UNIVERSIDAD DE
BUENOS AIRES

FACULTAD DE INGENIERÍA

[75.06/95.58] ORGANIZACIÓN DE
DATOS

Curso: Martinelli

1ER CUATRIMESTRE DE 2024

TRABAJO PRÁCTICO N° 2: Machine Learning

GRUPO: 6	
APELLIDO, Nombres	N° PADRÓN
Dominguez,Gonzalo Alejo	109759
Hsieh,Cindy Teresa	108051

tp2_analisis_y_plots.ipynb

regresor_lineal(baseline).ipynb

random_forest.ipynb

lightGMB.ipynb

Link carpeta con los CSVs subidos: CSVs entrega TP2

Link al video: https://youtu.be/095FaQrgYWk

Introducción:

El objetivo de este informe es explicar detalladamente la labor realizada para este segundo trabajo práctico.

Se incluye:

- Un detalle de la limpieza de los datos recibidos y las consideraciones que tuvimos.
- Análisis exploratorio que ayude a comprender en mayor profundidad los datos recibidos y que permita tener una idea de que se puede esperar de las predicciones realizadas por los modelos.
- Explicación de la labor de feature engineering realizada, qué features se dejaron y cuales se descartaron.
- Detalle del modelo baseline y los modelos más avanzados con búsqueda de hiperparametros utilizados.
- Resultados obtenidos en las predicciones subidas a la competencia de Kaggle

Datos Utilizados:

Se utilizaron a lo largo del trabajo los sig. conjuntos de datos:

1. **sales_train.csv**: Datos de ventas diarias que abarcan de enero de 2013 a octubre de 2015.
2. **test.csv**: Conjunto de pruebas para el cual hay que predecir las ventas de noviembre de 2015.
3. **items.csv**: Información adicional sobre los productos.
4. **item_categories.csv**: Información adicional sobre las categorías de productos.
5. **shops.csv**: Información adicional sobre las tiendas.

sample_submission.csv no lo utilizamos pero si sirvió de referencia para el armado de los resultados subidos a Kaggle.

Limpieza y Análisis Exploratorio

Link al colab: tp2_analisis_y_plots.ipynb

Después de cargar todos los datasets al notebook empezamos a analizar cada csv viendo principalmente de que valores estaba compuesto y qué información podemos aprovechar de cada uno.
En el que principalmente nos enfocamos fue en el dataset **sales_train.csv** que contenía la mayoría de la información que nos iba a ser útil a la hora de armar los set de entrenamiento y validación.

Lo primero que hicimos fue analizar la presencia de valores nulos, los cuales afortunadamente no había.
Lo que sí estaba presente eran datos con valores inválidos o incongruentes en algunos features.
Por un lado en 'item_price' había un único registro con un precio negativo, el cual descartamos. Luego en algunas filas notamos que 'item_cnt_day' el cual representaba la cantidad de item vendidos, para un id de ítem y un id de vendedor en un día, notamos valores negativos. Al no tener verdaderamente una certeza de que representaban esos datos (por ejemplo items devueltos) nos decantamos por descartarlos ya que tampoco queríamos correr el riesgo de entrenar lo modelos con información incorrecta, por ejemplo llenando esas filas con valores promedio o un valor como cero el cual sería incoherente en el contexto de los datos.
Cabe destacar igualmente que los datos descartados rondan las 7000 filas.
Siguiendo con **sales_train.csv** notamos que también habían seis filas duplicadas las cuales era necesario sacar ya que poco ayuda al modelo entrenar 2 veces con la misma informacion. Del par de filas solo nos quedamos con una para de esa manera mantener los datos.
Por último, en relación a la limpieza de datos, removimos outliers. Para ello usamos la técnica de detección de los mismos que implica usar un boxplot. Nos fijamos en los precios y en el número de ventas por día de los ítems. Efectivamente había una serie de outliers. Para el precio consideramos que un precio que supere los 50000 era outlier. Para las ventas después de los 1000. Nos quedamos con las filas que tuvieran valores menores a estos.

