Trenes en Véneto
Retrasos y modelo de clasificación

Informe final del TFM
para el máster en *Data Science e Inteligencia Artificial*

EBIS Business Techschool | EUNEIZ – Universidad de Vitoria-Gasteiz

Alessandro Bigolin

2025

Contenidos

1 Introducción

Este TFM tiene como objetivo diseñar y ejecutar un proyecto de ciencia de datos e inteligencia artificial sobre el tráfico ferroviario en la región italiana de Véneto, cubriendo el período de enero a mayo de 2025 (inclusive). Concretamente: (i) destacaré patrones recurrentes de retrasos; (ii) construiré un modelo predictivo de ocurrencia de retrasos; y (iii) publicaré una herramienta visual e interactiva en Tableau Public que haga accesibles los hallazgos.

El proyecto fue concebido inicialmente para la red de Renfe Rodalies en el área metropolitana de Barcelona. Sin embargo, la ausencia de datos abiertos para esta red lo hizo

inviable. En cambio, Trenitalia (el principal operador ferroviario de Italia) proporciona datos abiertos a través de ViaggiaTreno. Un servicio independiente y de acceso público, TrainStats, procesa estos datos en archivos JSON diarios que contienen información detallada sobre los movimientos de trenes de pasajeros en Italia (trenes de alta velocidad no incluidos) y los pone a disposición del público a través de Dropbox.

Elegí centrarme en la región del Véneto por dos razones principales. Primero, es mi región de origen, lo que me permite interpretar los datos teniendo en cuenta su geografía e infraestructura local. Segundo, el conjunto de datos solo para el Véneto, durante el período seleccionado de cinco meses, ya comprende aproximadamente 1,5 millones de registros. Ampliar el análisis a toda la red ferroviaria italiana habría producido, por lo tanto, un conjunto de datos de tamaño excesivo para los recursos disponibles para este proyecto.

La memoria se estructura de la siguiente manera. En la §2 describo la recopilación de datos. La §3 detalla el preprocesamiento y la limpieza de los datos, incluyendo un análisis exploratorio. La §4 aborda el modelado clasificatorio, incluyendo la división del conjunto de datos, la creación de características históricas, el entrenamiento base, la optimización de hiperparámetros y una evaluación del modelo final. La §5 presenta la interpretación del modelo mediante LIME y una analisis de los sesgos del modelo. En la §6 describo el desarrollo del dashboard interactivo en Tableau Public. Finalmente, en la §7 presento las conclusiones del proyecto y posibles direcciones futuras.

2 Recopilación de los datos

En la primera fase del trabajo (Notebook 1a) me centré en la recopilación de los datos. Inicialmente, generé un DataFrame de Pandas con todas las estaciones que ofrecen servicio en la región del Véneto, obteniendo la información mediante *web scraping* de la página web oficial de Trenitalia dedicada a las estaciones del Véneto.[1] A continuación, enriquecí esta lista con coordenadas geográficas para cada estación mediante llamadas a la API de Google Maps y guardé los datos como un archivo CSV, constituyendo así el conjunto de datos de la lista de estaciones (`stations_list`).

Tabla 1: Primeras 5 estaciones del conjunto de datos `stations_list`

Estación	Dirección	Ciudad	Prov.	Latitud	Longitud
Abano	Via della Stazione, 10	Abano Terme	PD	45.3621	11.7902
Adria	Via Umberto Maddalena, 32	Adria	RO	45.0555	12.0560
Alano-Fener-Valdobbiadene	Via Stazione, 106	Alano di Piave	BL	45.9026	11.9444
Albaredo	Via Stazione, 60	Vedelago	TV	45.6662	12.0122
Altavilla-Tavernelle	Piazzale Stazione, 35	Altavilla Vicentina	VI	45.5118	11.4544

Posteriormente, utilicé este conjunto de datos como referencia para filtrar los archivos JSON diarios proporcionados por TrainStats que contenían registros detallados de todos los movimientos de trenes en Italia entre el 1 de enero y el 31 de mayo de 2025. Para la construcción del conjunto de datos principal, conservé únicamente los trenes que realizaron al menos una parada en una estación de Véneto.

Los archivos JSON diarios incluyen varias secciones, como `giorno` (fecha de referencia), `timeZone` (desplazamiento horario en segundos), `riassunto` (estadísticas agrega-

[1] Véase https://www.trenitalia.com/it/regionale/veneto/stazioni-servite-da-trenitalia-veneto.html.

das del día), `avvisiRFI` y `avvisiTI` (avisos oficiales), y `treni`, que constituyó la sección principal para este proyecto. Cada objeto en `treni` representa un tren monitorizado, con atributos como número, estación de salida y llegada y sus respectivos andenes, categoría, horarios de salida y llegada programados, y un array de paradas. Cada parada incluye nombre de estación, andén de llegada y salida, horas programadas de llegada y salida, y retraso de llegada y salida.

