log(col("wdir")))
df_mejtrans = df_mejtrans.withColumn("square_wspd", pow(col("wspd"), 2))
df_mejtrans = df_mejtrans.withColumn("quartic_pres", pow(col("pres"), 4))
df_mejtrans = df_mejtrans.withColumn("log_anno", log(col("anno")))
df_mejtrans = df_mejtrans.withColumn("quartic_mes", pow(col("mes"), 4))
df_mejtrans = df_mejtrans.withColumn("sqrt_dia", sqrt(col("dia")))

```

```

df_mejtrans = df_mejtrans.withColumn("log_hora", log(col("hora")))
df_mejtrans = df_mejtrans.withColumn("exp_hora_punta", exp(col("hora_punt"))
df_mejtrans = df_mejtrans.withColumn("root4_condicion_adversa", pow(col("adver
df_mejtrans = df_mejtrans.withColumn("square_reduccion_tp", pow(col("reduccio
df_mejtrans = df_mejtrans.withColumn("square_puente", pow(col("puente")),
df_mejtrans = df_mejtrans.withColumn("root4_vacacional", pow(col("vacaciona
df_mejtrans = df_mejtrans.withColumn("square_operacion_salida", pow(col("operac

```

```

In [ ]: columnas_a_eliminar = [
    'ocupacion', 'carga', 'vmed', 'periodo_integracion', 'error', 'temp',
    'wspd', 'pres', 'anno', 'mes', 'dia', 'hora', 'hora_punta', 'condicion',
    'puente', 'vacacional', 'operacion_salida'
]

# Elimina las columnas especificadas
df_mejtrans = df_mejtrans.drop(*columnas_a_eliminar)

df_mejtrans = df_mejtrans.fillna(0)

```

```
In [ ]: df_mejtrans.show()
```

| id | tipo_elem | intensidad | fin_de_semana | festivo | tipo_de_festivo | estacion |
|-------|-----------|------------|---------------|---------|-----------------|----------|
| 10021 | M30 | 987.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 708.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 1254.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 1701.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 1968.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 1686.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 1269.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 888.0 | 0 | 1 | Nacional | Inviero |
| 10021 | M30 | 519.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 879.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 474.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 786.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 957.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 1176.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 711.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 675.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 528.0 | 0 | 1 | Nacional | Inviero |
| 10141 | M30 | 462.0 | 0 | 1 | Nacional | Inviero |
| 10261 | M30 | 177.0 | 0 | 1 | Nacional | Inviero |
| 10261 | M30 | 156.0 | 0 | 1 | Nacional | Inviero |

only showing top 20 rows

```

In [ ]: # Guardar el nuevo DataFrame en un archivo Parquet
df_mejtrans.write.mode("overwrite").parquet("gs://bucket_traffic_estimati

```

```
In [ ]:
```

.5. Script de PCA

PCA

```
In [ ]: from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.ml.feature import VectorAssembler, OneHotEncoder, StringIndexer
from pyspark.ml import Pipeline
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import PCA
```

```
In [ ]: # Crea una instancia de SparkSession
spark = SparkSession.builder \
    .appName("PCA") \
    .getOrCreate()
```

```
In [ ]: from IPython.core.display import HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))
```

```
In [ ]: df = spark.read.option("header", "true").parquet("gs://bucket_traffic_est")
df.show()
```

```
23/09/05 19:32:21 WARN org.apache.spark.sql.catalyst.util.package: Truncating data to fit in memory
```

| id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion |
|------|-----------|--------------------|-----------|-------|------|---------------------|
| 1001 | M30 | 1449.0 | 3 | 0 | 62 | |
| 1001 | M30 | 732.0 | 1 | 0 | 63 | |
| 1001 | M30 | 1224.0 | 3 | 0 | 62 | |
| 1001 | M30 | 1782.0 | 4 | 0 | 62 | |
| 1001 | M30 | 283.95715588406466 | 6 | 0 | 60 | |
| 1001 | M30 | 1686.0 | 4 | 0 | 60 | |
| 1001 | M30 | 1278.0 | 3 | 0 | 60 | |
| 1001 | M30 | 906.0 | 2 | 0 | 64 | |
| 1001 | M30 | 552.0 | 1 | 0 | 59 | |
| 1002 | M30 | 987.0 | 4 | 0 | 65 | |
| 1002 | M30 | 708.0 | 3 | 0 | 68 | |
| 1002 | M30 | 1254.0 | 5 | 0 | 68 | |
| 1002 | M30 | 1701.0 | 7 | 0 | 66 | |
| 1002 | M30 | 1968.0 | 9 | 0 | 64 | |
| 1002 | M30 | 1686.0 | 7 | 0 | 65 | |
| 1002 | M30 | 1269.0 | 6 | 0 | 66 | |
| 1002 | M30 | 888.0 | 4 | 0 | 66 | |
| 1002 | M30 | 519.0 | 2 | 0 | 64 | |
| 1003 | M30 | 1299.0 | 5 | 0 | 67 | |
| 1003 | M30 | 930.0 | 3 | 0 | 68 | |

only showing top 20 rows

```
In [ ]: columnas_numericas = [
    'intensidad',
    'ocupacion',
    'carga',
    'vmed',
    'periodo_integracion',
    'temp',
    'dwpt',
```

```
'rhum',
'prcp',
'wdir',
'wspd',
'pres',
'anno',
'mes',
'dia',
' hora',
' hora_punta',
'condicion_adversa',
'reduccion_tp',
'puente',
'vacacional',
'operacion_salida']
columnas_categoricas = ['tipo_elem', 'error', 'fin_de_semana', 'festivo',
```

```
In [ ]: # Indexar las variables categóricas
indexadores = [StringIndexer(inputCol=col, outputCol=col + "_index") for

# One-Hot Encoding de las variables categóricas indexadas
codificadores = [OneHotEncoder(inputCol=col + "_index", outputCol=col + "
```

```
In [ ]: # Ensamblar todas las características en un solo vector
ensamblador = VectorAssembler(inputCols=columnas_numericas + [col + "_enc"

# Escalar características
escalador = StandardScaler(inputCol="features", outputCol="scaled_feature
```

```
In [ ]: # Crear una instancia de PCA con el número de componentes deseado (k)
pca = PCA(k=10, inputCol="scaled_features", outputCol="pca_features")
```

```
In [ ]: # Construir el pipeline con todas las etapas, incluida PCA
pipeline = Pipeline(stages=indexadores + codificadores + [ensamblador, es
```

```
In [ ]: # Ajustar el pipeline al DataFrame
model = pipeline.fit(df)
```

```
23/09/05 20:02:43 WARN com.github.fommil.netlib.LAPACK: Failed to load im
23/09/05 20:02:43 WARN com.github.fommil.netlib.LAPACK: Failed to load im
```

```
In [ ]: # Obtener los resultados del PCA
result = model.transform(df)
```

```
In [ ]: import numpy as np

# Acceder al modelo PCA desde el modelo pipeline
pca_model = model.stages[-1]

# Obtener la varianza explicada
explained_variance = pca_model.explainedVariance

# Calcular la varianza explicada acumulada
cumulative_variance = np.cumsum(explained_variance)
```

```
In [ ]: import matplotlib.pyplot as plt

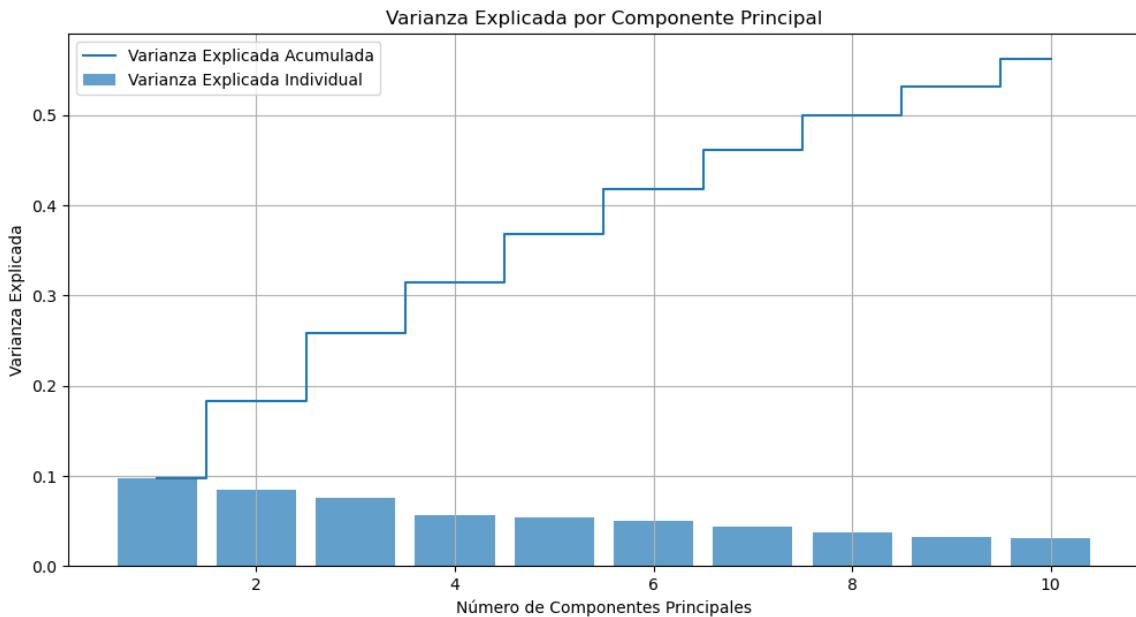
# Crear un gráfico de barras para la varianza explicada por cada componente
plt.figure(figsize=(12, 6))
plt.bar(range(1, len(explained_variance) + 1), explained_variance, alpha=0.5)
plt.step(range(1, len(explained_variance) + 1), cumulative_variance, where='mid')
```

```

plt.xlabel('Número de Componentes Principales')
plt.ylabel('Varianza Explicada')
plt.title('Varianza Explicada por Componente Principal')
plt.legend()
plt.grid(True)

# Mostrar el gráfico
plt.show()

```



```

In [ ]: # Obtener los loadings de las variables en cada componente principal y tr
loadings = pca_model.pc.toArray().T # Transponer los datos

# Crear un gráfico de barras apiladas para mostrar las contribuciones de
plt.figure(figsize=(12, 8))
num_componentes = len(loadings)
num_variables = len(columnas_numericas + [col + "_encoded" for col in columnas_categoricas])
variables = columnas_numericas + [col + "_encoded" for col in columnas_categoricas]

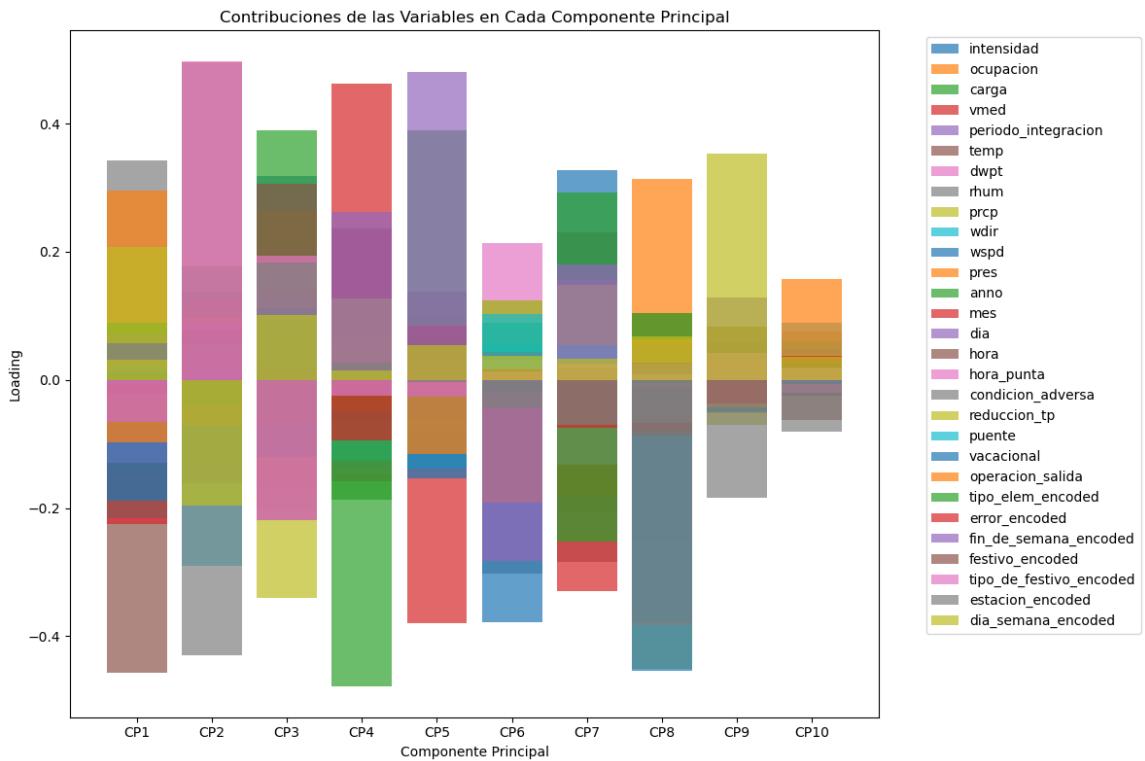
for i in range(num_variables):
    plt.bar(range(num_componentes), loadings[:, i], alpha=0.7, label=variables[i])

plt.title('Contribuciones de las Variables en Cada Componente Principal')
plt.xlabel('Componente Principal')
plt.ylabel('Loading')
plt.legend()
plt.xticks(range(num_componentes), [f'CP{i+1}' for i in range(num_componentes)])
plt.tight_layout()

# Colocar la leyenda fuera del gráfico
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

plt.show()

```



```
In [ ]: # Vamos a poner un umbral de 0.95 de varianza explicada, es decir, mantener
threshold = 0.95 # Umbral del 95% de varianza explicada
cumulative_variance = 0
num_components_to_keep = 0

for variance in explained_variance:
    cumulative_variance += variance
    num_components_to_keep += 1
    if cumulative_variance >= threshold:
        break

print(f"Número de componentes principales a retener: {num_components_to_k}
```

Número de componentes principales a retener: 10

```
In [ ]: # Ahora hacemos nuevamente PCA con el número óptimo de componentes principales
pca = PCA(k=num_components_to_keep, inputCol="scaled_features", outputCol="pca_features")

# Construir el pipeline
pipeline = Pipeline(stages=indexadores + codificadores + [ensamblador, estacion, dia_semana])

# Ajustar el pipeline al DataFrame
final_model = pipeline.fit(df)

pca_df = final_model.transform(df)
```

[Stage 52:> (0 +

```
In [ ]: # Obtener los resultados del PCA
pca_result = result.select("pca_features")

# Mostrar los resultados del PCA
pca_result.show(truncate=False)
```

```
In [ ]: pca_result.write.parquet("gs://bucket_traffic_estimation/Datasets/pca.parquet")
```

.6. Scripts de comparación de modelos sencillos

.6.1. Script comparación modelos sencillos dataset normal

Entrenamiento de modelos

1. Lectura del fichero

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor
```

```
In [ ]: import pandas as pd

# Ruta al archivo Parquet
ruta_archivo_parquet = "C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Database.parquet"

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
```

```
In [ ]: df.head(20)
```

```
Out[ ]:
```

| | id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion | error | fin_d |
|----|------|-----------|------------|-----------|-------|------|---------------------|-------|-------|
| 0 | 1026 | M30 | 177.0 | 1 | 0 | 46 | | 5 | N |
| 1 | 1026 | M30 | 156.0 | 0 | 0 | 48 | | 5 | N |
| 2 | 1026 | M30 | 255.0 | 1 | 0 | 57 | | 5 | N |
| 3 | 1026 | M30 | 330.0 | 2 | 0 | 57 | | 5 | N |
| 4 | 1026 | M30 | 372.0 | 3 | 0 | 56 | | 5 | N |
| 5 | 1026 | M30 | 306.0 | 2 | 0 | 57 | | 5 | N |
| 6 | 1026 | M30 | 198.0 | 1 | 0 | 55 | | 5 | N |
| 7 | 1026 | M30 | 87.0 | 0 | 0 | 41 | | 5 | N |
| 8 | 1026 | M30 | 48.0 | 0 | 0 | 39 | | 5 | N |
| 9 | 1049 | M30 | 573.0 | 2 | 0 | 63 | | 5 | N |
| 10 | 1049 | M30 | 327.0 | 1 | 0 | 59 | | 5 | N |
| 11 | 1049 | M30 | 444.0 | 2 | 0 | 62 | | 5 | N |
| 12 | 1049 | M30 | 771.0 | 3 | 0 | 61 | | 5 | N |
| 13 | 1049 | M30 | 870.0 | 3 | 0 | 63 | | 5 | N |
| 14 | 1049 | M30 | 693.0 | 3 | 0 | 64 | | 5 | N |
| 15 | 1049 | M30 | 543.0 | 2 | 0 | 64 | | 5 | N |
| 16 | 1049 | M30 | 357.0 | 1 | 0 | 63 | | 5 | N |

| | | | | | | | | |
|-----------|------|-----|-------|----|----|----|----|---|
| 17 | 1049 | M30 | 222.0 | 1 | 0 | 62 | 5 | N |
| 18 | 3445 | URB | 89.0 | 26 | 10 | 0 | 15 | N |
| 19 | 3445 | URB | 54.0 | 15 | 7 | 0 | 15 | N |

20 rows × 30 columns

```
In [ ]: df.dtypes
```

```
Out[ ]: id           int32
         tipo_elem    object
         intensidad   float64
         ocupacion    int32
         carga         int32
         vmed          int32
         periodo_integracion int32
         error         object
         fin_de_semana object
         festivo        object
         tipo_de_festivo object
         temp           float64
         dwpt           float64
         rhum           float64
         prcp           float64
         wdir           float64
         wspd           float64
         pres            float64
         anno           int32
         mes            int32
         dia             int32
         hora            int32
         estacion        object
         dia_semana     object
         hora_punta     int32
         condicion_adversa int32
         reduccion_tp   int32
         puente          int32
         vacacional      int32
         operacion_salida int32
dtype: object
```

```
In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion', 'vmed', 'periodo'])
y = df['intensidad']
```

```
In [ ]: # Codifica las características categóricas utilizando one-hot encoding  
X = pd.get_dummies(X)
```