Feature Engineering

Acá vamos a detallar cómo a partir de los features ya presentes en los distintos sets de datos como generamos features nuevas, que consideraciones tuvimos y como las encodear en los casos necesarios.
Lo primero que hicimos fue tomar la columna 'date' del dataset **sales_train.csv** y pasarla al formato datetime reconocido por pandas. De la misma generamos cuatro nuevas features por un lado el año, el mes, el número del día y por último cuál fue el día de la semana en el que cayó esa fecha. Nos parecía importante tener ambos datos del día ya que no es lo mismo una compra a fin que a inicio de mes como también no es lo mismo las compras durante la semana y el fin de semana.
Para el día de la semana al tratarse de una feature categórica nos decidimos por utilizar One Hot encoding principalmente por su simplicidad junto con también el hecho de que al solo tener siete posibles valores el feature se agregan tan solo siete columnas nuevas y no habría una explosion en las dimensiones del set de datos y entrenamiento.
La siguiente feature agregada era vital ya que era el target que debían predecir los modelos, agregamos número de ventas por mes para cada conjunto de vendedor e item: item_cnt_month al dataset. Una vez agregada esta features como para la competencia se pedía que los valores predichos debían encontrarse dentro del rango de cero a veinte aplicamos esta modificación al feature generado para que los modelos ya aprendieran con ese rango de valores.
De este dataset agregamos también el total de ventas histórico por cada item.

Otra feature que nos pareció vital dejar fue date_block_num que enumera del 0 al 33 el número de mes a nivel histórico en todo el registro de ventas ya que permite al modelo tener aun mas referencia del tiempo.
Del dataset shops.csv generamos los sig. features.
shops.csv tiene solo dos columnas: shop_name y shop_id, shops_id era una feature que ya teníamos en el dataset de sales train y shop_name al ser una serie de string no nos servía de mucho. Quisimos averiguar si podíamos hacer uso de alguna manera de ese feature. Como estaba en ruso necesitábamos traducirlo al inglés, allí notamos que la primera palabra siempre es alguna ciudad de Rusia a la que pertenece la tienda, nos pareció interesante como para transformarlo a un feature. Por ende transformamos esos nombres de ciudades en valores numéricos usando label encoding, ya que a diferencia del caso de los días de la semana resultaba inviable por la cantidad de valores usar One Hot encoding. A esta nueva feature la agregamos a una columna nueva llamada 'city_code'. Por último descartamos la columna 'shop_name'.