Iteré sobre todos los archivos JSON diarios, filtrando los viajes que incluían al menos una estación presente en `stations_list`. Extraí para cada parada atributos relevantes como número de tren, categoría, origen, destino, nombre de estación, horarios programados y retrasos de llegada y salida, manteniendo el orden cronológico de las paradas dentro de cada viaje y el orden de los viajes según los archivos diarios. Finalmente, agregué todos los registros en un único archivo, `trains_data_Veneto.json`, y también transformé los datos a un formato CSV. El conjunto de datos final contiene aproximadamente 1,5 millones de registros y 10 columnas.

Tabla 2: Últimos 5 registros del conjunto de datos `trains_data_Veneto`

Día	Nº Tren	Cat.	Origen	Destino	Estación	Llegada Prog.	Salida Prog.	Retr. Lleg.	Retr. Sal.
31/05/2025	2647	REG	MILANO CENTRALE	VERONA PORTA NUOVA	ROVATO	1748729940	1748730000	-2	1
31/05/2025	2647	REG	MILANO CENTRALE	VERONA PORTA NUOVA	BRESCIA	1748730660	1748730780	-2	1
31/05/2025	2647	REG	MILANO CENTRALE	VERONA PORTA NUOVA	DESENZANO D. G.-SIRMIONE	1748731680	1748731740	1	2
31/05/2025	2647	REG	MILANO CENTRALE	VERONA PORTA NUOVA	PESCHIERA DEL GARDA	1748732220	1748732280	1	3
31/05/2025	2647	REG	MILANO CENTRALE	VERONA PORTA NUOVA	VERONA PORTA NUOVA	1748733420	0	-6	N

3 Preprocesamiento y limpieza de los datos

3.1 Homogeneización de variables categóricas

En el Notebook 1b, comencé la fase de preprocesamiento. Primero verifiqué la consistencia de los nombres de las estaciones en los dos conjuntos de datos creados en los pasos anteriores: `stations_list` (cargado como `stations_df`) y `trains_data_Veneto` (cargado como `trains_df`). El objetivo principal era asegurar que todas las estaciones listadas en `stations_df` y presentes en los archivos JSON diarios originales estuvieran escritas de manera consistente en el conjunto de datos `trains_df`, para garantizar que ninguna estación relevante hubiera sido omitida o mal etiquetada inadvertidamente durante la adquisición y el filtrado de datos.

Muchas estaciones aparecían con variaciones en sus nombres. El primer paso fue, por lo tanto, asegurar la consistencia en la nomenclatura de las estaciones. Para lograr esto, utilicé un *fuzzy matching* con el objetivo de estandarizar los nombres de las estaciones dentro de los conjuntos de datos. Algunos reemplazos en `trains_df` debieron hacerse manualmente, ya que a veces no fue posible determinar de forma automatizada si dos nombres se referían a la misma estación o a estaciones distintas. Por ejemplo, “Marano” y “Marano Vicentino” corresponden a la misma estación, mientras que “Paese” y “Paese Castagnole” corresponden a dos estaciones diferentes. Se presentaron otros casos similares.

Después de estandarizar los nombres de las estaciones, verifiqué que todas las estaciones en `stations_df` estuvieran presentes en `trains_df`. Encontré que 13 estaciones no aparecían en este último. Una revisión en fuentes externas confirmó que dichas estaciones no estaban operativas durante el período analizado o permanecían cerradas de manera indefinida.

3.2 Tratamiento de valores nulos y fusión de los datasets

A continuación, analicé los tipos de columnas de ambos conjuntos de datos y me aseguré de que cada columna tuviera el tipo de datos apropiado. También verifiqué si había valores faltantes y los manejé según fuese necesario. El conjunto de datos `trains_df` presentaba valores faltantes en tres columnas: `arrival_delay`, `departure_delay` y `category`. Me centré primero en la columna `category`. Analicé los trenes con categoría faltante y observé que la mayoría tenían un número de tren de cuatro dígitos que comenzaba con 8 o 9, mientras que los restantes eran de cinco dígitos, en su mayoría comenzando con 35. También examiné los trenes con categoría válida y observé patrones regulares en la relación entre el prefijo y la longitud del número de tren y su categoría. A partir de estos patrones, construí un mapeo para combinaciones de prefijo–longitud donde una sola categoría ocurría con más del 95 % de frecuencia y utilicé este mapeo para imputar los valores faltantes. Lamentablemente, la imputación redujo el número de valores faltantes solamente de 8. Los valores restantes se asignaron a una categoría de marcador de posición, `UNK` (Desconocido).

En cuanto a los retrasos, dejé sin modificar los valores faltantes de `arrival_delay` en la primera estación de cada tren y de `departure_delay` en la última estación, ya que no era posible imputarlos de forma razonable. Para las estaciones intermedias, imputé los valores faltantes de `arrival_delay` tomando el promedio del retraso de salida de la estación anterior y el de la estación actual. En la última estación de cada tren, utilicé el retraso de salida de la penúltima estación. De manera similar, los valores faltantes de `departure_delay` en estaciones intermedias se imputaron como el promedio del retraso de llegada de la estación actual y de la siguiente.