Tn [] : x

| | | | | | | | | | | |
|--------|-------|---|----|------|-----|------|-----|------|------|--------|
| 859235 | 11004 | 0 | 14 | 29.0 | 2.3 | 18.0 | 0.0 | 41.0 | 18.4 | 1015.5 |
| 859236 | 11004 | 0 | 15 | 27.6 | 4.0 | 22.0 | 0.0 | 30.0 | 19.8 | 1015.8 |
| 859237 | 11004 | 0 | 14 | 25.7 | 4.2 | 25.0 | 0.0 | 16.0 | 22.7 | 1016.4 |
| 859238 | 11004 | 0 | 13 | 24.1 | 5.4 | 30.0 | 0.0 | 14.0 | 21.6 | 1017.0 |
| 859239 | 11004 | 0 | 15 | 23.1 | 6.4 | 34.0 | 0.0 | 15.0 | 20.9 | 1017.8 |

859240 rows × 43 columns

```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import BaggingRegressor
# Define los modelos de regresión que deseas probar
models = []

seed = 99

models.append(('DTR', DecisionTreeRegressor(random_state=seed)))
models.append(('KNNR', KNeighborsRegressor()))
models.append(('RFR', RandomForestRegressor(random_state=seed)))
models.append(('XGB', XGBRegressor(random_state=seed)))
models.append(('GBR', GradientBoostingRegressor(random_state=seed)))
models.append(('BAG', BaggingRegressor(base_estimator=DecisionTreeRegressor(
    random_state=seed), n_estimators=100, random_state=99)))

for name, model in models:
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    print(f'{name}: {score}')
    # Realizar predicciones en el conjunto de prueba
    y_pred = model.predict(X_test)

    # Calcular el RMSE
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    print(f'RMSE: {rmse}')

DTR: 0.8276121352993223
RMSE: 150.46736523220818
KNNR: 0.6501248440007186
RMSE: 214.36087628377487
RFR: 0.9072536287196962
RMSE: 110.36654176662384
XGB: 0.8538407710742387
RMSE: 138.5485843887316
GBR: 0.6451112753918471
RMSE: 215.8912653844647
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
BAG: 0.8968821029770895
RMSE: 116.37400878472737
```


.6.2. Script comparación modelos sencillos dataset transformado

Entrenamiento de modelos

1. Lectura del fichero

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
```

```
In [ ]: import pandas as pd

# Ruta al archivo Parquet
ruta_archivo_parquet = "C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Database.parquet"

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
```

```
In [ ]: df.head(20)
```

```
Out[ ]:
```

| | id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion | error | fin_de |
|----|------|-----------|------------|-----------|-------|------|---------------------|-------|--------|
| 0 | 1026 | M30 | 177.0 | 1 | 0 | 46 | | 5 | N |
| 1 | 1026 | M30 | 156.0 | 0 | 0 | 48 | | 5 | N |
| 2 | 1026 | M30 | 255.0 | 1 | 0 | 57 | | 5 | N |
| 3 | 1026 | M30 | 330.0 | 2 | 0 | 57 | | 5 | N |
| 4 | 1026 | M30 | 372.0 | 3 | 0 | 56 | | 5 | N |
| 5 | 1026 | M30 | 306.0 | 2 | 0 | 57 | | 5 | N |
| 6 | 1026 | M30 | 198.0 | 1 | 0 | 55 | | 5 | N |
| 7 | 1026 | M30 | 87.0 | 0 | 0 | 41 | | 5 | N |
| 8 | 1026 | M30 | 48.0 | 0 | 0 | 39 | | 5 | N |
| 9 | 1049 | M30 | 573.0 | 2 | 0 | 63 | | 5 | N |
| 10 | 1049 | M30 | 327.0 | 1 | 0 | 59 | | 5 | N |
| 11 | 1049 | M30 | 444.0 | 2 | 0 | 62 | | 5 | N |
| 12 | 1049 | M30 | 771.0 | 3 | 0 | 61 | | 5 | N |
| 13 | 1049 | M30 | 870.0 | 3 | 0 | 63 | | 5 | N |
| 14 | 1049 | M30 | 693.0 | 3 | 0 | 64 | | 5 | N |
| 15 | 1049 | M30 | 543.0 | 2 | 0 | 64 | | 5 | N |
| 16 | 1049 | M30 | 357.0 | 1 | 0 | 63 | | 5 | N |
| 17 | 1049 | M30 | 222.0 | 1 | 0 | 62 | | 5 | N |

| | | | | | | | | | |
|----|------|-----|------|----|----|---|--|----|---|
| 18 | 3445 | URB | 89.0 | 26 | 10 | 0 | | 15 | N |
| 19 | 3445 | URB | 54.0 | 15 | 7 | 0 | | 15 | N |

20 rows × 30 columns

In []: df.dtypes

```
Out[ ]: id                  int32
tipo_elem            object
intensidad          float64
ocupacion            int32
carga                int32
vmed                int32
periodo_integracion int32
error               object
fin_de_semana        object
festivo              object
tipo_de_festivo      object
temp                float64
dwpt                float64
rhum                float64
prcp                float64
wdir                float64
wspd                float64
pres                float64
anno                int32
mes                 int32
dia                 int32
hora                int32
estacion             object
dia_semana           object
hora_punta            int32
condicion_adversa    int32
reduccion_tp           int32
puente               int32
vacacional            int32
operacion_salida      int32
dtype: object
```

In []: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion', 'vmed']) # Asum
y = df['intensidad']

In []: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)

In []: X

```
Out[ ]:   id  vmed  periodo_integracion  temp  dwpt  rhum  prcp  wdir  wspd  pres
0   1026  46                   5  6.1  -3.6  50.0  0.0  291.0  18.7  1010.4
1   1026  48                   5  5.8  -3.3  52.0  0.0  287.0  18.7  1010.4
2   1026  57                   5  5.2  -2.6  57.0  0.0  279.0  14.8  1010.6
3   1026  57                   5  3.4  -2.6  65.0  0.0  280.0  12.2  1011.4
4   1026  56                   5  2.9  -2.4  68.0  0.0  277.0  12.6  1012.0
...   ...  ...
859235 11004  0                  14  29.0  2.3  18.0  0.0  41.0  18.4  1015.5
```

| | | | | | | | | | | |
|--------|-------|---|----|------|-----|------|-----|------|------|--------|
| 859236 | 11004 | 0 | 15 | 27.6 | 4.0 | 22.0 | 0.0 | 30.0 | 19.8 | 1015.8 |
| 859237 | 11004 | 0 | 14 | 25.7 | 4.2 | 25.0 | 0.0 | 16.0 | 22.7 | 1016.4 |
| 859238 | 11004 | 0 | 13 | 24.1 | 5.4 | 30.0 | 0.0 | 14.0 | 21.6 | 1017.0 |
| 859239 | 11004 | 0 | 15 | 23.1 | 6.4 | 34.0 | 0.0 | 15.0 | 20.9 | 1017.8 |

859240 rows × 43 columns

```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import BaggingRegressor
# Define los modelos de regresión que deseas probar
models = []

seed = 99

models.append(('LR', LinearRegression()))
models.append(('DTR', DecisionTreeRegressor(random_state=seed)))
models.append(('KNNR', KNeighborsRegressor()))
models.append(('Lasso', Lasso()))
models.append(('Ridge', Ridge()))
models.append(('RFR', RandomForestRegressor(random_state=seed)))
models.append(('EN', ElasticNet(random_state=seed)))
models.append(('MLP', MLPRegressor(random_state=seed)))
models.append(('XGB', XGBRegressor(random_state=seed)))
models.append(('GBR', GradientBoostingRegressor(random_state=seed)))
models.append(('BAG', BaggingRegressor(base_estimator=DecisionTreeRegressor(
    random_state=seed), n_estimators=10, random_state=seed)))

for name, model in models:
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    print(f'{name}: {score}')
    # Realizar predicciones en el conjunto de prueba
    y_pred = model.predict(X_test)

    # Calcular el RMSE
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    print(f'RMSE: {rmse}'
```

```
LR: 0.24788194664627672
RMSE: 314.29102391007746
DTR: 0.8290841199970818
RMSE: 149.8235829853273
KNNR: -0.1495175489392837
RMSE: 388.5498754124203
Lasso: 0.2449499618218256
RMSE: 314.90302902436184
```

```
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\linear_model\_ridge.py:216: LinAlgWarning: Ill-conditioned matrix (rcond=2.40234e-27): result may not be accurate.
    return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
Ridge: 0.2478823262518931
RMSE: 314.29094459629664
```

```
RFR: 0.9063712856605315
RMSE: 110.89028537477141
EN: 0.19219447097682274
RMSE: 325.71846680643773
MLP: -0.5428459281636222
RMSE: 450.14274037304693
XGB: 0.8633926444637766
RMSE: 133.94484786727182
GBR: 0.6464232167926045
RMSE: 215.49184590515853

c:\Users\evaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
BAG: 0.8958430668625234
RMSE: 116.95884289723311
```

.7. Scripts de grid searchs

.7.1. Bagging dataset original

Entrenamiento de modelos

1. Lectura del fichero

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import cross_val_score

import numpy as np
import matplotlib.pyplot as plt
```

```
In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/AlejandroAlvarez/OneDrive - Rock Interne

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
```

```
In [ ]: df.columns
```

```
Out[ ]: Index(['id', 'tipo_elem', 'intensidad', 'ocupacion', 'carga', 'vmed',
       'periodo_integracion', 'error', 'fin_de_semana', 'festivo',
       'tipo_de_festivo', 'temp', 'dwpt', 'rhum', 'prcp', 'wdir', 'wspd',
       'pres', 'anno', 'mes', 'dia', 'hora', 'estacion', 'dia_semana',
       'hora_punta', 'condicion_adversa', 'reduccion_tp', 'puente',
       'vacacional', 'operacion_salida'],
      dtype='object')
```

```
In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion', 'vmed', 'error',
                      'hora_punta', 'condicion_adversa', 'reduccion_tp', '']
y = df['intensidad']
```

```
In [ ]: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)
```

```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: # Define el modelo DecisionTreeRegressor
dt = DecisionTreeRegressor()
```

```

# Define los valores que deseas probar para los hiperparámetros
param_grid = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [1, 2, 3],
    'min_impurity_decrease': [0.0, 0.1, 0.2]
}

# Define la métrica que deseas utilizar para la validación cruzada (por e
scorer = make_scorer(r2_score)

# Configurar la búsqueda de cuadricula
grid_search = GridSearchCV(dt, param_grid, scoring=scorer, cv=5)

# Realizar la búsqueda de cuadricula en los datos
grid_search.fit(X, y)

```

Out[]:

```

▶      GridSearchCV
▶estimator: DecisionTreeRegressor
    ▶DecisionTreeRegressor

```

In []:

```

# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_

# Imprimir los mejores hiperparámetros
print("Mejores hiperparámetros:")
print(best_params)

```

Mejores hiperparámetros:

```

{'max_depth': 15, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'm
in_samples_split': 15}

```

In []:

```

# Modelo preentrenado, definimos best_params
best_params = {'max_depth': 15, 'min_impurity_decrease': 0.1, 'min_sample

# Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros
best_model = BaggingRegressor(
    base_estimator=DecisionTreeRegressor(**best_params),
    n_estimators=100,
    random_state=99
)

# Entrenar el modelo
best_model.fit(X_train, y_train)

```

```

c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(

```

Out[]:

```

▶      BaggingRegressor
▶base_estimator: DecisionTreeRegressor
    ▶DecisionTreeRegressor

```

In []:

```

# Predecir los valores de y para los datos de prueba
y_pred = best_model.predict(X_test)

```

```
In [ ]: # Calcular el error cuadrático medio
mse = mean_squared_error(y_test, y_pred)

# Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse)

# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:", np.sqrt(mse))

# Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_score(y_test, y_pr

# Imprimir el coeficiente de determinación ajustado (R-cuadrado ajustado)
print("Coeficiente de determinación ajustado (R-cuadrado ajustado):", 1 -
```

Error cuadrático medio: 17428.43837019701
Raíz del error cuadrático medio: 132.01681093783856
Coeficiente de determinación (R-cuadrado): 0.8672970636546325
Coeficiente de determinación ajustado (R-cuadrado ajustado): 0.8672847070543594

```
In [ ]: # Validación cruzada el mejor modelo
scores = cross_val_score(best_model, X, y, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
Resultados de la validación cruzada:
[0.73157955 0.71392795 0.79686111 0.77946557 0.75139651]
Media de los resultados de la validación cruzada:
```

```
0.7546461380671626
Desviación estándar de los resultados de la validación cruzada:
0.030327236734514273
Precisión de la validación cruzada:
R2: 0.75 (+/- 0.06)
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
Resultados de la validación cruzada:
[0.86448176 0.86543862 0.86276818 0.86468805 0.86317578]
Media de los resultados de la validación cruzada:
0.8641104777215803
Desviación estándar de los resultados de la validación cruzada:
0.0009910446207281821
Precisión de la validación cruzada:
R2: 0.86 (+/- 0.00)
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_test, y_test, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())
```

```

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\AlejandroAlvarez\AppData\Local\Programs\Python\Python39\lib\site
-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn()

Resultados de la validación cruzada:
[0.84712137 0.85065133 0.84500174 0.84061403 0.85342202]
Media de los resultados de la validación cruzada:
0.8473620977176685
Desviación estándar de los resultados de la validación cruzada:
0.004443138100089323
Precisión de la validación cruzada:
R2: 0.85 (+/- 0.01)

```

```

In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Puntuación")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                    train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                    test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Prueba")

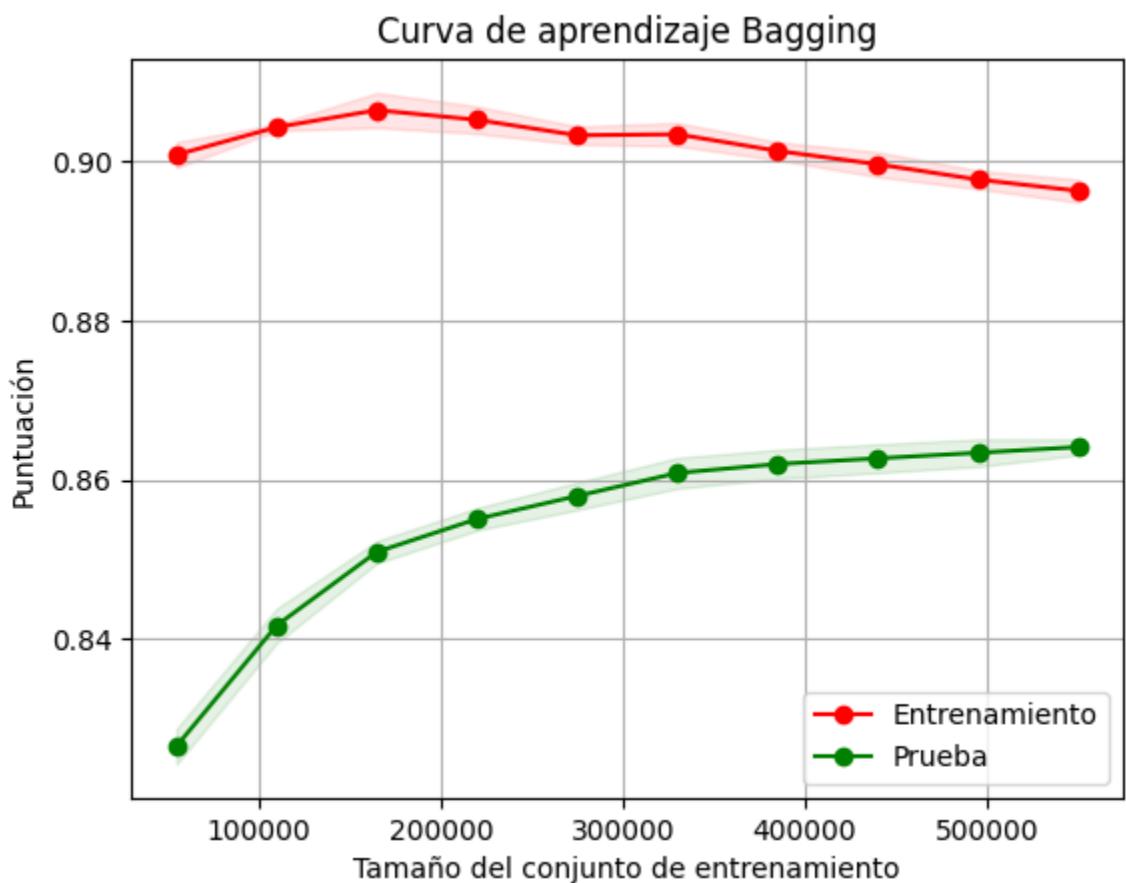
```

```

plt.legend(loc="best")
return plt

plt = plot_learning_curve(best_model, "Curva de aprendizaje Bagging", X_t