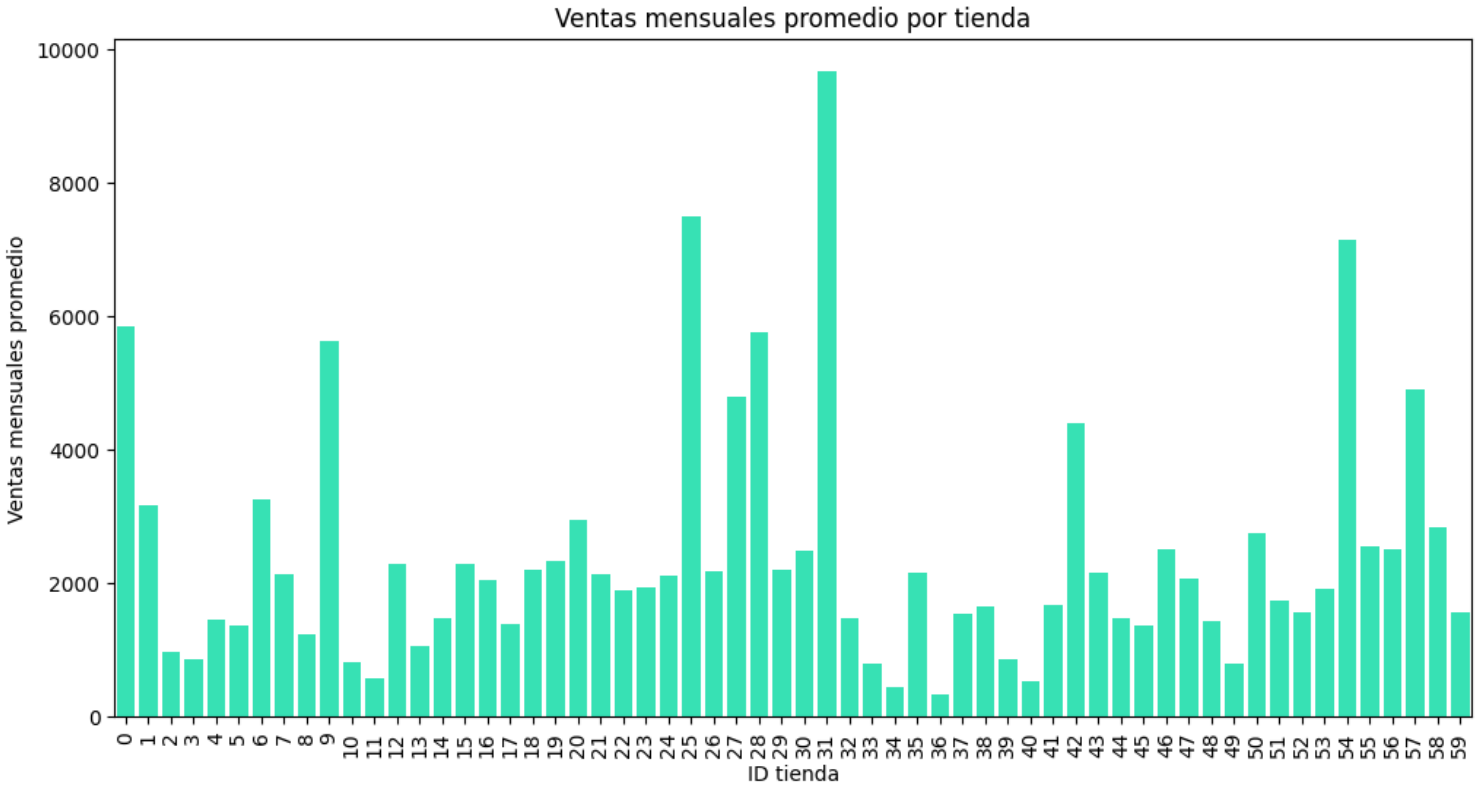
Del dataset item_categories.csv generamos los sig. features.
Similar al shops.csv, este está compuesto por 'item_category_name' e 'item_category_id'. Usando la misma técnica que en el caso anterior, detectamos que la primera palabra del ítem category_name representaba justamente la categoría general del ítem, suena redundante hacer un desglose más si ya tenemos el id de la categoría del ítem, pero una cosa es tener enumerados del 1 al 4 accesorios ps2/ps3/ps4/psp que tener un código para todos los que sean de la categoría 'accesorios', es por eso nos parecio util ese feature y creamos otra columna usando también label encoding: 'type_code'. Luego lo que hicimos fue descartar la columna item_category_name.

Merge de los datasets:
Finalmente ya teniendo todos los datasets limpios, analizados, con features nuevas y encodeadas procedimos a armar un dataset final que sirviera para generar los set de entrenamiento, validación y test.
Tomamos como base sales_train y lo mareamos con los demás datasets ya mencionados y de esa manera nos quedó un dataset final con los features que consideramos importantes para el entrenamiento del modelo.
Como los modelos aceptan solo valores numéricos sacamos afuera las fechas, el nombre de las tiendas y el nombre de los productos.

Los features que finalmente utilizamos son: date_block_num, shop_id, item_id, item_price, item_cnt_day, month, year, day, item_cnt_month, city_code, item_category_id, type_code, total_sales_item, y las features que van del 0 al 6 generadas por el OneHot encoding del dia de la semana.
En total son 19 features.