Las columnas de horarios planificados, `scheduled_departure` y `scheduled_arrival`, constituyeron un caso especial. Aunque a primera vista no contenían valores faltantes, en realidad deberían presentarlos: la columna `scheduled_departure` no debería estar definida en la última estación de un viaje, y la columna `scheduled_arrival` tampoco en la primera estación. En el dataset, estos valores faltantes legítimos estaban codificados como `0`. Dado que `0` no puede representar una marca de tiempo aceptable (las columnas se almacenan como tiempos Unix), reemplacé todos los ceros por `NaN`.

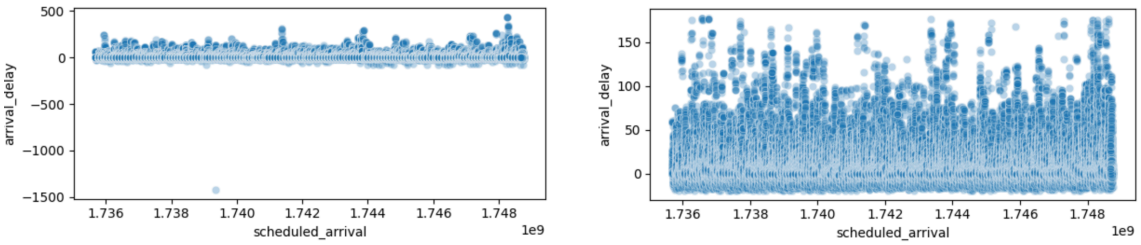
Posteriormente, integré el `trains_df` con la información de `stations_df`, realizando un *left merge* para añadir la provincia, latitud y longitud a cada parada de tren en el Véneto (aunque las columnas de latitud y longitud no se usarían en el entrenamiento del modelo, resultaron útiles para la visualización y el ploteo de los datos).

3.3 Detección y tratamiento de outliers

En esta etapa me centré en la identificación y el tratamiento de valores anómalos en las columnas numéricas del conjunto de datos `trains_df`, en particular `arrival_delay`, `departure_delay`, `scheduled_arrival` y `scheduled_departure`. El objetivo fue garantizar que los datos resultaran consistentes y adecuados para el entrenamiento de modelos de aprendizaje automático.

Comencé con una inspección exploratoria de valores atípicos, tanto uni como bivariables. Para las variables numéricas utilicé boxplots e identifiqué posibles outliers aplicando la regla de $1.5 \times$IQR. Para las variables categóricas analicé las distribuciones de frecuencia. También exploré relaciones entre pares de variables numéricas mediante diagramas de dispersión, relaciones numérico-categóricas mediante boxplots, y relaciones entre pares de variables categóricas mediante tablas cruzadas.

La exploración reveló algunos valores extremadamente altos o bajos en las variables de retraso. Una posible estrategia habría sido aplicar técnicas de recorte (capping o Winsorization), limitando los valores a percentiles como el 1 y el 99. Sin embargo, al probar este procedimiento los máximos de `arrival_delay` quedaban reducidos a apenas 34 minutos. Consideré que esta aproximación suponía un riesgo significativo de perder información relevante, por tres razones principales: (i) un rango artificialmente estrecho podría inducir al modelo a sobreajustarse a los casos comunes e ignorar retrasos extremos pero plausibles; (ii) los retrasos elevados suelen reflejar dinámicas importantes del sistema ferroviario (efectos de propagación entre estaciones o características de ciertas rutas), cuya eliminación impediría al modelo captarlas; y (iii) los modelos tenderían a sesgarse hacia valores promedio, subestimando de forma sistemática los retrasos de mayor magnitud. Por estas razones opté por una estrategia más conservadora: eliminar únicamente las entradas con retrasos superiores a 180 minutos o inferiores a -20 minutos. Este umbral se justificó tanto por la rareza de estos casos como por la evidencia gráfica de que, dentro de estos límites, aún se observan patrones regulares en los datos (Figura 1).



(a) Antes del tratamiento

(b) Después del tratamiento

Figura 1: Distribución de retrasos de llegada (en minutos) a lo largo del conjunto de datos (1 de enero a 31 de mayo), antes y después del tratamiento de outliers.

Finalmente, en el caso de la variable `category`, decidí agregar las categorías minoritarias `EXP`, `IR` y `NCL` a la categoría `UNK` (Desconocido). NCL está por Non Clasificado. Además, las tres categorías representaban un número muy reducido de trenes en comparación con las otras categorías (véase Tabla 3), por lo que mantenerlas separadas no habría aportado información estadísticamente robusta y solo habría introducido ruido en los modelos posteriores.

Tabla 3: Comparación de la frecuencia de paradas por categoría de trenes antes y después de unificar las categorías minoritarias y tratar los outliers en las columnas numéricas.

	REG	UNK	IC	EC	ICN	EN	EXP	IR	NCL
Antes	1 269 062	130 872	22 426	16 896	5 671	4 609	177	23	14
Después	1 268 950	131 045	22 299	16 877	5 442	4 541	-	-	-

3.4 Transformación y codificación de variables

Tras tratar los outliers, realicé varias transformaciones para preparar los datos para el entrenamiento de los modelos de aprendizaje automático. En primer lugar, generé nuevas columnas a partir de los horarios planificados: extraje horas y minutos de `scheduled_departure` y `scheduled_arrival`, y a su vez creé transformaciones usando funciones seno y coseno para capturar la naturaleza cíclica del tiempo. Posteriormente

(Notebook 2), decidí conservar únicamente las columnas numéricas correspondientes a horas y minutos, descartando tanto las versiones originales en formato Unix como las transformaciones seno/coseno, ya que los modelos basados en árboles seleccionados no se beneficiaban de la representación cíclica.