```

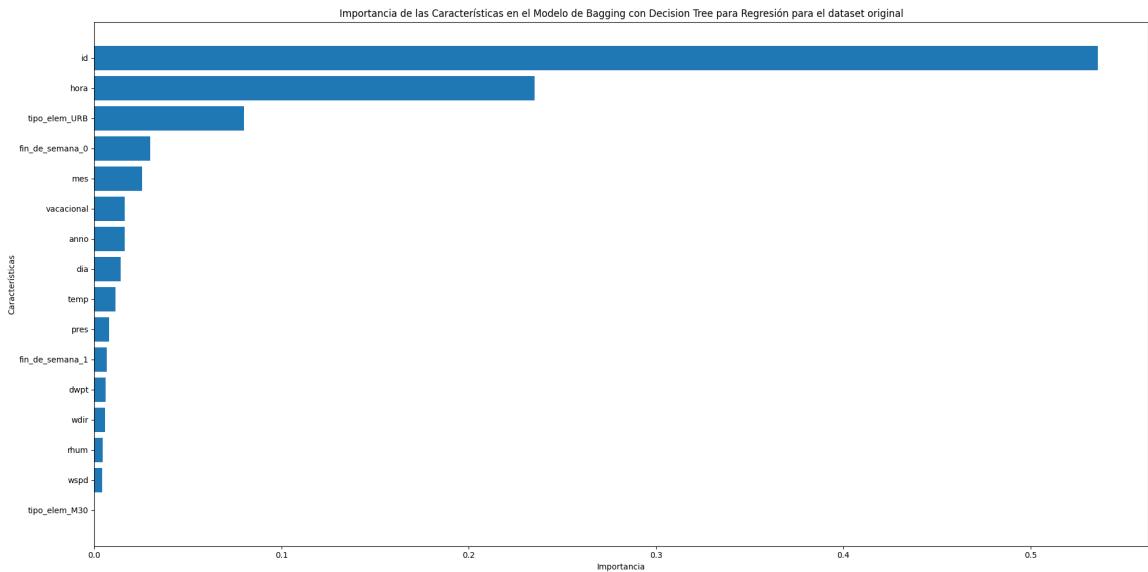


```
In [ ]: # Obtén la importancia de las características
feature_importance = best_model.estimators_[0].feature_importances_
```

```
In [ ]: # Ordenar las características por importancia de mayor a menor
sorted_idx = np.argsort(feature_importance)

# Crear el gráfico de barras horizontales
plt.figure(figsize=(20, 10)) # Ajusta el tamaño del gráfico según sea necesario
plt.barh(range(len(feature_importance)), feature_importance[sorted_idx])
plt.xlabel('Importancia')
plt.ylabel('Características')
plt.title('Importancia de las Características en el Modelo de Bagging con')
plt.yticks(range(len(feature_importance)), [X.columns[i] for i in sorted_idx])
plt.tight_layout()

# Mostrar el gráfico
plt.show()
```



```
In [ ]: # Guardar el modelo
import joblib

joblib.dump(best_model, 'bagging_filtrado.pkl')
```

```
Out[ ]: ['bagging_filtrado.pkl']
```

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import validation_curve

# Define el rango de valores de n_estimators que deseas probar
param_range = [5, 10, 15]

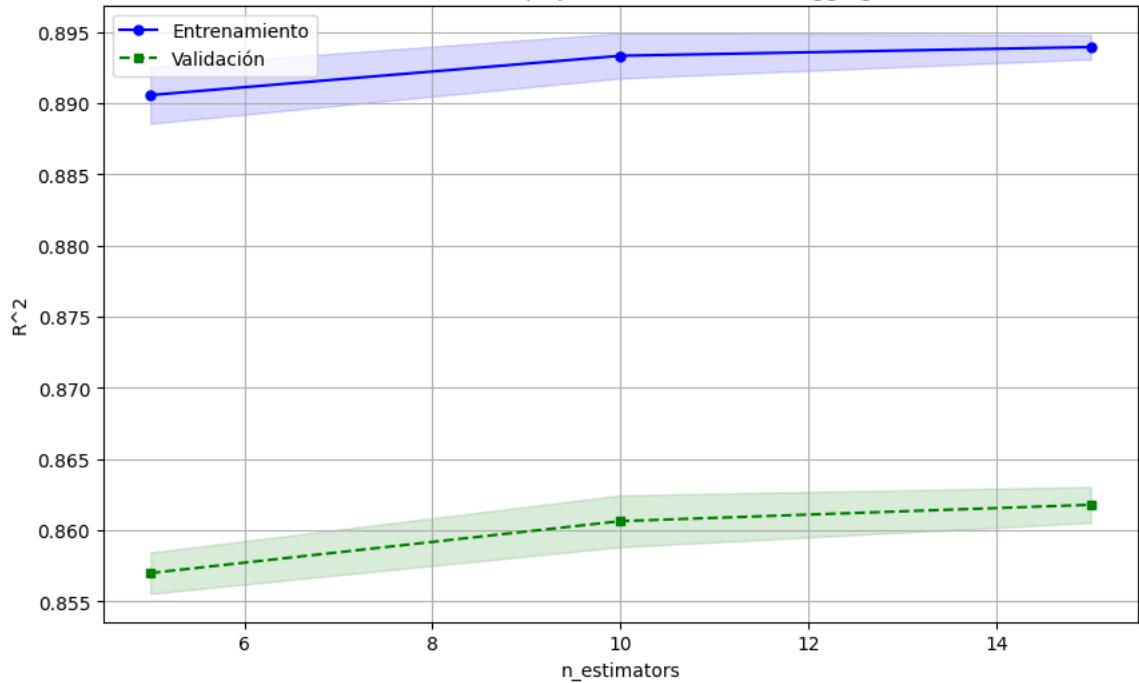
# Crea un modelo de Bagging
model = BaggingRegressor(base_estimator=DecisionTreeRegressor(**best_params))

# Calcula la curva de validación
train_scores, test_scores = validation_curve(
    estimator=model,
    X=X_train,
    y=y_train,
    param_name='n_estimators', # Hiperparámetro que deseas variar
    param_range=param_range,
    cv=5, # Número de divisiones en la validación cruzada
    scoring='r2' # La métrica que deseas evaluar (R^2 en este caso)
)

# Calcula las medias y desviaciones estándar de los puntajes
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Crea el gráfico de la curva de complejidad
plt.figure(figsize=(10, 6))
plt.plot(param_range, train_mean, color='blue', marker='o', markersize=5,
         plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, alpha=0.2, color='blue')
plt.plot(param_range, test_mean, color='green', linestyle='--', marker='s',
         plt.fill_between(param_range, test_mean - test_std, test_mean + test_std, alpha=0.2, color='green')
plt.xlabel('n_estimators')
plt.ylabel('R^2')
plt.title('Curva de Complejidad del Modelo de Bagging')
```


Curva de Complejidad del Modelo de Bagging



.7.2. Bagging dataset transformado

Entrenamiento de modelos

1. Lectura del fichero

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import cross_val_score
```



```
In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Database.parquet'

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
```



```
In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion']) # Asume que 'intensidad' es la variable objetivo
y = df['intensidad']
```



```
In [ ]: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)
```



```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=99
)
```



```
In [ ]: # Define el modelo DecisionTreeRegressor
dt = DecisionTreeRegressor()

# Define los valores que deseas probar para los hiperparámetros
param_grid = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [1, 2, 3],
    'min_impurity_decrease': [0.0, 0.1, 0.2]
}

# Define la métrica que deseas utilizar para la validación cruzada (por ejemplo)
scorer = make_scorer(r2_score)

# Configurar la búsqueda de cuadrícula
grid_search = GridSearchCV(dt, param_grid, scoring=scorer, cv=5)
```

```
# Realizar la búsqueda de cuadrícula en los datos
grid_search.fit(X, y)
```

Out[]:

```
► GridSearchCV
  ► estimator: DecisionTreeRegressor
    ► DecisionTreeRegressor
```

In []:

```
# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_
```

```
# Imprimir los mejores hiperparámetros
print("Mejores hiperparámetros:")
print(best_params)
```

Mejores hiperparámetros:

```
{'max_depth': 15, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 15}
```

In []:

```
# Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros
best_model = BaggingRegressor(
```

```
  base_estimator=DecisionTreeRegressor(**best_params),
  n_estimators=100,
  random_state=99
)
```

```
# Entrenar el modelo
best_model.fit(X_train, y_train)
```

```
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

Out[]:

```
► BaggingRegressor
  ► base_estimator: DecisionTreeRegressor
    ► DecisionTreeRegressor
```

In []:

```
# Predecir los valores de y para los datos de prueba
y_pred = best_model.predict(X_test)
```

In []:

```
# Calcular el error cuadrático medio
mse = mean_squared_error(y_test, y_pred)
```

```
# Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse)
```

```
# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:", np.sqrt(mse))
```

```
# Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_score(y_test, y_pr
```

```
# Imprimir el coeficiente de determinación ajustado (R-cuadrado ajustado)
print("Coeficiente de determinación ajustado (R-cuadrado ajustado):", 1 -
```

Error cuadrático medio: 13972.297036386402

```
Raíz del error cuadrático medio: 118.20447130454247
Coeficiente de determinación (R-cuadrado): 0.8936126803311988
Coeficiente de determinación ajustado (R-cuadrado ajustado): 0.8935860531
586897
```

```
In [ ]: # Validación cruzada el mejor modelo
scores = cross_val_score(best_model, X, y, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
Resultados de la validación cruzada:
[0.81024103 0.76637897 0.8745404  0.83247653 0.83422596]
Media de los resultados de la validación cruzada:
0.8235725774013712
Desviación estándar de los resultados de la validación cruzada:
0.03532515711341829
Precisión de la validación cruzada:
R2: 0.82 (+/- 0.07)
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())
```

```

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

```

```

c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn()

Resultados de la validación cruzada:
[0.89271334 0.89644433 0.89059661 0.89195659 0.89157564]
Media de los resultados de la validación cruzada:
0.8926573024925897
Desviación estándar de los resultados de la validación cruzada:
0.0020125663165106275
Precisión de la validación cruzada:
R2: 0.89 (+/- 0.00)

```

```

In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_test, y_test, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

```

```

c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\levaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(

```

```
c:\Users\evaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\evaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
c:\Users\evaro\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed
to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
Resultados de la validación cruzada:
[0.87446325 0.87524811 0.8732858  0.86923629 0.87665772]
Media de los resultados de la validación cruzada:
0.873778232000104
Desviación estándar de los resultados de la validación cruzada:
0.0025216262992559337
Precisión de la validación cruzada:
R2: 0.87 (+/- 0.01)
```

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

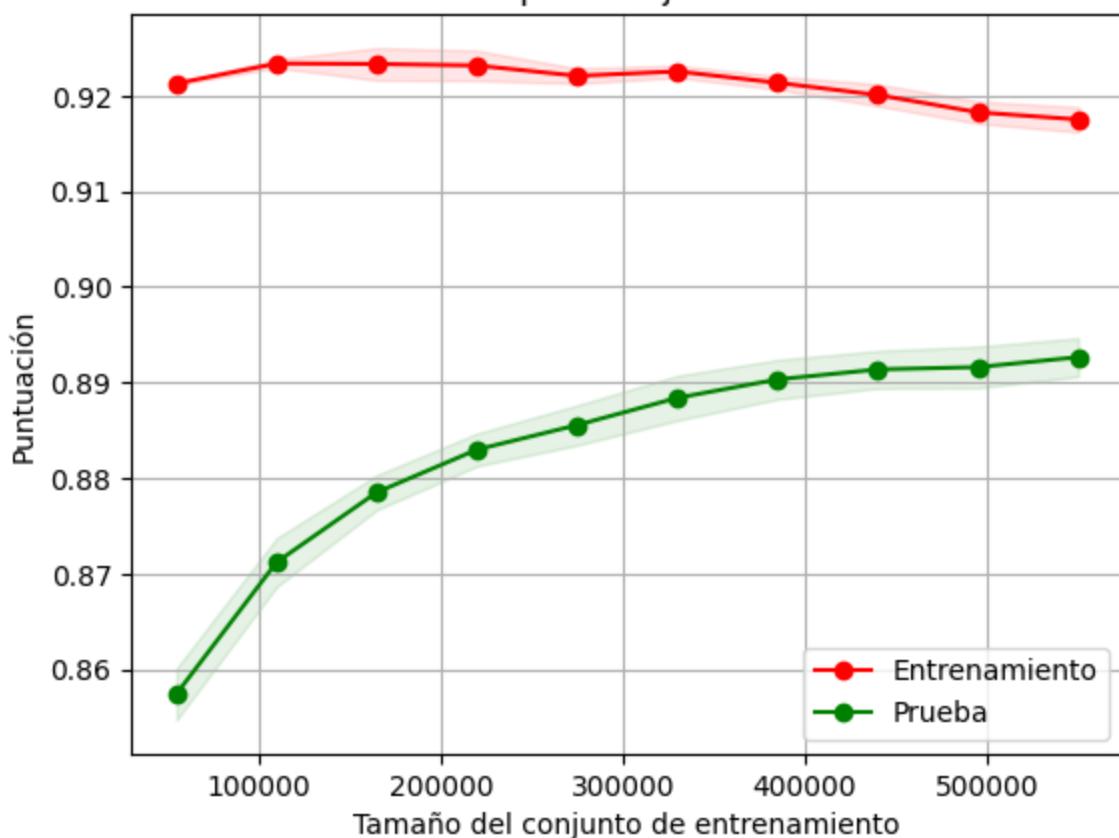
def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5), **kwargs):
    plt.figure()
    plt.title(title)
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Puntuación")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                    train_scores_mean + train_scores_std, alpha=0.1, color="b")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                    test_scores_mean + test_scores_std, alpha=0.1, color="r")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="b",
             label="Entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="r",
             label="Prueba")

    plt.legend(loc="best")
    return plt

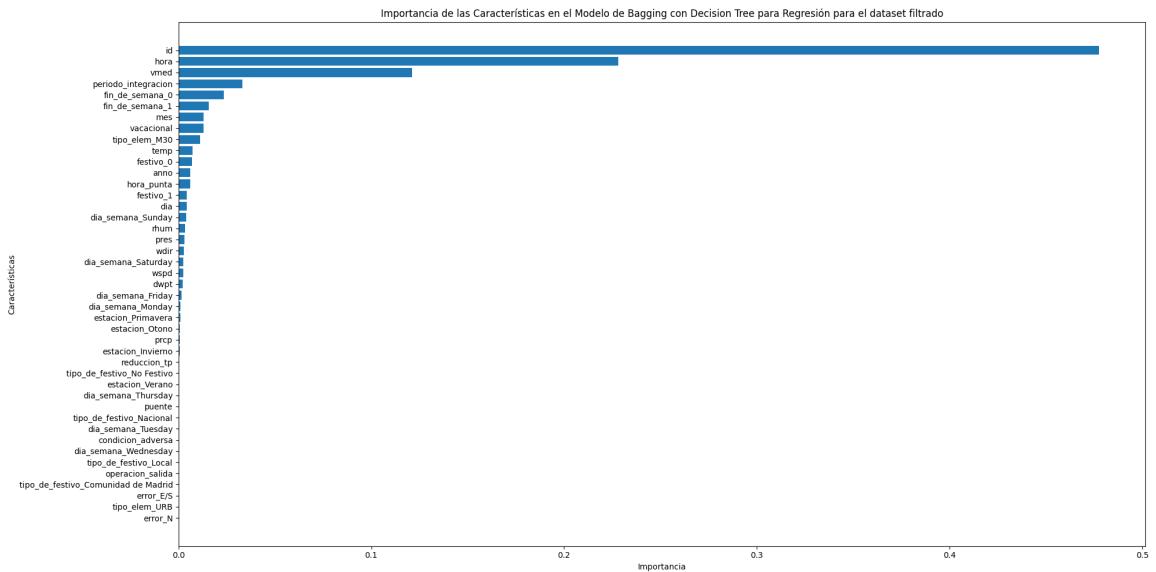
plt = plot_learning_curve(best_model, "Curva de aprendizaje Bagging", X_t)
```

Curva de aprendizaje DecisionTree



```
In [ ]: # Obtén la importancia de las características  
feature_importance = best_model.estimators_[0].feature_importances_
```

```
In [ ]: # Ordenar las características por importancia de mayor a menor  
sorted_idx = np.argsort(feature_importance)  
  
# Crear el gráfico de barras horizontales  
plt.figure(figsize=(20, 10)) # Ajusta el tamaño del gráfico según sea necesario  
plt.barh(range(len(feature_importance)), feature_importance[sorted_idx])  
plt.xlabel('Importancia')  
plt.ylabel('Características')  
plt.title('Importancia de las Características en el Modelo de Bagging con DecisionTree')  
plt.yticks(range(len(feature_importance)), [X.columns[i] for i in sorted_idx])  
plt.tight_layout()  
  
# Mostrar el gráfico  
plt.show()
```



```
In [ ]: # Guardar el modelo
import joblib

joblib.dump(best_model, 'C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Resultados/bagging_transformadas.pkl')

Out[ ]: ['C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Resultados/bagging_transformadas.pkl']

In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import validation_curve

# Define el rango de valores de n_estimators que deseas probar
param_range = [5, 10, 15]

# Crea un modelo de Bagging
model = BaggingRegressor(base_estimator=DecisionTreeRegressor(**best_params))

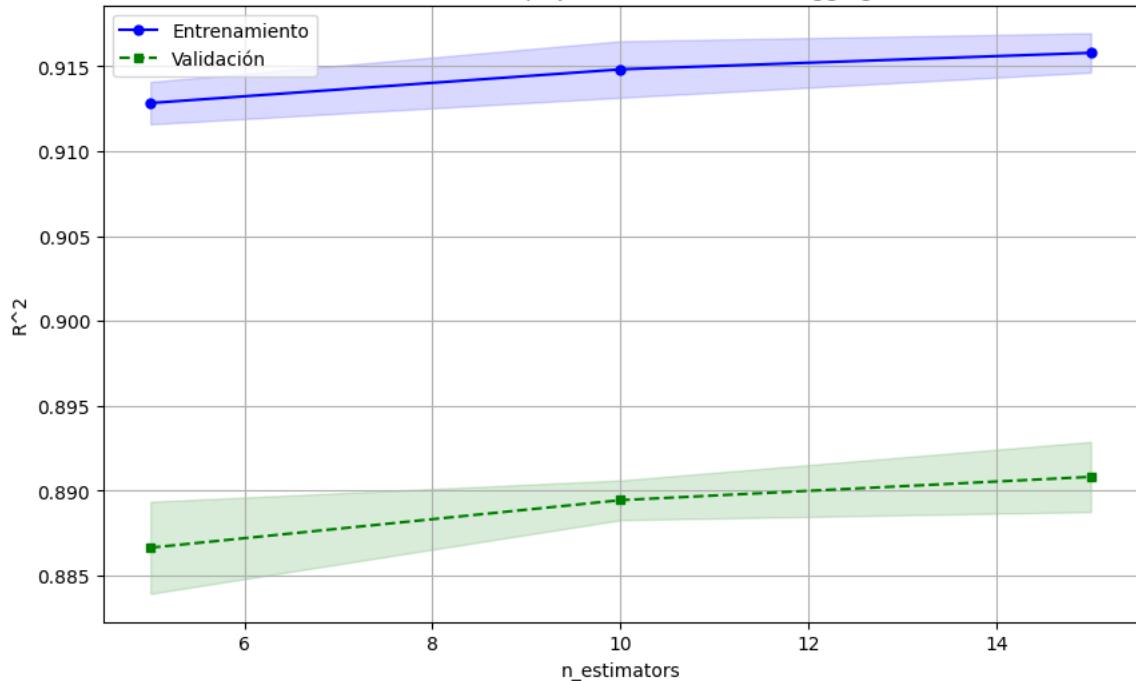
# Calcula la curva de validación
train_scores, test_scores = validation_curve(
    estimator=model,
    X=X_train,
    y=y_train,
    param_name='n_estimators', # Hiperparámetro que deseas variar
    param_range=param_range,
    cv=5, # Número de divisiones en la validación cruzada
    scoring='r2' # La métrica que deseas evaluar (R^2 en este caso)
)

# Calcula las medias y desviaciones estándar de los puntajes
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Crea el gráfico de la curva de complejidad
plt.figure(figsize=(10, 6))
plt.plot(param_range, train_mean, color='blue', marker='o', markersize=5,
         plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, alpha=0.2, color='blue')
plt.plot(param_range, test_mean, color='green', linestyle='--', marker='s',
         plt.fill_between(param_range, test_mean - test_std, test_mean + test_std, alpha=0.2, color='green')
plt.xlabel('n_estimators')
plt.ylabel('R^2')
plt.title('Curva de Complejidad del Modelo de Bagging')
```

```
plt.legend()  
plt.grid()  
plt.show()
```