A continuación, extraje el día de la semana a partir de la columna `day` y codifiqué las variables categóricas `category` y `province` mediante one-hot encoding. Otras columnas categóricas podrían haberse transformado de la misma manera, pero algunas (p.ej., `station`) contenían muchos valores únicos, lo que habría generado un conjunto de datos excesivamente amplio. Por esta razón, opté por utilizar modelos de IA capaces de manejar variables categóricas de forma nativa (véase §4.1) y solo apliqué codificación one-hot de manera parcial, suficiente para cumplir con los requisitos del TFM.

Posteriormente, decidí excluir ciertas columnas consideradas redundantes o problemáticas. Eliminé `lat` y `lon`, considerando que la información espacial estaba suficientemente representada mediante combinación de `station`, `province`, `origin` y `train_number`. También descarté las columnas numéricas generadas por el one-hot encoding y `departure_delay`, ya que esta última habría actuado como un proxy directo de `arrival_delay` y podría haber sesgado al modelo. Finalmente, excluí todas las columnas `scheduled_departure_...` y `scheduled_arrival_...` excepto las columnas numéricas `scheduled_arrival_hour` y `scheduled_arrival_minute`, que deberían ser suficientes para los modelos basados en árboles.

Luego, creé nuevas columnas para capturar la estructura de los viajes y facilitar la comprensión temporal de los trayectos. Generé `previous_station`, que indica la estación anterior de cada parada. Posteriormente eliminé la primera estación de cada viaje (`origin == station`), ya que esta estación carecía de valores de `arrival_delay`. A pesar de eliminar estas filas, la información sobre la estación de origen de cada viaje se conservó en el dataset, ya que `previous_station` en la primera parada restante de cada viaje indicaba cuál era la estación inicial. Además, creé `train_id` para identificar de manera unívoca cada viaje; `stop_index`, el índice de cada parada dentro del viaje; y `n_total_stops`, que indica el número total de paradas de cada viaje. Estas columnas permiten al modelo diferenciar distintos viajes y estimar la duración de los trayectos.

Finalmente, consideré los posibles efectos de los días festivos entre semana. Usando `train_id` y `day`, observé que en los días festivos el número de trenes era significativamente menor que en días laborables normales y cercano al promedio de fines de semana. Por ello, creé la columna `is_holiday`, que toma valor 1 para los días festivos entre semana y 0 en caso contrario.

Para garantizar la consistencia temporal de cara a la división del conjunto de datos en subconjuntos durante el entrenamiento de los modelos, verifiqué que todas las combinaciones de número de tren y categoría aparecieran en todos los meses del conjunto de datos. Aquellas entradas que no cumplían esta condición fueron eliminadas.

3.5 Análisis exploratorio

Al final del preprocesamiento, realicé un análisis exploratorio de datos para identificar patrones y tendencias en los retrasos. Utilicé gráficos de barras, scatterplot y mapas de calor para visualizar la distribución de los retrasos según diferentes variables, como categoría de tren, provincia, hora del día y día de la semana. Estos patrones pueden explorarse de manera interactiva en el Notebook 1b y en el dashboard final. Destaco en particular el gráfico de retraso medio de llegada frente al número medio de paradas programadas por hora del día, ya que muestra cierta correlación: a medida que aumenta el número de

paradas, el retraso medio también tiende a aumentar ligeramente, y viceversa (Figura 2).[2]

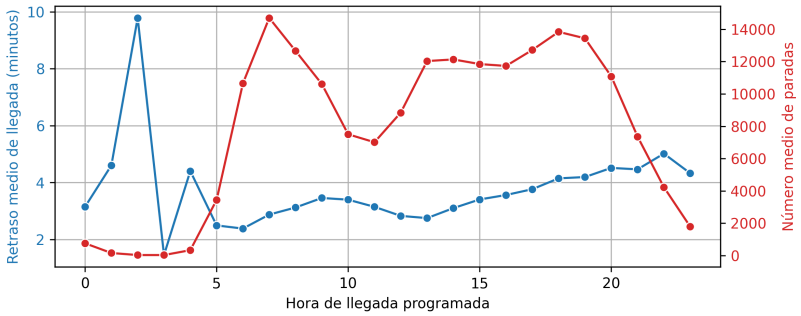


Figura 2: Retraso medio de llegada y número medio de paradas por hora del día.

También se enseña un mapa de calor por provincia y categoría de tren (Figura 3). Se observa que los trenes regionales (REG) tienen retrasos bajos, mientras que los de larga distancia presentan retrasos mayores.