Curva de Complejidad del Modelo de Bagging



In []:

In []:

In []:

.7.3. Decision Tree dataset original

Entrenamiento de modelos

1. Lectura del fichero

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer
from sklearn.metrics import r2_score
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

```
In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/nebur/Desktop/TFM/dataset_final.parquet'

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
```

```
In [ ]: df.columns
```

```
Out[ ]: Index(['id', 'tipo_elem', 'intensidad', 'ocupacion', 'carga', 'vmed',
       'periodo_integracion', 'error', 'fin_de_semana', 'festivo',
       'tipo_de_festivo', 'temp', 'dwpt', 'rhum', 'prcp', 'wdir', 'wspd',
       'pres', 'anno', 'mes', 'dia', 'hora', 'estacion', 'dia_semana',
       'hora_punta', 'condicion_adversa', 'reduccion_tp', 'puente',
       'vacacional', 'operacion_salida'],
      dtype='object')
```

```
In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion','vmed','error','t
                     'hora_punta','condicion_adversa','reduccion_tp','pue
y = df['intensidad']
```

```
In [ ]: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)
```

```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: # Define el modelo DecisionTreeRegressor
dt = DecisionTreeRegressor()

# Define los valores que deseas probar para los hiperparámetros
param_grid = {
```

```

'max_depth': [5, 10, 15],
'min_samples_split': [5, 10, 15],
'min_samples_leaf': [1, 2, 3],
'min_impurity_decrease': [0.0, 0.1, 0.2]
}

# Define la métrica que deseas utilizar para la validación cruzada (por ejemplo)
scorer = make_scorer(r2_score)

# Configurar la búsqueda de cuadrícula
grid_search = GridSearchCV(dt, param_grid, scoring=scorer, cv=5)

# Realizar la búsqueda en cuadrícula
grid_search.fit(X, y) # Asegúrate de tener X (variables explicativas) e y

# Obtener los mejores hiperparámetros y el mejor estimador
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

```

```
if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any()
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
```

```
In [ ]: # Imprimir los mejores hiperparámetros
        print("Mejores hiperparámetros:")
        print(best_params)
```

Mejores hiperparámetros:
{'max_depth': 15, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 15}

```
In [ ]: # Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros  
best_model = DecisionTreeRegressor(**best_params,  
                                    random_state=99)  
  
# Entrenar el modelo  
best_model.fit(X_train, y_train)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
 3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
DecisionTreeRegressor(max_depth=15, min_samples_leaf=3, min_samples_split
```

```
Out[ ]: =15,
         random_state=99)

In [ ]: # Predecir los valores de y para los datos de prueba
y_pred = best_model.predict(X_test)

C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():

In [ ]: # Calcular el error cuadrático medio
mse = mean_squared_error(y_test, y_pred)

# Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse)

# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:", np.sqrt(mse))

# Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_score(y_test, y_pr

# Imprimir el coeficiente de determinación ajustado (R-cuadrado ajustado)
print("Coeficiente de determinación ajustado (R-cuadrado ajustado):", 1 -

Error cuadrático medio: 19193.966693917988
Raíz del error cuadrático medio: 138.54229207688888
Coeficiente de determinación (R-cuadrado): 0.853854046685348
Coeficiente de determinación ajustado (R-cuadrado ajustado): 0.8538387371
208411

In [ ]: # Validación cruzada el mejor modelo
scores = cross_val_score(best_model, X, y, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
```

```

print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():

Resultados de la validación cruzada:
[0.8517669 0.84598173 0.84935357 0.84841965 0.84711323]
Media de los resultados de la validación cruzada:
0.8485270134811149
Desviación estándar de los resultados de la validación cruzada:
0.0019833716254069145
Precisión de la validación cruzada:
R2: 0.85 (+/- 0.00)

```

In []: # Realizar la validación cruzada con el mejor modelo y la métrica especificada

```

scores = cross_val_score(best_model, X_test, y_test, cv=5, scoring='r2')

```

```

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

```

```

C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():

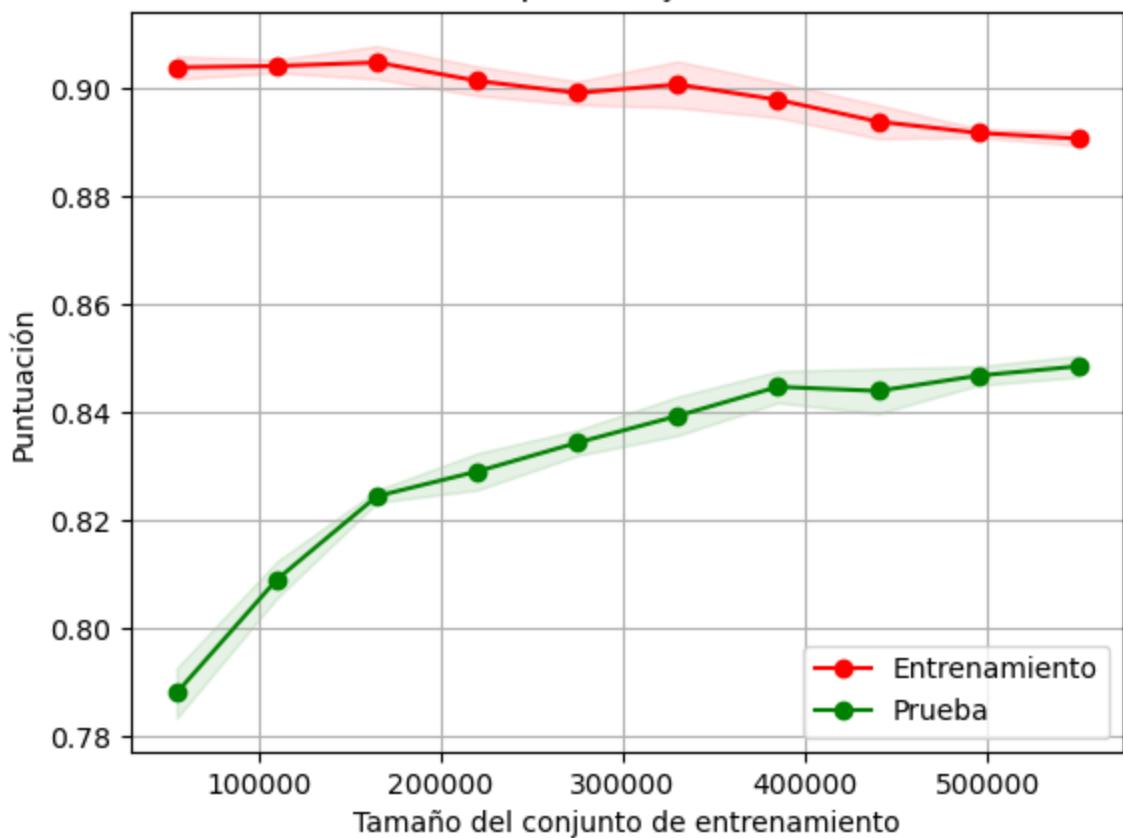
Resultados de la validación cruzada:
[0.8116417  0.81837988 0.8152229  0.81397999 0.82678903]

```

```
Media de los resultados de la validación cruzada:  
0.8172026980533481  
Desviación estándar de los resultados de la validación cruzada:  
0.0052633382609536565  
Precisión de la validación cruzada:  
R2: 0.82 (+/- 0.01)  
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future  
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():
```

```
In [ ]: import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model_selection import learning_curve  
  
def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train  
plt.figure()  
plt.title(title)  
plt.xlabel("Tamaño del conjunto de entrenamiento")  
plt.ylabel("Puntuación")  
train_sizes, train_scores, test_scores = learning_curve(  
    estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)  
train_scores_mean = np.mean(train_scores, axis=1)  
train_scores_std = np.std(train_scores, axis=1)  
test_scores_mean = np.mean(test_scores, axis=1)  
test_scores_std = np.std(test_scores, axis=1)  
plt.grid()  
  
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,  
                 train_scores_mean + train_scores_std, alpha=0.1, col  
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,  
                 test_scores_mean + test_scores_std, alpha=0.1, color  
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",  
         label="Entrenamiento")  
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",  
         label="Prueba")  
  
plt.legend(loc="best")  
return plt  
  
plt = plot_learning_curve(best_model, "Curva de aprendizaje DecisionTree")
```

Curva de aprendizaje DecisionTree



In []:

In []:

In []:

In []:

.7.4. Decision Tree dataset transformado

Entrenamiento de modelos

1. Lectura del fichero

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
from sklearn.ensemble import BaggingRegressor
```



```
In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/nebur/Desktop/TFM/dataset_final_mejtrans

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
```



```
In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion', 'vmed', 'error', 't
                     'hora_punta', 'condicion_adversa', 'reduccion_tp', 'pue
y = df['intensidad']
```



```
In [ ]: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)
```



```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=99
)
```



```
In [ ]: # Define el modelo DecisionTreeRegressor
dt = DecisionTreeRegressor()

# Define los valores que deseas probar para los hiperparámetros
param_grid = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [1, 2, 3],
    'min_impurity_decrease': [0.0, 0.1, 0.2]
}

# Define la métrica que deseas utilizar para la validación cruzada (por e
scorer = make_scorer(r2_score)

# Configurar la búsqueda de cuadrícula
grid_search = GridSearchCV(dt, param_grid, scoring=scorer, cv=5)
```



```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
```

```
In [ ]: # Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros  
best_model = DecisionTreeRegressor(**best_params,  
                                    random_state=99)  
  
# Entrenar el modelo  
best_model.fit(X_train, y_train)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
 3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
      if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
```

```
Out[ ]: DecisionTreeRegressor(max_depth=15, min_samples_leaf=3, min_samples_split=15, random_state=99)
```

```
In [ ]: # Predecir los valores de y para los datos de prueba  
y_pred = best_model.predict(X_test)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
 3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
```

```
In [ ]: # Calcular el error cuadrático medio
mse = mean_squared_error(y_test, y_pred)

# Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse)

# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:",
```

```
# Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_score(y_test, y_pr

# Imprimir el coeficiente de determinación ajustado (R-cuadrado ajustado)
print("Coeficiente de determinación ajustado (R-cuadrado ajustado):", 1 -
```

Error cuadrático medio: 19005.081986499605
Raíz del error cuadrático medio: 137.85892059094184
Coeficiente de determinación (R-cuadrado): 0.8552922452647467
Coeficiente de determinación ajustado (R-cuadrado ajustado): 0.8552770863591764

```
In [ ]: # Validación cruzada el mejor modelo
scores = cross_val_score(best_model, X, y, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

# Guardar el modelo
#joblib.dump(best_model, 'C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Mode
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
```

```
():  
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future  
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():  
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future  
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():  
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future  
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():  
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future  
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():  
Resultados de la validación cruzada:  
[0.70189655 0.62702363 0.77797014 0.75907859 0.71426023]  
Media de los resultados de la validación cruzada:  
0.716045828281345  
Desviación estándar de los resultados de la validación cruzada:  
0.05256383020283593  
Precisión de la validación cruzada:  
R2: 0.72 (+/- 0.11)
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada  
scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='r2')  
  
# Imprimir los resultados de la validación cruzada  
print("Resultados de la validación cruzada:")  
print(scores)  
  
# Imprimir la media de los resultados de la validación cruzada  
print("Media de los resultados de la validación cruzada:")  
print(scores.mean())  
  
# Imprimir la desviación estándar de los resultados de la validación cruzada  
print("Desviación estándar de los resultados de la validación cruzada:")  
print(scores.std())  
  
# Imprimir la precisión de la validación cruzada  
print("Precisión de la validación cruzada:")  
print("R2: %.2f (+/- %.2f)" % (scores.mean(), scores.std() * 2))  
  
# Guardar el modelo  
joblib.dump(best_model, 'C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Modelo')
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future  
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():  
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future  
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():  
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62  
3: FutureWarning: is_sparse is deprecated and will be removed in a future
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_test, y_test, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
```



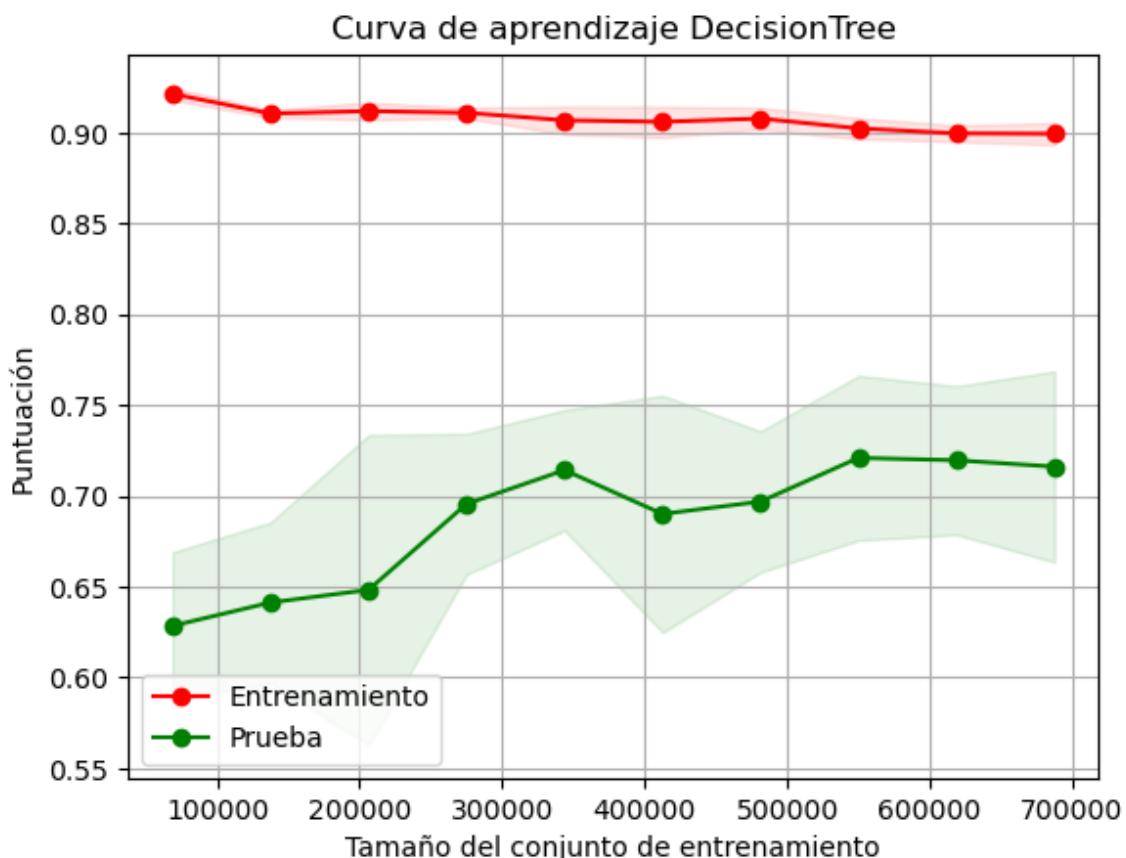
```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Puntuación")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Prueba")

    plt.legend(loc="best")
    return plt

plt = plot_learning_curve(best_model, "Curva de aprendizaje DecisionTree")
```



```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
```

```

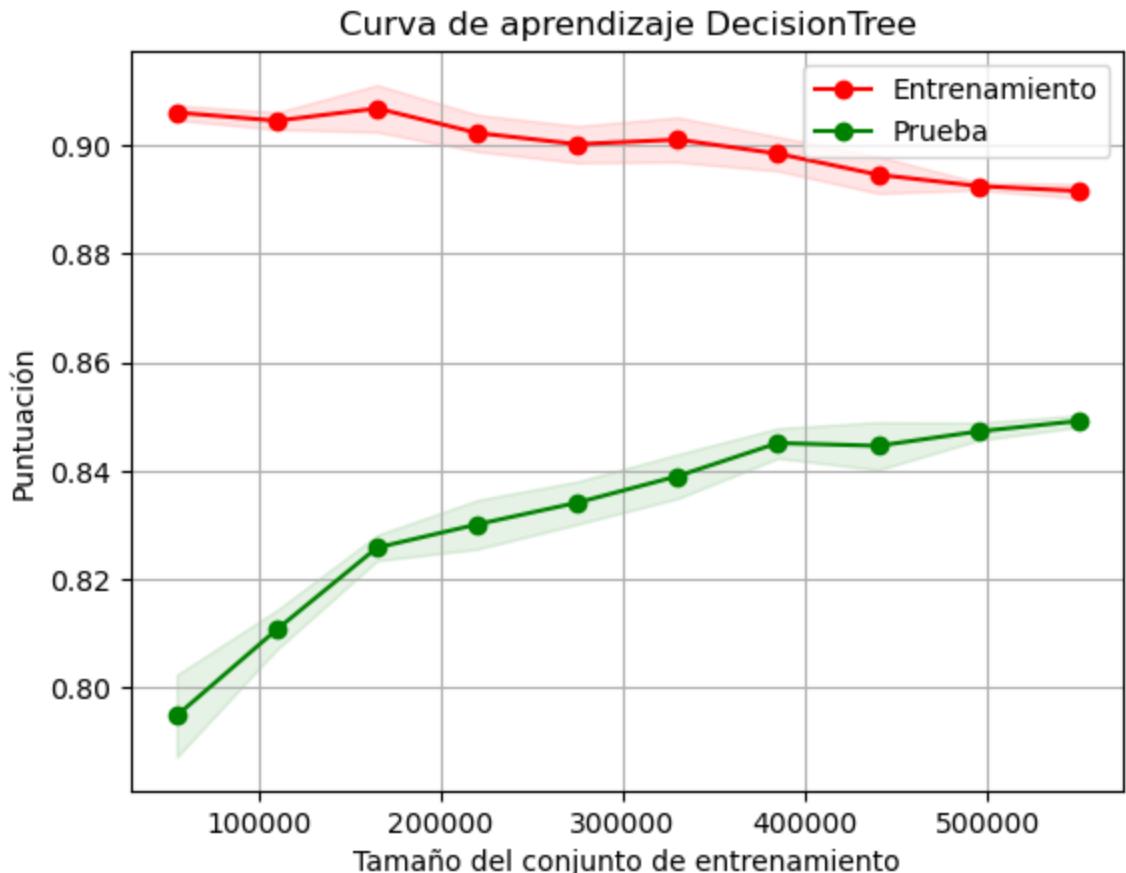
def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train_sizes=[100000, 200000, 300000, 400000, 500000]):
    plt.figure()
    plt.title(title)
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Puntuación")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                    train_scores_mean + train_scores_std, alpha=0.1, color="red")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                    test_scores_mean + test_scores_std, alpha=0.1, color="green")
    plt.plot(train_sizes, train_scores_mean, 'o--', color="red",
             label="Entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o--', color="green",
             label="Prueba")

    plt.legend(loc="best")
    return plt

plt = plot_learning_curve(best_model, "Curva de aprendizaje DecisionTree")

```



```

In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train_sizes=[100000, 200000, 300000, 400000, 500000]):
    plt.figure()
    plt.title(title)

```

```

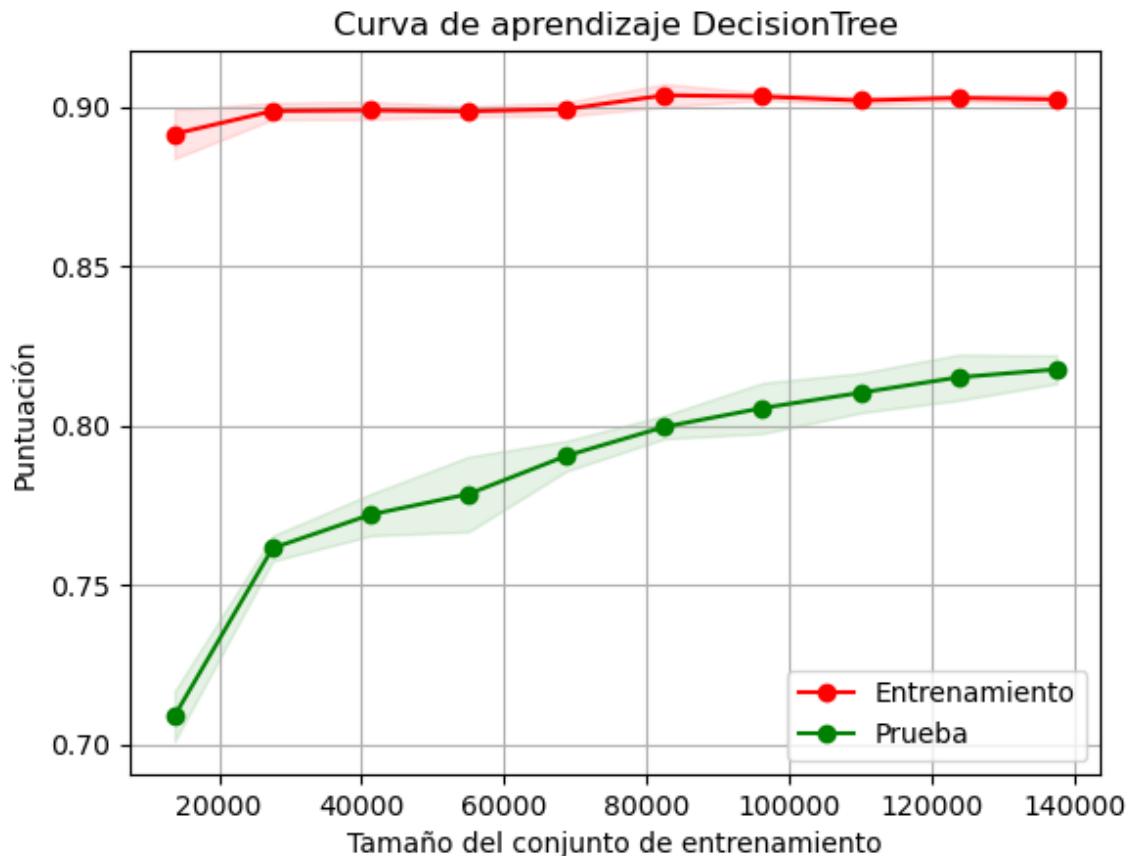
plt.xlabel("Tamaño del conjunto de entrenamiento")
plt.ylabel("Puntuación")
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Entrenamiento")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Prueba")

plt.legend(loc="best")
return plt

```

plt = plot_learning_curve(best_model, "Curva de aprendizaje DecisionTree")



In []:

In []:

In []:

In []:

.7.5. XGBoost dataset original

Entrenamiento de modelos

1. Lectura del fichero

```
In [ ]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import cross_val_score
```

```
In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/Database.parquet'

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
```

```
In [ ]: df.columns
```

```
Out[ ]: Index(['id', 'tipo_elem', 'intensidad', 'ocupacion', 'carga', 'vmed',
       'periodo_integracion', 'error', 'fin_de_semana', 'festivo',
       'tipo_de_festivo', 'temp', 'dwpt', 'rhum', 'prcp', 'wdir', 'wspd',
       'pres', 'anno', 'mes', 'dia', 'hora', 'estacion', 'dia_semana',
       'hora_punta', 'condicion_adversa', 'reduccion_tp', 'puente',
       'vacacional', 'operacion_salida'],
      dtype='object')
```

```
In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion', 'vmed', 'tipo_de_festivo',
                      'condicion_adversa', 'puente', 'operacion_salida', 'vacacional'])
y = df['intensidad']
```

```
In [ ]: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)
```

```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: # Define el modelo DecisionTreeRegressor
xgb = XGBRegressor()

# Define los valores que deseas probar para los hiperparámetros
param_grid = {
```

```

        'max_depth': [5, 10, 15],
        'min_child_weight': [1, 3, 5],
        'colsample_bytree': [0.8, 0.9, 1.0],
        'learning_rate': [0.01, 0.1, 0.2]
    }

# Define la métrica que deseas utilizar para la validación cruzada (por e
scorer = make_scorer(r2_score)

# Configurar la búsqueda de cuadrícula
grid_search = GridSearchCV(xgb, param_grid, scoring=scorer, cv=5)

# Realizar la búsqueda de cuadrícula en los datos
grid_search.fit(X, y)

```

Out[]:

```

►      GridSearchCV
►estimator: XGBRegressor
    ►XGBRegressor

```

In []:

```

# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_

# Imprimir los mejores hiperparámetros
print("Mejores hiperparámetros:")
print(best_params)

```

Mejores hiperparámetros:
{'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 10, 'min_chi
ld_weight': 3}

In []:

```

# Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros
best_model = XGBRegressor(**best_params,
    n_estimators=100,
    random_state=99
)

# Entrenar el modelo
best_model.fit(X_train, y_train)

```

Out[]:

```

▼          XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
            colsample_bylevel=None, colsample_bynode=None,
            colsample_bytree=0.8, device=None, early_stopping_r  
ounds=None,
            enable_categorical=False, eval_metric=None, feature  
_types=None,
            gamma=None, grow_policy=None, importance_type=None,
            interaction_constraints=None, learning_rate=0.1, ma  
x_bin=None,
            max_cat_threshold=None, max_cat_to_onehot=None,
            max_delta_step=None, max_depth=10, max_leaves=None,
            max_leaves=None)

```

In []:

```

# Predecir los valores de y para los datos de prueba
y_pred = best_model.predict(X_test)

```

In []:

```

# Calcular el error cuadrático medio
mse = mean_squared_error(y_test, y_pred)

```

```

# Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse)

# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:", np.sqrt(mse))

# Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_score(y_test, y_pr

# Imprimir el coeficiente de determinación ajustado (R-cuadrado ajustado)
print("Coeficiente de determinación ajustado (R-cuadrado ajustado):", 1 -
```

Error cuadrático medio: 15978.124083998358
Raíz del error cuadrático medio: 126.40460467877884
Coeficiente de determinación (R-cuadrado): 0.878339990181619
Coeficiente de determinación ajustado (R-cuadrado ajustado): 0.8783229968790015

```

In [ ]: # Validación cruzada el mejor modelo
scores = cross_val_score(best_model, X, y, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %.2f (+/- %.2f)" % (scores.mean(), scores.std() * 2))
```

Resultados de la validación cruzada:
[0.76774559 0.78654708 0.81893263 0.8176822 0.79734293]
Media de los resultados de la validación cruzada:
0.7976500870500629
Desviación estándar de los resultados de la validación cruzada:
0.019348810182080466
Precisión de la validación cruzada:
R2: 0.80 (+/- 0.04)

```

In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring='r2'

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
```

```
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Resultados de la validación cruzada:
[0.87514001 0.87659719 0.87316214 0.87405437 0.87296915]
Media de los resultados de la validación cruzada:
0.8743845732438359
Desviación estándar de los resultados de la validación cruzada:
0.0013470829135409426
Precisión de la validación cruzada:
R2: 0.87 (+/- 0.00)
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
scores = cross_val_score(best_model, X_test, y_test, cv=5, scoring='r2')

# Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada:")
print(scores)

# Imprimir la media de los resultados de la validación cruzada
print("Media de los resultados de la validación cruzada:")
print(scores.mean())

# Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Resultados de la validación cruzada:
[0.86570205 0.87356814 0.86419843 0.86133397 0.87323379]
Media de los resultados de la validación cruzada:
0.8676072759143375
Desviación estándar de los resultados de la validación cruzada:
0.004935464717093291
Precisión de la validación cruzada:
R2: 0.87 (+/- 0.01)
```

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Puntuación")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                    train_scores_mean + train_scores_std, alpha=0.1, color="b")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                    test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="b",
             label="Entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Prueba")
    plt.legend(loc="best")
    plt.show()
```

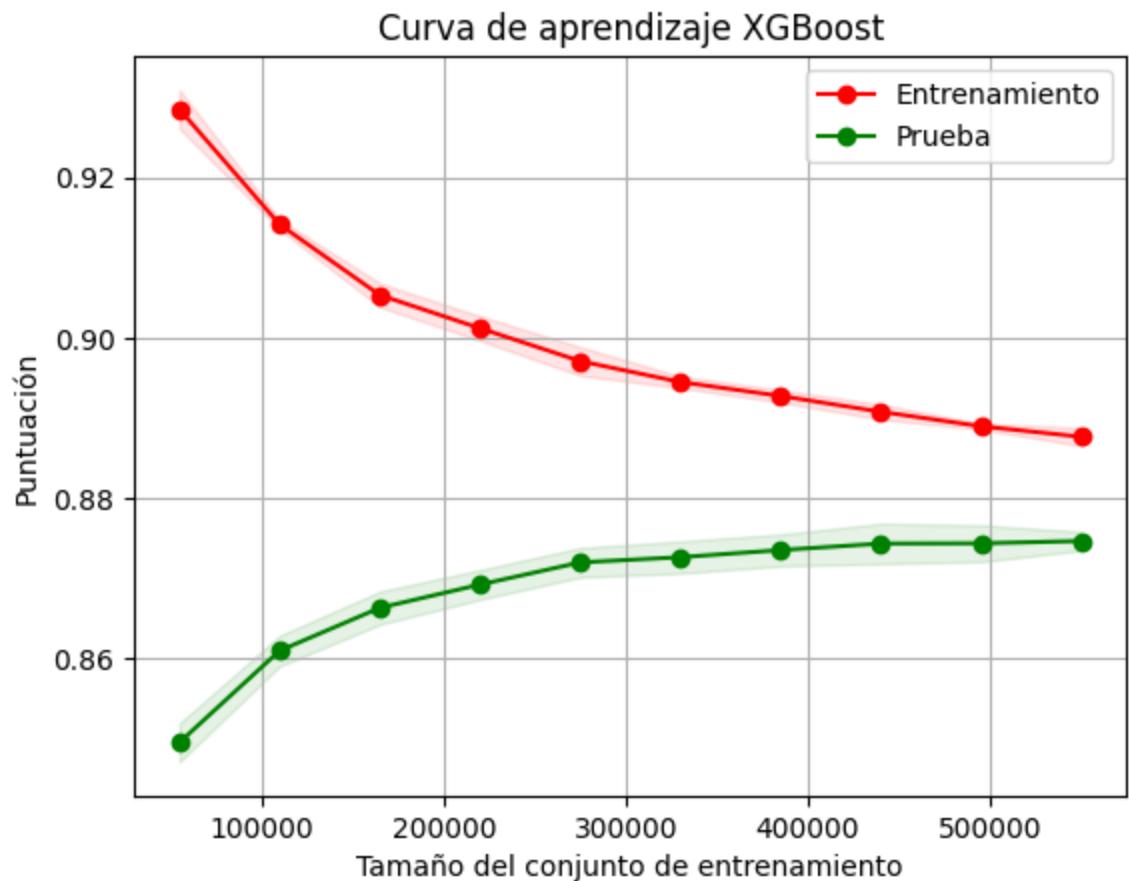
```

        label="Prueba")

plt.legend(loc="best")
return plt

plt = plot_learning_curve(best_model, "Curva de aprendizaje XGBoost", X_t

```

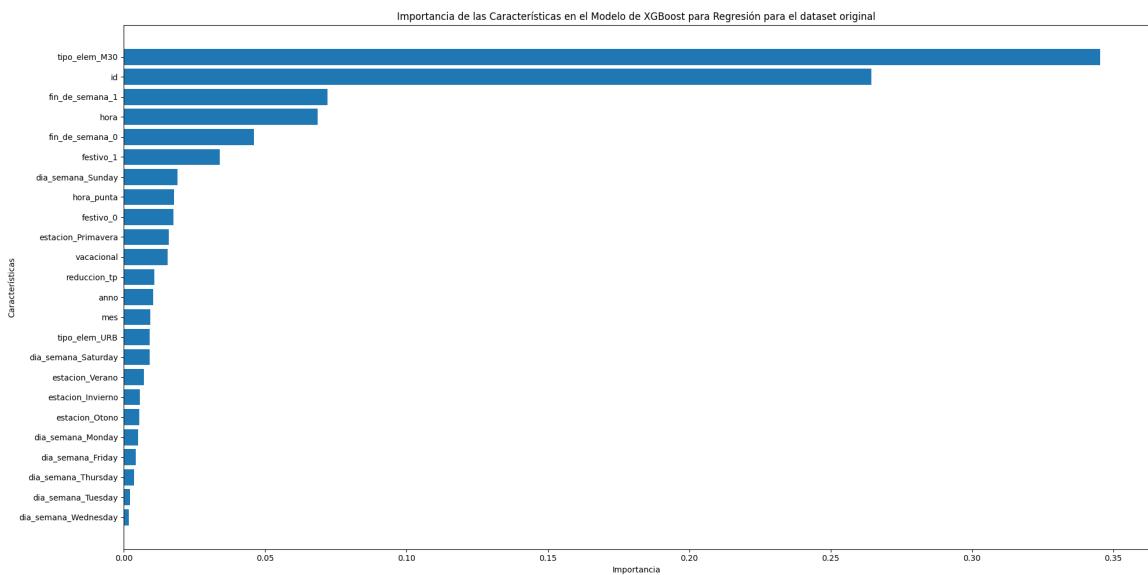


```
In [ ]: # Obtén la importancia de las características
feature_importance = best_model.feature_importances_
```

```
In [ ]: # Ordenar las características por importancia de mayor a menor
sorted_idx = np.argsort(feature_importance)

# Crear el gráfico de barras horizontales
plt.figure(figsize=(20, 10)) # Ajusta el tamaño del gráfico según sea necesario
plt.barh(range(len(feature_importance)), feature_importance[sorted_idx])
plt.xlabel('Importancia')
plt.ylabel('Características')
plt.title('Importancia de las Características en el Modelo de XGBoost para la Clasificación')
plt.yticks(range(len(feature_importance)), [X.columns[i] for i in sorted_idx])
plt.tight_layout()

# Mostrar el gráfico
plt.show()
```



```
In [ ]: # Guardar el modelo
import joblib
```

```
joblib.dump(best_model, 'C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/xgboos
```

```
Out[ ]: ['C:/Users/evaro/OneDrive/Escritorio/UCM/TFM/xgboost.pkl']
```