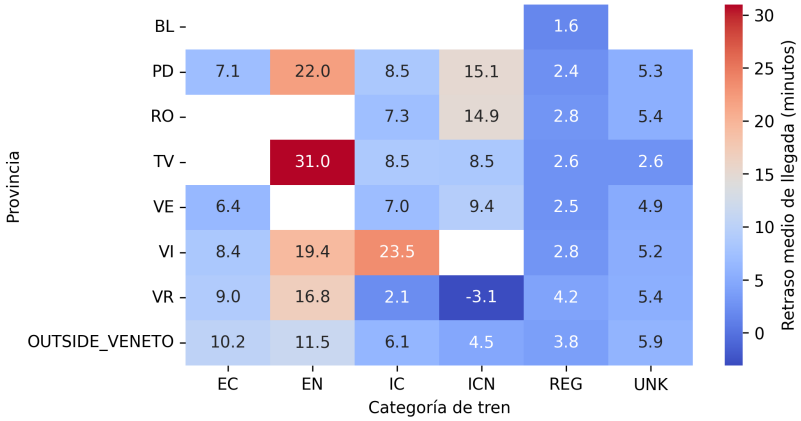


Figura 3: Retraso medio de llegada por provincia y categoría de tren.

4 Modelado de clasificación

En esta sección describo el desarrollo y evaluación del modelo de retrasos. El objetivo es predecir si un tren llegará con un retraso superior a 4 minutos a una estación determinada, planteando así un problema de clasificación binaria. Para definir este umbral, calculé el percentil 75 de la distribución de retrasos en intervalos acumulativos de enero a abril (excluyendo mayo, que constituirá completamente el conjunto de `test`) y promedié estos resultados. El valor obtenido, 4 minutos, se utilizó como umbral de clasificación.

4.1 Selección de los modelos

Para esta tarea, decidí entrenar modelos de aprendizaje automático basados en árboles. Excluí otros modelos, como K-Nearest Neighbors (KNN) o Support Vector Classifier

[2]Se observa una excepción durante las horas nocturnas, donde los retrasos son anómalamente altos (tal vez debido a operaciones de mantenimiento en las líneas, o porque a esas horas no circulan trenes regionales y emergen los patrones de retraso de los trenes nocturnos).

(SVC), debido a que, dado el tamaño del conjunto de datos, su escalabilidad es limitada: KNN requiere calcular distancias a todos los puntos de entrenamiento para cada predicción, y SVC resuelve un problema de optimización que involucra todos los puntos de entrenamiento, lo que resulta costoso en términos de tiempo y memoria. Los modelos basados en árboles, en cambio, construyen árboles de decisión que permiten un entrenamiento más eficiente y paralelizable, y por ello resultan más adecuados para este conjunto de datos. Asimismo, consideré la posibilidad de utilizar redes neuronales, dado que pueden capturar interacciones complejas y no lineales entre características. Sin embargo, en este caso resultan menos prácticas, ya que entrenar sobre el conjunto de datos completo sería muy intensivo en tiempo y recursos, además de presentar menor interpretabilidad en comparación con los modelos basados en árboles. Finalmente, para el entrenamiento base opté por LightGBM y CatBoost, dado que el dataset contiene varias columnas categóricas con un gran número de valores únicos. Estos modelos basados en árboles permiten manejar las variables categóricas de manera nativa, sin necesidad de técnicas de codificación que aumentarían excesivamente la dimensión del conjunto de datos.

4.2 División del conjunto de datos

Antes de realizar el entrenamiento de base, definí la variable objetivo y dividí el conjunto de datos en dos partes principales: el conjunto `trainval` (entrenamiento/validación) y el conjunto `test`. Dado que el dataset abarca los viajes en tren desde el 1 de enero hasta el 31 de mayo, la división se realizó siguiendo la progresión temporal de los datos, con el fin de asegurar que el modelo se entrene de manera realista y coherente con el tiempo. El conjunto `trainval` incluyó los datos desde el 1 de enero hasta el 30 de abril, mientras que el conjunto `test` abarcó todo mayo. Dado que cada fila del dataset corresponde a una parada de un viaje en tren, utilicé la columna `train_id`, que asigna un identificador único a cada viaje y se mantiene constante para todas sus paradas, para garantizar que cualquier viaje que cruce la división temporal se asigne completamente a un solo conjunto. El conjunto `test` se mantuvo completamente separado hasta la fase de evaluación del modelo final. El conjunto `trainval` se empleó tanto para el entrenamiento de base como, subdividido en folds, para la optimización de hiperparámetros mediante validación cruzada.

4.3 Características históricas

Para evitar fugas de información, el modelo nunca accede directamente a la columna `arrival_delay`, ya que esta está estrechamente relacionada con la variable objetivo (un tren se considera tarde si su retraso es mayor o igual a 4 minutos). No obstante, creé diversas características históricas derivadas de `arrival_delay`, que permiten capturar patrones sistemáticos de retrasos y constituyen los principales predictores del modelo:

- `mean_delay_station_train`: retraso medio de llegada para un tren específico en cada estación.
- `mean_delay_station_train_day`: retraso medio de llegada para un tren específico en cada estación y día de la semana.
- `mean_delay_station_cat_day`: retraso medio de llegada para trenes de la misma categoría en cada estación y día de la semana.