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import validation_curve

# Define el rango de valores de n_estimators que deseas probar
param_range = [5, 10, 15]

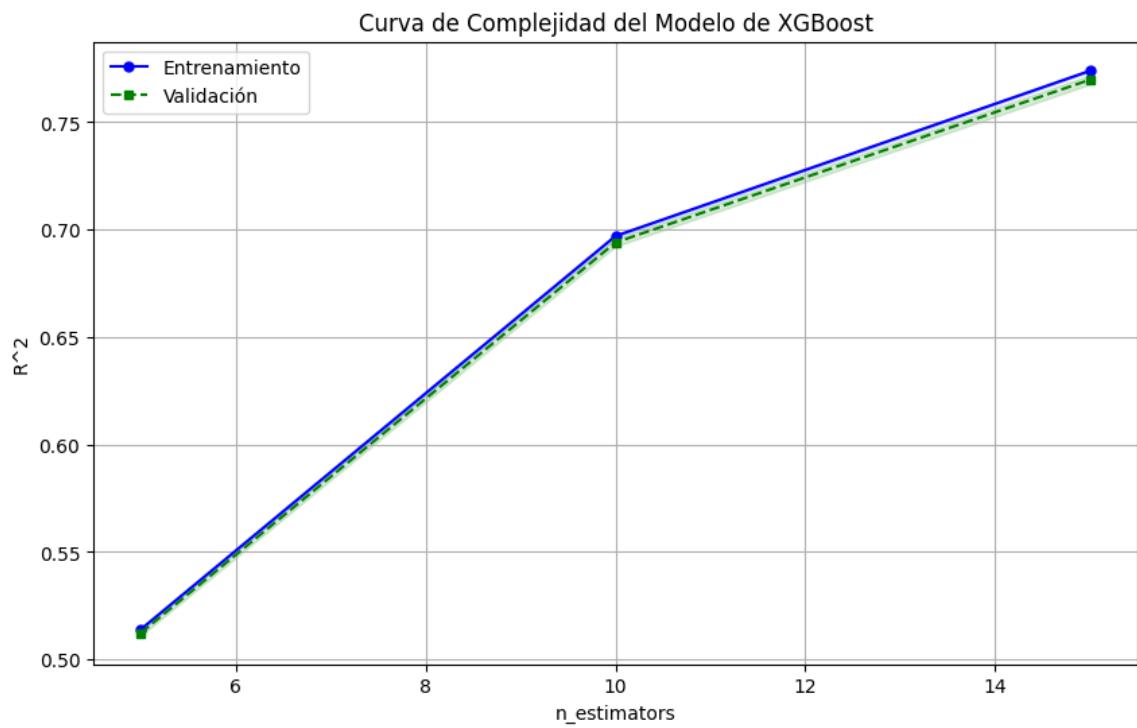
# Crea un modelo de Bagging
model = best_model

# Calcula la curva de validación
train_scores, test_scores = validation_curve(
    estimator=model,
    X=X_train,
    y=y_train,
    param_name='n_estimators', # Hiperparámetro que deseas variar
    param_range=param_range,
    cv=5, # Número de divisiones en la validación cruzada
    scoring='r2' # La métrica que deseas evaluar (R^2 en este caso)
)

# Calcula las medias y desviaciones estándar de los puntajes
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Crea el gráfico de la curva de complejidad
plt.figure(figsize=(10, 6))
plt.plot(param_range, train_mean, color='blue', marker='o', markersize=5,
         plt.fill_between(param_range, train_mean - train_std, train_mean + train_
         plt.plot(param_range, test_mean, color='green', linestyle='--', marker='s'
         plt.fill_between(param_range, test_mean - test_std, test_mean + test_std,
         plt.xlabel('n_estimators')
         plt.ylabel('R^2')
         plt.title('Curva de Complejidad del Modelo de XGBoost')
```

```
plt.legend()  
plt.grid()  
plt.show()
```



In []:

.7.6. XGBoost dataset transformado

```
In [ ]: from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import make_scorer, r2_score
import numpy as np
```

```
In [ ]: import pandas as pd

# Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/HP/Desktop/19.TFM/Datasets/Dataset_final.parquet'

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)
df.count()
```

```
Out[ ]: id                859240
tipo_elem          859240
intensidad         859240
ocupacion          859240
carga              859240
vmed               859240
periodo_integracion 859240
error              859240
fin_de_semana      859240
festivo             859240
tipo_de_festivo    859240
temp               859240
dwpt               859240
rhum               859240
prcp               859240
wdir               859240
wspd               859240
pres               859240
anno               859240
mes                859240
dia                859240
hora               859240
estacion            859240
dia_semana         859240
hora_punta          859240
condicion_adversa 859240
reduccion_tp        859240
puente              859240
vacacional          859240
operacion_salida   859240
dtype: int64
```

```
In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion']) # Asume que 'intensidad' es la variable objetivo
y = df['intensidad']
```

```
In [ ]: # Aplica get_dummies a las columnas restantes
X = pd.get_dummies(X)
```

```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, r2_score

# Define el modelo XGBoost Regressor
xgb_model = xgb.XGBRegressor()

# Define los valores que deseas probar para los hiperparámetros
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 6],
    'subsample': [0.8, 0.9, 0.7]
}

# Define la métrica que deseas utilizar para la validación cruzada (por e
scorer = make_scorer(r2_score)

# Configurar la búsqueda de cuadrícula
grid_search = GridSearchCV(xgb_model, param_grid, scoring=scorer, cv=5)

# Realizar la búsqueda de cuadrícula en los datos
grid_search.fit(X, y)
```

```
Out[ ]: GridSearchCV(cv=5,
                     estimator=XGBRegressor(base_score=None, booster=None,
                                            callbacks=None, colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=None, device=None,
                                            early_stopping_rounds=None,
                                            enable_categorical=False, eval_metric=None,
                                            feature_types=None, gamma=None,
                                            grow_policy=None, importance_type=None,
                                            interaction_constraints=None,
                                            learning_rate=None, max...
                                            max_cat_to_onehot=None, max_delta_step=None,
                                            max_depth=None, max_leaves=None,
                                            min_child_weight=None, missing=nan,
                                            monotone_constraints=None,
                                            multi_strategy=None, n_estimators=None,
                                            n_jobs=None, num_parallel_tree=None,
                                            random_state=None, ...),
                     param_grid={'learning_rate': [0.01, 0.1, 0.2],
                                'max_depth': [3, 4, 6], 'n_estimators': [50, 100, 150],
                                'subsample': [0.8, 0.9, 0.7]},
                     scoring=make_scorer(r2_score))
```

```
In [ ]: # Obtener los mejores hiperparámetros
best_params = grid_search.best_params_

# Imprimir los mejores hiperparámetros
print("Mejores hiperparámetros:")
print(best_params)
```

Mejores hiperparámetros:

```
{'learning_rate': 0.2, 'max_depth': 6, 'n_estimators': 150, 'subsample': 0.8}
```

```
In [ ]: # Extraer los mejores hiperparámetros
best_learning_rate = best_params['learning_rate']
best_max_depth = best_params['max_depth']
best_n_estimators = best_params['n_estimators']
best_subsample = best_params['subsample']

# Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros
best_model = xgb.XGBRegressor(
    learning_rate=best_learning_rate,
    max_depth=best_max_depth,
    n_estimators=best_n_estimators,
    subsample=best_subsample
)
```

```
In [ ]: # Exportar los resultados de R2 del mejor modelo
scores = cross_val_score(best_model, X_train, y_train, cv=5, scoring=scor
print("R2 del mejor modelo:")
print(np.mean(scores))

print("Resultados (R2):")
print(scores)

R2 del mejor modelo:
0.8709607381926323
Resultados (R2):
[0.87228316 0.87290367 0.8707859  0.87037686 0.86845411]
```

```
In [ ]: # Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

Desviación estándar de los resultados de la validación cruzada:
0.0015606695673329946
Precisión de la validación cruzada:
R2: 0.87 (+/- 0.00)
```

```
In [ ]: # Realizar la validación cruzada con el mejor modelo y la métrica especificada
cv_scores = cross_val_score(best_model, X_test, y_test, cv=5, scoring=scor
```

```
In [ ]: # Imprimir los resultados de la validación cruzada
print("Resultados de la validación cruzada (R2):")
print(cv_scores)

Resultados de la validación cruzada (R2):
[0.85767087 0.86322346 0.85786149 0.85558392 0.86161724]
```

```
In [ ]: # Calcula y muestra el promedio de los resultados de la validación cruzada
mean_r2 = np.mean(cv_scores)
print(f"R2 promedio en la validación cruzada: {mean_r2}")

R2 promedio en la validación cruzada: 0.8591913976568997
```

```
In [ ]: # Imprimir la desviación estándar de los resultados de la validación cruzada
print("Desviación estándar de los resultados de la validación cruzada:")
print(cv_scores.std())

# Imprimir la precisión de la validación cruzada
print("Precisión de la validación cruzada:")
print("R2: %0.2f (+/- %0.2f)" % (cv_scores.mean(), cv_scores.std() * 2))
```

Desviación estándar de los resultados de la validación cruzada:
0.0028013122551622145
Precisión de la validación cruzada:
R2: 0.86 (+/- 0.01)

```
In [ ]: from sklearn.metrics import mean_squared_error

best_model.fit(X_train, y_train)

predictions = best_model.predict(X_test)

# Calcular el RMSE
rmse = np.sqrt(mean_squared_error(y_test, predictions))

# Imprimir el valor del RMSE
print("RMSE del mejor modelo:")
print(rmse)
```

RMSE del mejor modelo:
128.8368918819876

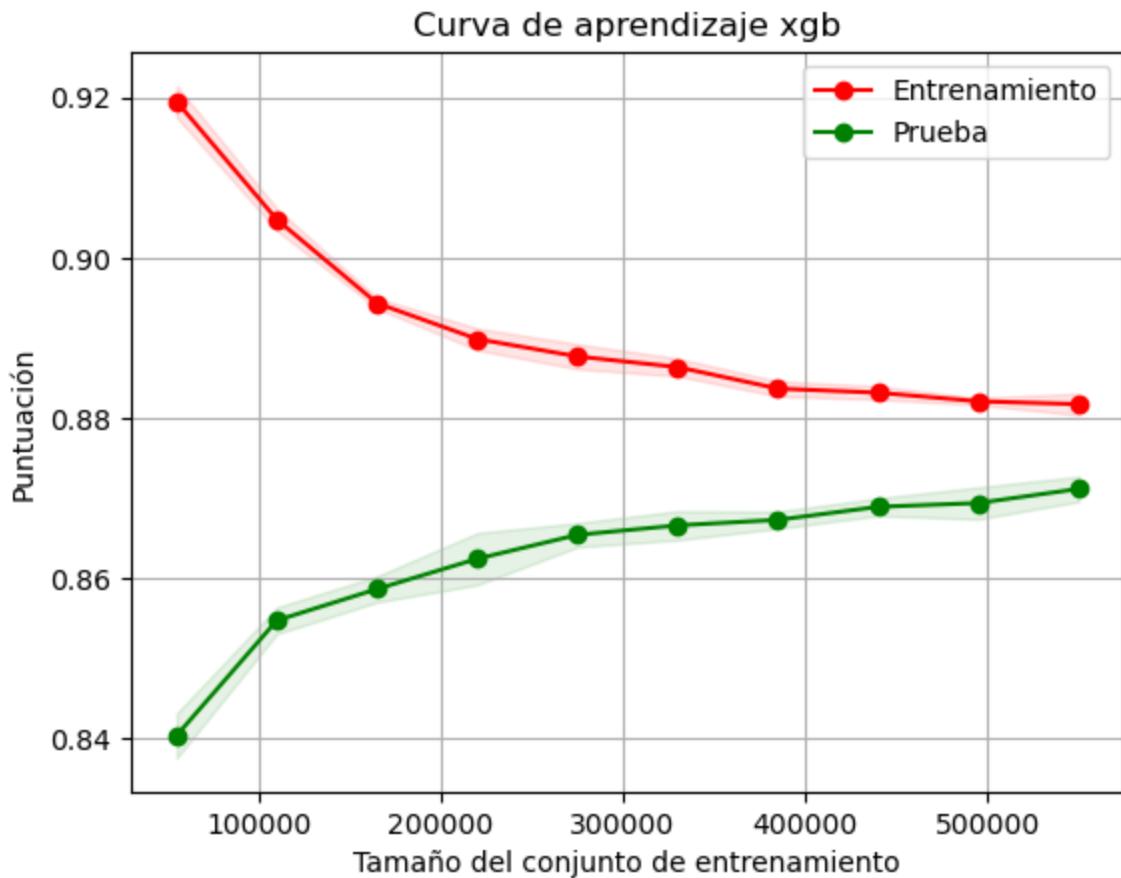
```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    plt.xlabel("Tamaño del conjunto de entrenamiento")
    plt.ylabel("Puntuación")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                    train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                    test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Entrenamiento")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Prueba")

    plt.legend(loc="best")
    return plt

plt = plot_learning_curve(best_model, "Curva de aprendizaje xgb", X_train)
```



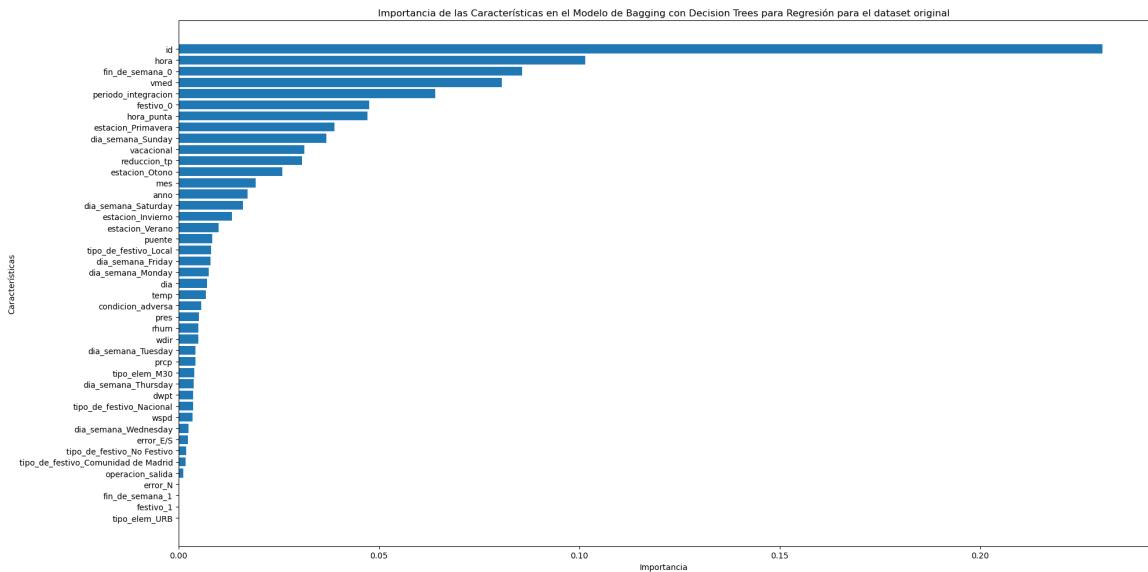
```
In [ ]: # Obtén la importancia de las características
feature_importance = best_model.feature_importances_
```



```
In [ ]: # Ordenar las características por importancia de mayor a menor
sorted_idx = np.argsort(feature_importance)

# Crear el gráfico de barras horizontales
plt.figure(figsize=(20, 10)) # Ajusta el tamaño del gráfico según sea necesario
plt.barh(range(len(feature_importance)), feature_importance[sorted_idx])
plt.xlabel('Importancia')
plt.ylabel('Características')
plt.title('Importancia de las Características en el Modelo de Bagging con XGBoost')
plt.yticks(range(len(feature_importance)), [X.columns[i] for i in sorted_idx])
plt.tight_layout()

# Mostrar el gráfico
plt.show()
```



```
In [ ]: # Guardar el modelo
import joblib

joblib.dump(best_model, 'C:/Users/HP/Desktop/19.TFM/xgb_trans.pkl')
```

```
Out[ ]: ['C:/Users/HP/Desktop/19.TFM/xgb_trans.pkl']
```

```
In [ ]:
```

.8. Script del modelo final


```
5),
        n_estimators=100, random_state=99)
```

```
In [ ]: # Predecir los valores de y para los datos de prueba
y_pred_bagging = best_model_bagging.predict(X_test)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
```

```
In [ ]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s
# Calcular las métricas de regresión para Bagging
mse_bagging = mean_squared_error(y_test, y_pred_bagging)
mae_bagging = mean_absolute_error(y_test, y_pred_bagging)
r2_bagging = r2_score(y_test, y_pred_bagging)
```

```
In [ ]: # Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse_bagging)

# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:", np.sqrt(mse_bagging))

# Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_bagging)
```

```
Error cuadrático medio: 17428.43837019701
Raíz del error cuadrático medio: 132.01681093783856
Coeficiente de determinación (R-cuadrado): 0.8672970636546325
```

```
In [ ]: y_pred_bag = best_model_bagging.predict(X)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
```

```
In [ ]: # import numpy as np
# import matplotlib.pyplot as plt
# from sklearn.model_selection import learning_curve

# def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, tra
#     plt.figure()
#     plt.title(title)
#     plt.xlabel("Tamaño del conjunto de entrenamiento")
#     plt.ylabel("Puntuación")
#     train_sizes, train_scores, test_scores = learning_curve(
#         estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
#     train_scores_mean = np.mean(train_scores, axis=1)
#     train_scores_std = np.std(train_scores, axis=1)
#     test_scores_mean = np.mean(test_scores, axis=1)
#     test_scores_std = np.std(test_scores, axis=1)
#     plt.grid()

#     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
#                     train_scores_mean + train_scores_std, alpha=0.1, c
#     plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
#                     test_scores_mean + test_scores_std, alpha=0.1, col
#     plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
#             label="Entrenamiento")
#     plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
```

```

#           label="Prueba")

#     plt.legend(loc="best")
#     return plt

# plt = plot_learning_curve(best_model_bagging, "Curva de aprendizaje Bag"

```

```

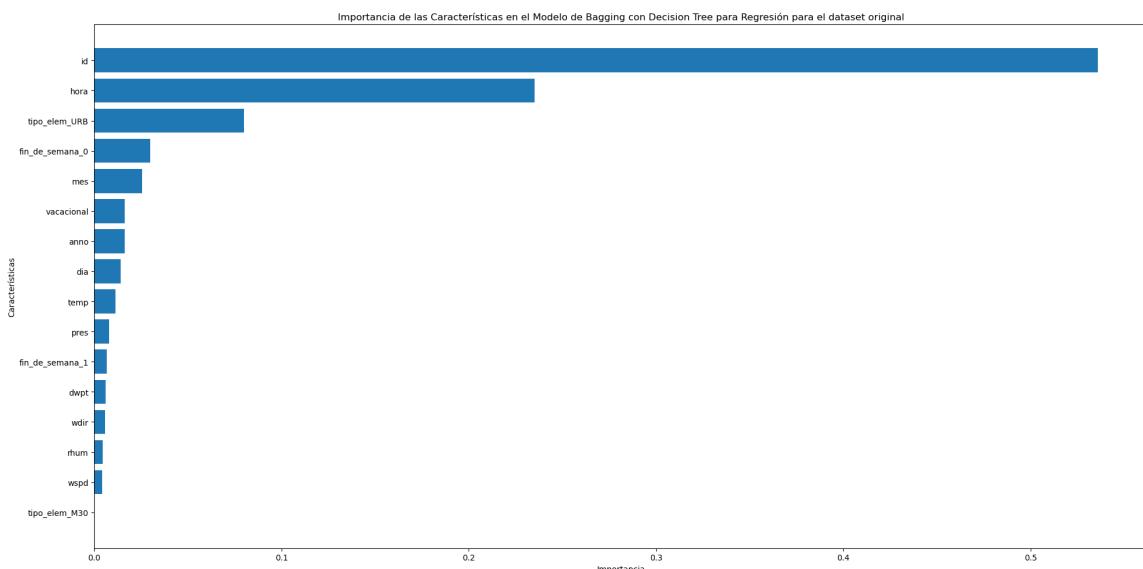
In [ ]: # Obtén la importancia de las características
feature_importance = best_model_bagging.estimators_[0].feature_importance

# Ordenar las características por importancia de mayor a menor
sorted_idx = np.argsort(feature_importance)

# Crear el gráfico de barras horizontales
plt.figure(figsize=(20, 10)) # Ajusta el tamaño del gráfico según sea necesario
plt.barh(range(len(feature_importance)), feature_importance[sorted_idx])
plt.xlabel('Importancia')
plt.ylabel('Características')
plt.title('Importancia de las Características en el Modelo de Bagging con Decision Tree para Regresión para el dataset original')
plt.yticks(range(len(feature_importance)), [X.columns[i] for i in sorted_idx])
plt.tight_layout()

# Mostrar el gráfico
plt.show()

```



```
In [ ]: # ----- XGBoost -----
```

```

In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/nebur/Desktop/TFM/dataset_final.parquet'

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)

```

```

In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion', 'vmed', 'tipo_de_condicion_adversa', 'puente', 'operacion_salida', 'p'])
y = df['intensidad']

```

```

In [ ]: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)

```

```

In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(

```

```
    X, y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: # Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros
best_params_xgb = {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 100, 'random_state': 99}

# Entrenar el modelo
best_model_xgb.fit(X_train, y_train)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:335: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if is_sparse(dtype):
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:338: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    is_categorical_dtype(dtype) or is_pa_ext_categorical_dtype(dtype)
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:384: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    if is_categorical_dtype(dtype):
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:359: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    return is_int or is_bool or is_float or is_categorical_dtype(dtype)
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:520: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if is_sparse(data):
```

```
Out[ ]: XGBRegressor(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
                      colsample_bytree=0.8, device=None, early_stopping_rounds=None,
                      enable_categorical=False, eval_metric=None, feature_types=None,
                      gamma=None, grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=0.1, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=10, max_leaves=None,
                      min_child_weight=3, missing=nan, monotone_constraints=None,
                      multi_strategy=None, n_estimators=100, n_jobs=None,
                      num_parallel_tree=None, random_state=99, ...)
```

```
In [ ]: # Predecir los valores de y para los datos de prueba
y_pred_xgb = best_model_xgb.predict(X_test)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:335: FutureWarning: is_sparse is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if is_sparse(dtype):
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:338: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
    is_categorical_dtype(dtype) or is_pa_ext_categorical_dtype(dtype)
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:384: FutureWar
```

```
ning: is_categorical_dtype is deprecated and will be removed in a future
version. Use isinstance(dtype, CategoricalDtype) instead
    if is_categorical_dtype(dtype):
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:359: FutureWar
ning: is_categorical_dtype is deprecated and will be removed in a future
version. Use isinstance(dtype, CategoricalDtype) instead
        return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```

```
In [ ]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s
# Calcular las métricas de regresión para XGBoost
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)
```

```
In [ ]: # Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse_xgb)

# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:", np.sqrt(mse_xgb))

#Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_xgb)
```

```
Error cuadrático medio: 15978.124083998358
Raíz del error cuadrático medio: 126.40460467877884
Coeficiente de determinación (R-cuadrado): 0.878339990181619
```

```
In [ ]: y_pred_xgboost = best_model_xgb.predict(X)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:335: FutureWar
ning: is_sparse is deprecated and will be removed in a future version. Ch
eck `isinstance(dtype, pd.SparseDtype)` instead.
    if is_sparse(dtype):
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:338: FutureWar
ning: is_categorical_dtype is deprecated and will be removed in a future
version. Use isinstance(dtype, CategoricalDtype) instead
    is_categorical_dtype(dtype) or is_pa_ext_categorical_dtype(dtype)
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:384: FutureWar
ning: is_categorical_dtype is deprecated and will be removed in a future
version. Use isinstance(dtype, CategoricalDtype) instead
    if is_categorical_dtype(dtype):
C:\Users\nebur\anaconda3\lib\site-packages\xgboost\data.py:359: FutureWar
ning: is_categorical_dtype is deprecated and will be removed in a future
version. Use isinstance(dtype, CategoricalDtype) instead
        return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```

```
In [ ]: # import numpy as np
# import matplotlib.pyplot as plt
# from sklearn.model_selection import learning_curve

# def plot_learning_curve(estimator, title, X, y, cv=None, n_jobs=-1, tra
#     plt.figure()
#     plt.title(title)
#     plt.xlabel("Tamaño del conjunto de entrenamiento")
#     plt.ylabel("Puntuación")
#     train_sizes, train_scores, test_scores = learning_curve(
#         estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
#     train_scores_mean = np.mean(train_scores, axis=1)
#     train_scores_std = np.std(train_scores, axis=1)
#     test_scores_mean = np.mean(test_scores, axis=1)
#     test_scores_std = np.std(test_scores, axis=1)
#     plt.grid()
```

```

#     plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
#                         train_scores_mean + train_scores_std, alpha=0.1, c
#     plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
#                         test_scores_mean + test_scores_std, alpha=0.1, col
#     plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
#             label="Entrenamiento")
#     plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
#             label="Prueba")

#     plt.legend(loc="best")
#     return plt

# plt = plot_learning_curve(best_model_xgb, "Curva de aprendizaje XGBoost"

```

```

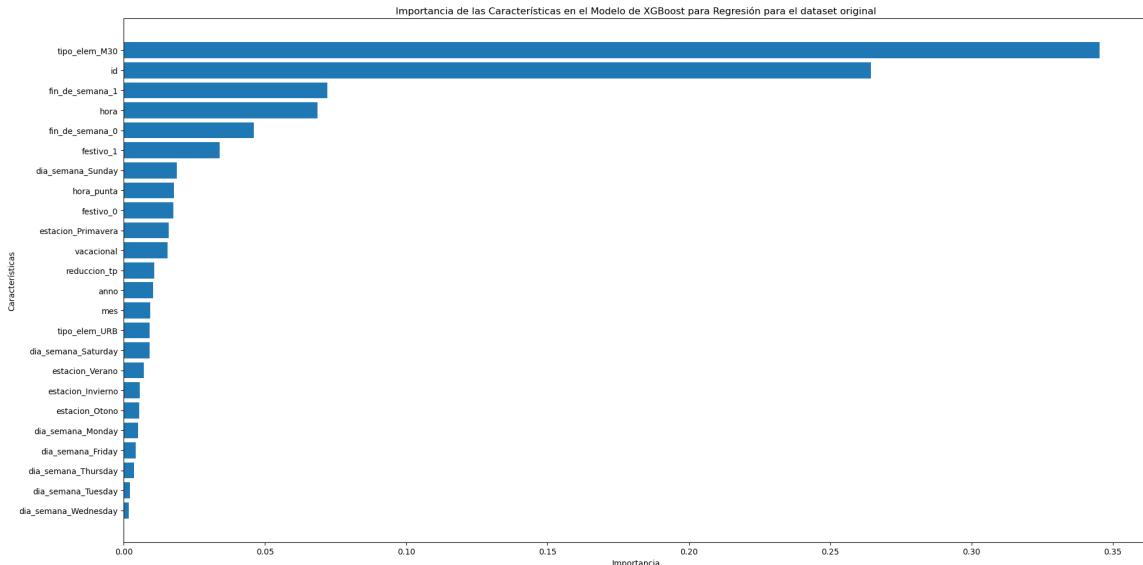
In [ ]: # Obtén la importancia de las características
feature_importance = best_model_xgb.feature_importances_

# Ordenar las características por importancia de mayor a menor
sorted_idx = np.argsort(feature_importance)

# Crear el gráfico de barras horizontales
plt.figure(figsize=(20, 10)) # Ajusta el tamaño del gráfico según sea necesario
plt.barh(range(len(feature_importance)), feature_importance[sorted_idx])
plt.xlabel('Importancia')
plt.ylabel('Características')
plt.title('Importancia de las Características en el Modelo de XGBoost para Regresión')
plt.yticks(range(len(feature_importance)), [X.columns[i] for i in sorted_idx])
plt.tight_layout()

# Mostrar el gráfico
plt.show()

```



DecisionTree

```

In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/nebur/Desktop/TFM/dataset_final.parquet'

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)

```

```

In [ ]: # Define X (características) y y (variable objetivo)
X = df.drop(columns=['intensidad', 'carga', 'ocupacion','vmed','error','temp','humedad','velocidad','presion','radiacion','humedad_relativa','temperatura_minima','temperatura_maxima','velocidad_viento','presion_atmosferica','humedad_suelo','radiacion_solar','velocidad_viento_maxima','presion_atmosferica_minima','presion_atmosferica_maxima','humedad_suelo_minima','humedad_suelo_maxima','radiacion_solar_maxima','velocidad_viento_promedio','presion_atmosferica_promedio','humedad_suelo_promedio'])
y = df['intensidad']

```

```
In [ ]: # Codifica las características categóricas utilizando one-hot encoding
X = pd.get_dummies(X)
```

```
In [ ]: # Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=99
)
```

```
In [ ]: # Definir el modelo DecisionTreeRegressor con los mejores hiperparámetros
best_params_dtr = {'max_depth': 15, 'min_impurity_decrease': 0.0, 'min_sa
best_model_dtr = DecisionTreeRegressor(**best_params_dtr,
    random_state=99
)

# Entrenar el modelo
best_model_dtr.fit(X_train, y_train)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
```

```
Out[ ]: DecisionTreeRegressor(max_depth=15, min_samples_leaf=3, min_samples_split
=15,
                               random_state=99)
```

```
In [ ]: # Predecir los valores de y para los datos de prueba
y_pred_dtr = best_model_dtr.predict(X_test)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
```

```
In [ ]: from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_s
# Calcular las métricas de regresión para XGBoost
mse_dtr = mean_squared_error(y_test, y_pred_dtr)
mae_dtr = mean_absolute_error(y_test, y_pred_dtr)
r2_dtr = r2_score(y_test, y_pred_dtr)
```

```
In [ ]: # Imprimir el error cuadrático medio
print("Error cuadrático medio:", mse_dtr)

# Imprimir la raíz del error cuadrático medio
print("Raíz del error cuadrático medio:", np.sqrt(mse_dtr))

# Imprimir el coeficiente de determinación (R-cuadrado)
print("Coeficiente de determinación (R-cuadrado):", r2_dtr)
```

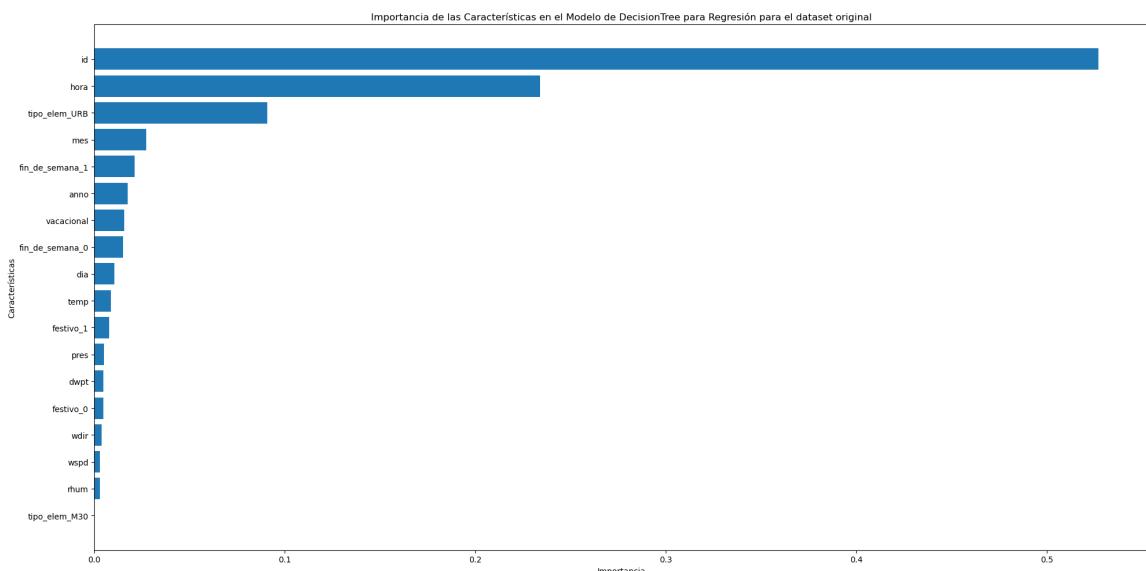
```
Error cuadrático medio: 19193.966693917988
Raíz del error cuadrático medio: 138.54229207688888
Coeficiente de determinación (R-cuadrado): 0.853854046685348
```

```
In [ ]: y_pred_dt = best_model_dtr.predict(X)
```

```
C:\Users\nebur\anaconda3\lib\site-packages\sklearn\utils\validation.py:62
3: FutureWarning: is_sparse is deprecated and will be removed in a future
```

```
version. Check `isinstance(dtype, pd.SparseDtype)` instead.  
    if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any  
():
```

```
In [ ]: # Obtén la importancia de las características  
feature_importance = best_model_dtr.feature_importances_  
  
# Ordenar las características por importancia de mayor a menor  
sorted_idx = np.argsort(feature_importance)  
  
# Crear el gráfico de barras horizontales  
plt.figure(figsize=(20, 10)) # Ajusta el tamaño del gráfico según sea necesario  
plt.barh(range(len(feature_importance)), feature_importance[sorted_idx])  
plt.xlabel('Importancia')  
plt.ylabel('Características')  
plt.title('Importancia de las Características en el Modelo de DecisionTree')  
plt.yticks(range(len(feature_importance)), [X.columns[i] for i in sorted_idx])  
plt.tight_layout()  
  
# Mostrar el gráfico  
plt.show()
```



```
In [ ]: # ----- Comparación -----
```

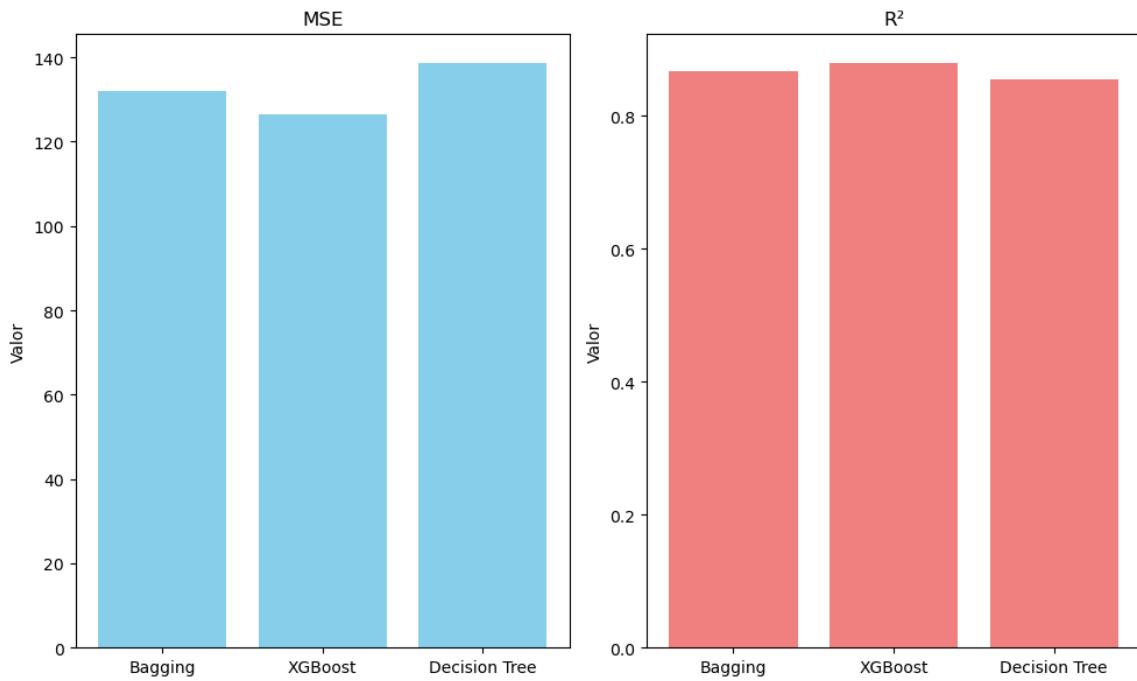
```
In [ ]: import matplotlib.pyplot as plt  
  
# Métricas de rendimiento para cada modelo  
model_names = ["Bagging", "XGBoost", "Decision Tree"]  
mse_scores = [np.sqrt(mse_bagging), np.sqrt(mse_xgb), np.sqrt(mse_dtr)]  
r2_scores = [r2_bagging, r2_xgb, r2_dtr]  
  
# Crear un gráfico comparativo  
plt.figure(figsize=(10, 6))  
  
# Gráfico de MSE  
plt.subplot(1, 2, 1)  
plt.bar(model_names, mse_scores, color='skyblue')  
plt.title("MSE")  
plt.ylabel("Valor")  
  