- `mean_delay_station_day_hour_half`: retraso medio para cada estación en intervalos de media hora, diferenciados por día de la semana. Para su cálculo, creé la columna `hour_half`, que asigna cada hora de llegada programada a un intervalo de 30 minutos a partir de la medianoche.

Para evitar fugas de datos, estas características históricas se calcularon únicamente sobre los conjuntos de entrenamiento y luego se copiaron a los conjuntos de validación. De esta manera, se garantizó que el modelo no utilizara información de retrasos futuros al aprender a predecir la variable objetivo.

Al calcular las características históricas traté temporalmente los días festivos que caen en medio de la semana laboral como si fueran sábados (`day_of_week` = 5), dado que el número de trenes en operación se asemeja más al de un sábado típico que al de un día laborable normal. Restauré el día correcto de la semana en la columna `day_of_week` antes de entrenar los modelos.

4.4 Entrenamiento base

Para el entrenamiento base, subdividí el conjunto `trainval` en un conjunto de entrenamiento (enero a marzo) y un conjunto de validación (abril). Utilicé las métricas de precisión, recall, F1, ROC_AUC y PR_AUC para evaluar el rendimiento de los modelos. Además, medí el tiempo de entrenamiento para cada modelo. Los modelos de base obtuvieron los siguientes resultados:

Tabla 4: Desempeño de los modelos de base sobre el conjunto de validación.

Modelo	Precisión	Recall	F1	ROC_AUC	PR_AUC	Tiempo entr. (s)
LightGBM	0.559	0.706	0.624	0.794	0.682	12.65
CatBoost	0.560	0.696	0.621	0.790	0.677	48.71

Como se observa, los resultados de los dos modelos son muy similares. No obstante, LightGBM presenta un tiempo de entrenamiento significativamente menor. Por esta razón, decidí centrarme exclusivamente en este modelo en la siguiente etapa de ajuste de hiperparámetros, ya que su rápida ejecución habría permitido explorar un mayor número de combinaciones de manera más eficiente, sin sacrificar el rendimiento predictivo.

4.5 Optimización de hiperparámetros

Para la optimización de hiperparámetros, preparé tres folds de validación cruzada utilizando el conjunto `trainval` (enero a abril). Adopté un enfoque de validación cruzada progresiva: cada fold utiliza un conjunto de entrenamiento creciente y un conjunto de validación posterior en el tiempo. Los folds se definieron de la siguiente manera:

- Fold 1: entrenamiento = enero, validación = febrero
- Fold 2: entrenamiento = enero-febrero, validación = marzo
- Fold 3: entrenamiento = enero-marzo, validación = abril

Para cada fold, calculé las características históricas únicamente sobre la porción de entrenamiento y luego las transferí al conjunto de validación correspondiente.[3] Realicé el ajuste de hiperparámetros muestreando 200 combinaciones de una cuadrícula predefinida de parámetros, estableciendo 2000 iteraciones de boosting y aplicando early stopping tras 50 rondas. Utilicé la métrica F1 para evaluar el rendimiento en cada fold y guiar la optimización de los hiperparámetros.

La mejor combinación de hiperparámetros produzco métricas consistentes en los tres conjuntos de validación, con valores estables entre folds (Tabla 5).

Tabla 5: Métricas de validación por fold para la mejor combinación de hiperparámetros.

Fold	Precisión	Recall	F1	ROC_AUC	PR_AUC
1	0.533	0.707	0.608	0.780	0.675
2	0.575	0.694	0.629	0.785	0.688
3	0.567	0.698	0.626	0.795	0.684

4.6 Entrenamiento final y evaluación

Entrené el modelo final con el conjunto `trainval` completo (viajes de enero a abril), utilizando el conjunto `test` con los viajes de mayo, separado al principio, como validación. Las características históricas se calcularon únicamente sobre el conjunto de entrenamiento y se trasladaron al conjunto de `test`, evitando así cualquier fuga de datos.

Tabla 6: Rendimiento del modelo final en los conjuntos de entrenamiento y test.

Conjunto	Precisión	Recall	F1	ROC_AUC	PR_AUC
Entrenamiento	0.599	0.757	0.669	0.837	0.740
Test	0.579	0.694	0.631	0.787	0.690

En el conjunto `test`, tanto precisión como recall disminuyen respecto al entrenamiento, aunque la recall se mantiene más alta, reflejando la tendencia del modelo a identificar más casos positivos. La F1 muestra una caída moderada. ROC-AUC y PR-AUC también se reducen, lo que indica una ligera pérdida de discriminación en datos no vistos. No obstante, las diferencias son contenidas, lo que sugiere un sobreajuste limitado.