# Gráfico de R2  
plt.subplot(1, 2, 2)  
plt.bar(model_names, r2_scores, color='lightcoral')
```

```

plt.title("R2")
plt.ylabel("Valor")

plt.tight_layout()
plt.show()

```



```

In [ ]: # Ruta al archivo Parquet
ruta_archivo_parquet = 'C:/Users/nebur/Desktop/TFM/dataset_final.parquet'

# Lee el archivo Parquet y carga los datos en un DataFrame
df = pd.read_parquet(ruta_archivo_parquet)

```

```
In [ ]: df['y_pred_bagging'] = y_pred_bag
```

```
In [ ]: df['y_pred_xgb'] = y_pred_xgboost
```

```
In [ ]: df['y_pred_dt'] = y_pred_dt
```

```
In [ ]: df['dif_bag'] = df['y_pred_bagging'] - df['intensidad']
df['dif_xgb'] = df['y_pred_xgb'] - df['intensidad']
df['dif_dt'] = df['y_pred_dt'] - df['intensidad']
```

```
In [ ]: df
```

| | id | tipo_elem | intensidad | ocupacion | carga | vmed | periodo_integracion | error |
|---------------|-----------|------------------|-------------------|------------------|--------------|-------------|----------------------------|--------------|
| 0 | 1026 | M30 | 177.0 | 1 | 0 | 46 | | 5 N |
| 1 | 1026 | M30 | 156.0 | 0 | 0 | 48 | | 5 N |
| 2 | 1026 | M30 | 255.0 | 1 | 0 | 57 | | 5 N |
| 3 | 1026 | M30 | 330.0 | 2 | 0 | 57 | | 5 N |
| 4 | 1026 | M30 | 372.0 | 3 | 0 | 56 | | 5 N |
| ... | ... | ... | ... | ... | ... | ... | | ... |
| 859235 | 11004 | URB | 66.0 | 0 | 4 | 0 | | 14 N |
| 859236 | 11004 | URB | 70.0 | 0 | 5 | 0 | | 15 N |
| 859237 | 11004 | URB | 66.0 | 2 | 5 | 0 | | 14 N |

| | | | | | | | | | |
|--------|-------|-----|------|---|---|---|--|----|---|
| 859238 | 11004 | URB | 54.0 | 0 | 3 | 0 | | 13 | N |
| 859239 | 11004 | URB | 45.0 | 0 | 2 | 0 | | 15 | N |

859240 rows × 36 columns

```
In [ ]: ruta_archivo_csv = "C:/Users/nebur/Downloads/Distritos_Completar.xlsx"
```

```
# Lee el archivo CSV en un DataFrame de Pandas
df1 = pd.read_excel(ruta_archivo_csv)
```

```
In [ ]: datos = pd.merge(df, df1, on='id', how='inner')
```

```
In [ ]: datos = datos.drop('nombre', axis=1)
```

```
In [ ]: datos
```

```
Out[ ]:   id tipo_elem intensidad ocupacion carga vmed periodo_integracion error
```

| | | | | | | | | | |
|--------|-------|-----|-------|-----|-----|-----|--|-----|-----|
| 0 | 1026 | M30 | 177.0 | 1 | 0 | 46 | | 5 | N |
| 1 | 1026 | M30 | 156.0 | 0 | 0 | 48 | | 5 | N |
| 2 | 1026 | M30 | 255.0 | 1 | 0 | 57 | | 5 | N |
| 3 | 1026 | M30 | 330.0 | 2 | 0 | 57 | | 5 | N |
| 4 | 1026 | M30 | 372.0 | 3 | 0 | 56 | | 5 | N |
| ... | ... | ... | ... | ... | ... | ... | | ... | ... |
| 840913 | 11004 | URB | 66.0 | 0 | 4 | 0 | | 14 | N |
| 840914 | 11004 | URB | 70.0 | 0 | 5 | 0 | | 15 | N |
| 840915 | 11004 | URB | 66.0 | 2 | 5 | 0 | | 14 | N |
| 840916 | 11004 | URB | 54.0 | 0 | 3 | 0 | | 13 | N |
| 840917 | 11004 | URB | 45.0 | 0 | 2 | 0 | | 15 | N |

840918 rows × 38 columns

```
In [ ]: datos = datos.rename(columns={'Distrito_nombbre': 'Nombre_Distrito'})
```

```
In [ ]: datos
```

```
Out[ ]:   id tipo_elem intensidad ocupacion carga vmed periodo_integracion error
```

| | | | | | | | | | |
|--------|-------|-----|-------|-----|-----|-----|--|-----|-----|
| 0 | 1026 | M30 | 177.0 | 1 | 0 | 46 | | 5 | N |
| 1 | 1026 | M30 | 156.0 | 0 | 0 | 48 | | 5 | N |
| 2 | 1026 | M30 | 255.0 | 1 | 0 | 57 | | 5 | N |
| 3 | 1026 | M30 | 330.0 | 2 | 0 | 57 | | 5 | N |
| 4 | 1026 | M30 | 372.0 | 3 | 0 | 56 | | 5 | N |
| ... | ... | ... | ... | ... | ... | ... | | ... | ... |
| 840913 | 11004 | URB | 66.0 | 0 | 4 | 0 | | 14 | N |
| 840914 | 11004 | URB | 70.0 | 0 | 5 | 0 | | 15 | N |
| 840915 | 11004 | URB | 66.0 | 2 | 5 | 0 | | 14 | N |

| | | | | | | | | |
|---------------|-------|-----|------|---|---|---|----|---|
| 840916 | 11004 | URB | 54.0 | 0 | 3 | 0 | 13 | N |
| 840917 | 11004 | URB | 45.0 | 0 | 2 | 0 | 15 | N |

840918 rows × 38 columns

```
In [ ]: ruta_completa = "C:/Users/nebur/Downloads/datos.csv"  
        datos.to_csv(ruta_completa, index=False)
```

```
In [ ]:
```


.9. Scripts de la productivización del modelo

.9.1. transformaciones.py

```
import pandas as pandas
import datetime
from datetime import date, timedelta
from dateutil.relativedelta import relativedelta
from dateutil.easter import easter

# ESTACION
def calcular_estacion(dia, mes):
    if (mes == 3 and dia >= 21) or (4 <= mes <= 5) or (mes == 6 and dia <=
        20):
        return 'Primavera'
    elif (mes == 6 and dia >= 21) or (7 <= mes <= 8) or (mes == 9 and dia
        <= 20):
        return 'Verano'
    elif (mes == 9 and dia >= 21) or (10 <= mes <= 11) or (mes == 12 and
        dia <= 20):
        return 'Otono'
    else:
        return 'Invierno'

# DIAS DE LA SEMANA
def calcular_dias_semana(anno, mes, dia):
    fecha = datetime.date(anno, mes, dia)
    nombre_dia = fecha.strftime('%A')
    return nombre_dia

# HORA PUNTA
def calcular_hora_punta(mes, dia_semana, hora):
    if mes == 8: # Agosto
        if dia_semana in ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
            'Friday']:
```

```
if 12 <= hora <= 15 or hora == 19:
    return 1
elif dia_semana in ['Saturday', 'Sunday']:
    if 12 <= hora <= 14 or hora == 20:
        return 1
else: # Resto de meses
    if dia_semana in ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                      'Friday']:
        if hora == 9 or 14 <= hora <= 15 or hora == 19:
            return 1
        elif dia_semana in ['Saturday', 'Sunday']:
            if 12 <= hora <= 14 or 19 <= hora <= 21:
                return 1

    return 0

# REDUCCION TRANSPORTE PUBLICO
def calcular_reduccion_tp(hora):
    if 1 <= hora <= 6:
        return 1
    else:
        return 0

# VACACIONAL
def calcular_vacacional(anno, mes, dia):
    def is_holiday(anno, mes, dia):
        easter_date = easter(anno)
        christmas_date = date(anno, 12, 25)
        kings_day_date = date(anno, 1, 6)
        summer_start = date(anno, 7, 1) # July 1
        summer_end = date(anno, 8, 31) # August 31

        # Calculate the start and end dates of the week before Easter
        pass
```

```
week_before_easter_start = easter_date - timedelta(days=7)
week_before_easter_end = easter_date - timedelta(days=1)

input_date = date(anno, mes, dia)

# Check if the date is a holiday
return int(
    input_date == easter_date or
    (christmas_date - relativedelta(days=3)) <= input_date <=
        date(anno, 12, 31) or
    (date(anno, 1, 1) <= input_date <= kings_day_date) or
    (week_before_easter_start <= input_date <=
        week_before_easter_end) or
    (summer_start <= input_date <= summer_end)
)

return is_holiday(anno, mes, dia)

# FIN DE SEMANA
def es_fin_de_semana(fecha):
    # Convierte la fecha a un objeto de fecha si no lo es
    if not isinstance(fecha, datetime.date):
        return False

    # Obtiene el da de la semana (0 es lunes, 6 es domingo)
    dia_semana = fecha.weekday()

    # Comprueba si es sbado (5) o domingo (6)
    if dia_semana in [5, 6]:
        return True
    else:
        return False
```

.9.2. servicio_modelo.py

```
# backend.py
from flask import Flask, request, jsonify
import numpy as np
import pandas as pd
import pickle as pkl
from xgboost import XGBRegressor

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

app = Flask(__name__)

ruta_pickle = '/app/xgboost.pkl'

with open(ruta_pickle, 'rb') as archivo:
    modelo = pkl.load(archivo)

def predict(data):
    # Reemplaza las variables que utiliza tu modelo
    row = np.array([data['id'], data['anno'], data['mes'], data['hora'],
                   data['hora_punta'],
                   data['reduccion_tp'], data['vacacional'],
                   data['tipo_elem_M30'],
                   data['tipo_elem_URB'], data['fin_de_semana_0'],
                   data['fin_de_semana_1'],
                   data['festivo_0'], data['festivo_1'],
                   data['estacion_Invierno'], data['estacion_Otono'],
                   data['estacion_Primavera'], data['estacion_Verano'],
                   data['dia_semana_Friday'], data['dia_semana_Monday'],
                   data['dia_semana_Saturday'], data['dia_semana_Sunday'],
                   data['dia_semana_Thursday'], data['dia_semana_Tuesday'],
                   data['dia_semana_Wednesday']])

    X = pd.DataFrame([row], columns=['id', 'anno', 'mes', 'hora',
                                     'hora_punta',
```

```

        'reduccion_tp', 'vacacional',
        'tipo_elem_M30',
        'tipo_elem_URB', 'fin_de_semana_0',
        'fin_de_semana_1',
        'festivo_0', 'festivo_1',
        'estacion_Invierno', 'estacion_Otono',
        'estacion_Primavera', 'estacion_Verano',
        'dia_semana_Friday', 'dia_semana_Monday',
        'dia_semana_Saturday', 'dia_semana_Sunday',
        'dia_semana_Thursday',
        'dia_semana_Tuesday',
        'dia_semana_Wednesday'])

prediction = modelo.predict(X)[0]
return prediction

@app.route('/predict', methods=['POST'])
def predict_endpoint():
    try:
        data = request.get_json()
        prediction = predict(data)
        return jsonify({"prediction": float(prediction)})
    except Exception as e:
        return jsonify({"error": str(e)})

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080, debug=False)

```

.9.3. app.py

```

# frontend.py
import streamlit as st
import requests
import pandas as pd
from transformaciones import *
import datetime
import random

```

```
import base64

# Store the initial value of widgets in session state
if "visibility" not in st.session_state:
    st.session_state.visibility = "visible"
    st.session_state.disabled = False

distritos = pd.read_csv('C:/Users/AlejandroAlvarez/OneDrive - Rock
    Internet SL/Desktop/Visualizacion Traffic
    Estimation/Backend/distritos.csv')
festivos = pd.read_excel('C:/Users/AlejandroAlvarez/OneDrive - Rock
    Internet SL/Desktop/Visualizacion Traffic
    Estimation/Backend/Festivos_Madrid.xls')

# Obtener los nombres nicos de la columna 'nombre_columna'
distritos_list = list(distritos['distrito'].unique())

st.title(':car: Intensidad de tráfico ')

distrito = st.selectbox(
    "Distrito",
    distritos_list,
    label_visibility=st.session_state.visibility,
    disabled=st.session_state.disabled,
)

# id
ids = distritos[distritos['distrito']==distrito]['id'].unique()

# Predicciones
```

```
predictions = []

# Fecha
fecha = st.date_input("Selecciona una fecha", datetime.date(2023, 6, 13))

# Crea un slider para representar la hora (0 a 24)
hora = st.slider("Selecciona una hora", min_value=0, max_value=24, step=1,
                 value=12)

# Obtn el campo "tipo_elem" como una cadena para el distrito especfico
tipo_elem = distritos.loc[distritos['distrito'] == distrito,
                           'tipo_elem'].unique()[0]

# Calculo de las variables temporales adicionales
estacion = calcular_estacion(fecha.day, fecha.month)
dia_semana = calcular_dias_semana(fecha.year, fecha.month, fecha.day)
hora_punta = calcular_hora_punta(fecha.month, dia_semana, hora)
reduccion_tp = calcular_reduccion_tp(hora)
vacacional = calcular_vacacional(fecha.year, fecha.month, fecha.day)
fin_de_semana = es_fin_de_semana(fecha=fecha)

# st.write(f"dia de la semana: {dia_semana}")
# st.write(f"Estacion: {estacion}")

# Festivo
# 1. Convertimos el campo dia a fecha para eliminar las horas
festivos['Dia'] = festivos['Dia'].dt.date
# 2. Convierte la fecha a un objeto datetime
fecha_a_verificar = pd.to_datetime(fecha)
# Busca en el DataFrame de festivos si la fecha (sin horas) est marcada
# como festivo
conteo_festivo = len(festivos[(festivos['Dia'] == fecha_a_verificar) &
                               (festivos['Festivo'] == 1)])
# Asigna el valor 1 a la variable festivo si el conteo es mayor que 0, de
# lo contrario, asigna 0
```

```
festivo = 1 if conteo_festivo > 0 else 0

# Dummification

# 1. Estaciones
estaciones = {'Invierno': 0, 'Otono': 0, 'Primavera': 0, 'Verano': 0}

if estacion in estaciones:
    estaciones[estacion] = 1

variables_dinamicas = {
    f'estacion_{estacion}': valor for estacion, valor in estaciones.items()
}

# 2. Dias semana
dias_semana = ['Friday', 'Monday', 'Saturday', 'Sunday', 'Thursday',
                'Tuesday', 'Wednesday']

# Crea un diccionario para mapear los das de la semana a sus valores
# correspondientes
dias_semana_mapping = {f'dia_semana_{dia}': 0 for dia in dias_semana}

# Asigna 1 al valor correspondiente al da de la semana deseado (por
# ejemplo, "Saturday")
# Aqu puedes cambiar "Saturday" por el da de la semana que necesites.
if f'dia_semana_{dia_semana}' in dias_semana_mapping:
    dias_semana_mapping[f'dia_semana_{dia_semana}'] = 1

# 3. Festivo
festivo_0 = 1 if festivo == 0 else 0
festivo_1 = 1 if festivo == 1 else 0

# 4. fin_de_semana_1
fin_de_semana_0 = 1 if fin_de_semana == 0 else 0
fin_de_semana_1 = 1 if fin_de_semana == 1 else 0
```

```
# 5. Tipo elem
tipo_elem_URB = 1 if tipo_elem == 'URB' else 0
tipo_elem_M30 = 1 if tipo_elem == 'M30' else 0

if st.button('Predict'):

    # Realizar predicciones para cada ID y almacenarlas en la lista
    for id in ids:
        # Crear un diccionario con las variables y sus nombres como claves
        input_data = {
            'id': int(id),
            'anno': int(fecha.year),
            'mes': int(fecha.month),
            'hora': int(hora),
            'hora_punta': int(hora_punta),
            'reduccion_tp': int(reduccion_tp),
            'vacacional': int(vacacional),
            'tipo_elem_M30': int(tipo_elem_M30),
            'tipo_elem_URB': int(tipo_elem_URB),
            'fin_de_semana_0': int(fin_de_semana_0),
            'fin_de_semana_1': int(fin_de_semana_1),
            'festivo_0': int(festivo_0),
            'festivo_1': int(festivo_1),
            'estacion_Invierno':
                int(variables_dinamicas['estacion_Invierno']),
            'estacion_Otono': int(variables_dinamicas['estacion_Otono']),
            'estacion_Primavera':
                int(variables_dinamicas['estacion_Primavera']),
            'estacion_Verano': int(variables_dinamicas['estacion_Verano']),
            'dia_semana_Friday':
                int(dias_semana_mapping['dia_semana_Friday']),
            'dia_semana_Monday':
                int(dias_semana_mapping['dia_semana_Monday']),
            'dia_semana_Saturday':
                int(dias_semana_mapping['dia_semana_Saturday']),
```

```
'dia_semana_Sunday':
    int(dias_semana_mapping['dia_semana_Sunday']),
'dia_semana_Thursday':
    int(dias_semana_mapping['dia_semana_Thursday']),
'dia_semana_Tuesday':
    int(dias_semana_mapping['dia_semana_Tuesday']),
'dia_semana_Wednesday':
    int(dias_semana_mapping['dia_semana_Wednesday'])
}

response = requests.post("http://127.0.0.1:8080/predict",
    json=input_data)

if response.status_code == 200:
    result = response.json()
    prediction = float(result["prediction"]) # Convertir a float
    # Ajustar la predicción a cero si es negativa
    prediction = max(prediction, 0)
    predictions.append(prediction)

# Calcular la media de las predicciones
if predictions:
    average_prediction = sum(predictions) / len(predictions)
    st.success(f'Predicción de la intensidad: {average_prediction:.2f} '
              'vehículos/hora')
```


Bibliografía

- [1] W. Jiang, J. Luo. *Big Data for Traffic Estimation and Prediction: A Survey of Data and Tools.*
- [2] Datos de Tráfico en Madrid
- [3] Ubicación de los puntos de medición en Madrid
- [4] Calendario Laboral de Madrid
- [5] Datos Meteorológicos en Madrid