La matriz de confusión (Tabla 7) muestra que el modelo clasifica correctamente la mayoría de los trenes a tiempo y un número sustancial de trenes retrasados. Sin embargo, tiende a producir un número bastante grande de falsos positivos, prediciendo un retraso cuando no ocurre ninguno. Esto ayuda al modelo a capturar una buena parte de los

[3] Dado que quería evitar dividir viajes individuales entre folds y también preservar el orden temporal (entrenando con viajes anteriores y validando con viajes más recientes), no utilicé directamente métodos como `GroupKFold` o `TimeSeriesSplit`. En su lugar, construí los folds manualmente, también asegurándome de que todas las filas correspondientes a un mismo viaje permanecieran dentro del mismo fold. Este enfoque garantizó que cada fold respetara el orden temporal del conjunto de datos, que las características históricas calculadas sobre la porción de entrenamiento del fold se refirieran a los viajes más antiguos, y que el modelo aprendiera a manejar progresivamente mayores cantidades de datos históricos, reflejando un escenario realista y mejorando su capacidad de generalización.

retrasos reales, pero también reduce la precisión, ya que muchas de las predicciones de retraso resultan ser falsas alarmas.

Tabla 7: Matriz de confusión del modelo final en el conjunto de test.

	Pred. 0	Pred. 1
Actual 0	124849	46138
Actual 1	27985	63412

Para evaluar la utilidad real del modelo, lo comparé con dos referencias simples basadas únicamente en estadísticas históricas de retraso por combinación de tren y estación. La primera referencia predice un retraso si el retraso histórico promedio supera los 4 minutos; sin embargo, ningún promedio excedía ese umbral, por lo que esta referencia no identificaba ningún retraso (precisión, recall y F1 = 0, accuracy = 0.652). La segunda referencia, basada en la fracción histórica de paradas retrasadas, calcula para cada tren y estación la proporción de paradas con retraso ≥ 4 minutos y predice retraso si esta fracción supera un umbral óptimo determinado para maximizar la F1 en el conjunto `test`. Esta estrategia alcanza una accuracy = 0.703, con métricas de precisión, recall y F1 muy similares a las del modelo LightGBM (accuracy = 0.718), ya que aprovecha directamente la información más predictiva contenida en el conjunto de datos. El modelo LightGBM, sin acceso directo a esa fracción histórica, reconstruye la señal predictiva a partir de las características disponibles, mostrando capacidad de generalización: puede predecir retrasos incluso para combinaciones de tren y estación no vistas previamente, donde las referencias históricas no serían aplicables. Esto evidencia que LightGBM aprende patrones subyacentes de los datos en lugar de memorizar resultados pasados.

5 Interpretabilidad del modelo

5.1 Importancia de las características

Para comprender mejor el comportamiento del modelo y la importancia relativa de las características, utilicé LIME (Local Interpretable Model-Agnostic Explanations).

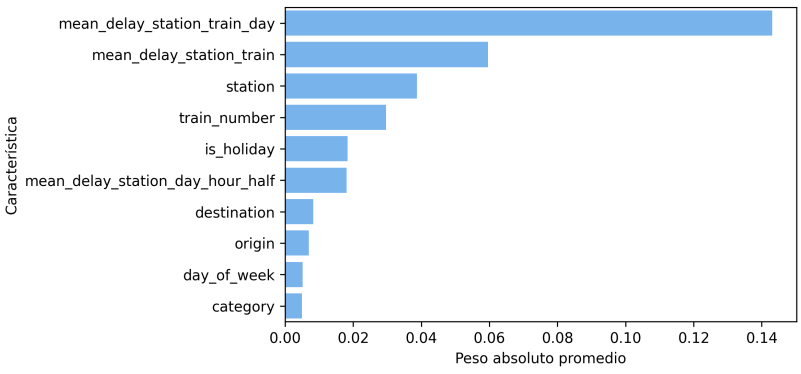


Figura 4: Importancia global de las características (calculada con LIME).

El análisis muestra que el modelo se apoya principalmente en los patrones históricos de retraso, en particular en `mean_delay_station_train_day` y `mean_delay_station_train`. Este resultado era esperable, ya que estas variables fueron diseñadas específicamente para aportar información sobre el rendimiento pasado y, de este modo, ayudar a

predecir retrasos futuros. Las columnas `station` y `train_number`, que también contribuyen a las dos variables anteriores, también muestran una contribución destacada. En cambio, variables como `day_of_week` o `category` tienen una importancia global menor. En el caso de `day_of_week`, es posible que el modelo ya recoja los patrones asociados a días laborables y fines de semana a través de las características históricas de retraso. Por su parte, la influencia limitada de `category` puede deberse tanto a la fuerte predominancia de trenes REG en el conjunto de datos como al ruido introducido por la categoría `UNK` (desconocida), que diluye la señal de las demás categorías.

5.2 Evaluación de sesgos

Evalué posibles sesgos en el modelo centrándome en la categoría, la provincia, el día de la semana y la hora de llegada programada. Para cada uno de estos aspectos, calculé el error residual medio restando la probabilidad predicha por el modelo de la etiqueta real (1 si hubo retraso ≥4 min, 0 en caso contrario). Un valor positivo indica que el modelo tiende a subestimar los retrasos (el retraso real es mayor que el predicho).

Tabla 8: Medias de residuos por provincia, categoría y día de la semana (conjunto test).

Provincia				**Categoría**				**Día de la semana**			
Prov	Mean	Std	Count	Cat	Mean	Std	Count	D	Mean	Std	Count
RO	0.04	0.42	5626	EC	0.08	0.46	3231	3	0.03	0.42	44887
PD	0.04	0.40	34254	UNK	0.07	0.44	23467	0	0.03	0.42	38609
VE	0.04	0.41	53871	IC	0.02	0.46	4184	2	0.02	0.42	39072
VR	0.02	0.43	27290	EN	0.02	0.50	768	1	0.01	0.42	33397
VI	0.02	0.38	18552	REG	0.01	0.41	229618	4	0.01	0.41	43623
TV	-0.01	0.40	35988	ICN	0.01	0.46	1116	5	0.01	0.40	38998
BL	-0.03	0.31	2155					6	0.01	0.40	23798
NO_VEN	0.01	0.43	84648								

En general, el modelo muestra desviaciones ligeras pero consistentes (Tabla 8; los gráficos correspondientes están disponibles en el Notebook 2, §7.2). La mayoría de las provincias presentan medias de residuos ligeramente positivas. De manera similar, se observan desviaciones ligeras hacia la subestima para categorías y días de la semana.

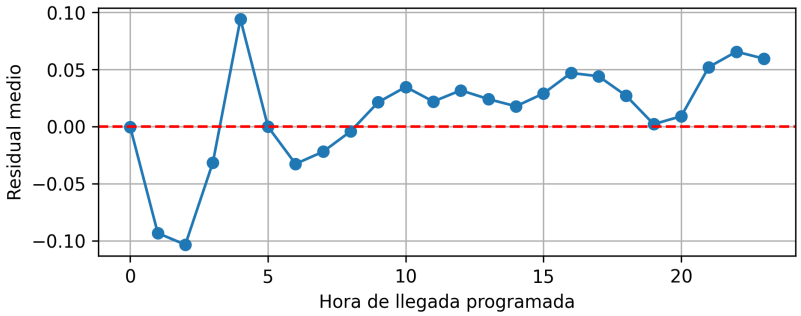


Figura 5: Medias de los residuos por hora de llegada programada.

Por hora del día (Figura 5), la subestimación es menor durante la mayor parte de la jornada, mientras que en la madrugada las medias de los residuos oscilan más debido a la menor cantidad de observaciones y a la irregularidad de los retrasos (cf. Figura 2).

6 Dashboard interactivo

Desarrollé el dashboard interactivo con Tableau Public.[4] El dashboard permite analizar los retrasos de trenes en la región de Véneto desde múltiples perspectivas, incluyendo distribuciones de retrasos por estación, provincia, categoría de tren, hora del día y día de la semana. También incluye visualizaciones de la distribución del tráfico de trenes por estación mediante un bubble map creado a partir de las columnas de `lat` y `lon` incorporadas durante el proyecto, así como un mapa coroplético con el retraso medio por provincia, con los contornos de las provincias extraídos de un archivo GeoJSON disponible en línea (Opendatasoft, CC BY 3.0) que integré con el conjunto de datos principal usando la columna `province` como clave. Finalmente, el dashboard contiene visualizaciones del rendimiento del modelo.

7 Conclusiones

En este proyecto, he desarrollado un modelo de aprendizaje automático para predecir retrasos de trenes en la región de Véneto, Italia. El proceso incluyó la adquisición y limpieza de datos, la ingeniería de características históricas, el entrenamiento y ajuste de modelos basados en árboles, y la evaluación de su rendimiento e interpretabilidad. Además, desarrollé un dashboard interactivo para el análisis visual de los datos y de los resultados.

El modelo logró identificar señales relevantes para la predicción de retrasos. En el conjunto `test` obtuvo un ROC-AUC de 0.787 y un PR-AUC de 0.690, superando la línea base aleatoria. El rendimiento global es moderado, con un F1-score de 0.631 y una precisión de 0.578. El análisis de importancia de características revela que las variables históricas de retraso son los principales predictores, mientras que la evaluación de sesgos indica ligeras subestimaciones en ciertas provincias y categorías de tren.

Algunas limitaciones pueden afectar el desempeño. En primer lugar, el conjunto de datos abarca únicamente el periodo de enero a mayo de 2025, por lo que no captura posibles patrones estacionales. Además, las características empleadas se restringen a horarios, datos categóricos y un historial limitado de retrasos, sin incorporar variables externas como condiciones meteorológicas, incidentes o obras en las líneas. Por último, la predicción de retrasos ferroviarios es, en esencia, un problema altamente estocástico, con un componente de aleatoriedad que ningún modelo puede eliminar por completo.

De cara a trabajos futuros, se podrían explorar modelos como LSTM o Transformers para capturar la propagación temporal de los retrasos. Asimismo, sería interesante aplicar codificaciones cíclicas para variables como la hora y el día de la semana, así como emplear Redes Neuronales de Grafos para modelar la estructura de estaciones y líneas ferroviarias. Otras líneas de mejora incluyen la creación de clústeres de estaciones para calcular estadísticas agregadas y la integración de fuentes de datos externas más completas, como información meteorológica o incidentes y obras en las líneas.

[4] El dashboard está disponible públicamente en el siguiente enlace: https://public.tableau.com/app/profile/alessandro.bigolin/viz/Trains_Veneto/Trains_Veneto_Dashboard.