Plots

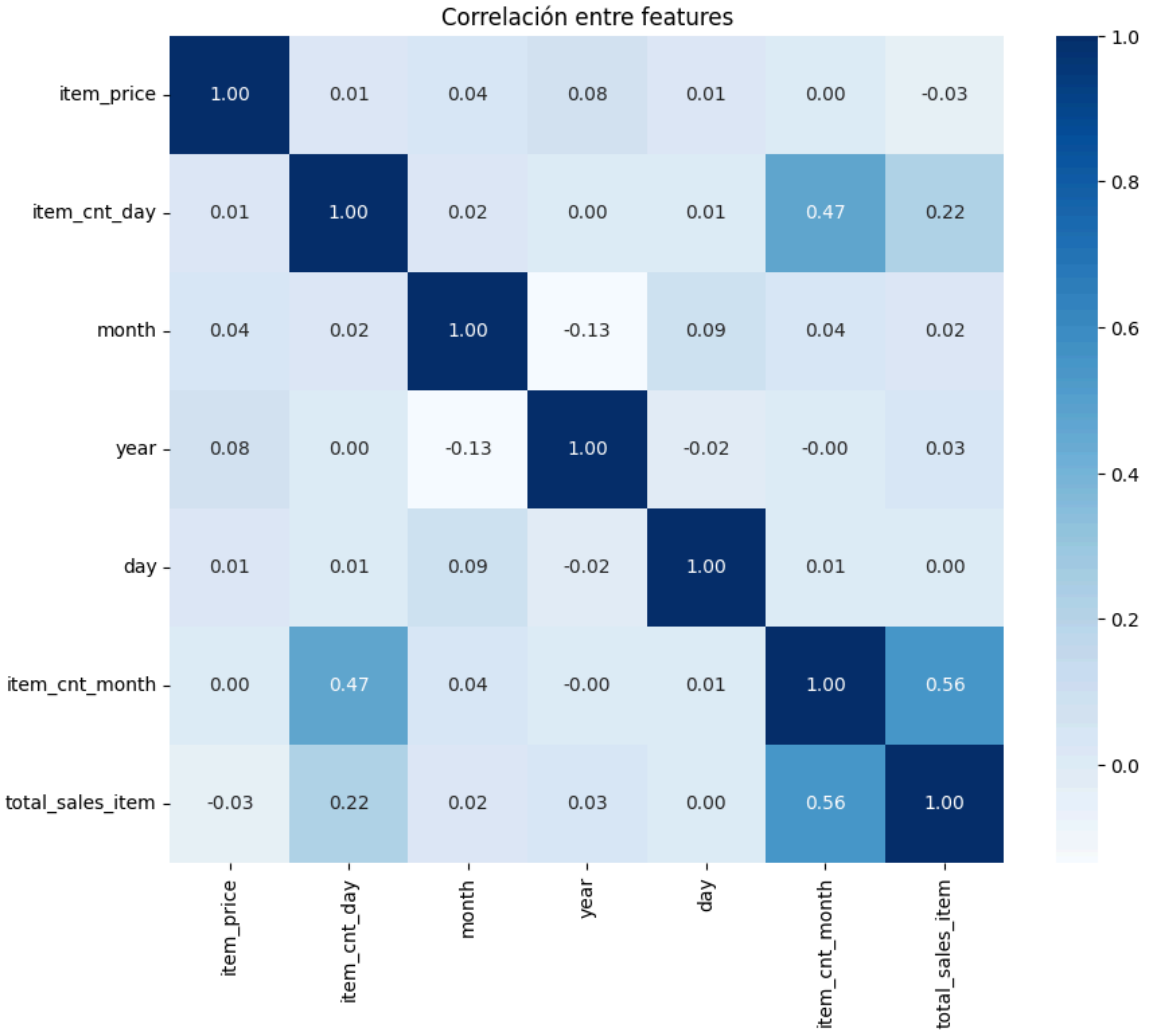
Gráfico 1: Ventas mensuales promedio por tienda



Al identificar las tiendas con mayores y menores ventas promedio por mes, podemos tener una idea de que podemos esperar de las predicciones que hagamos con los modelos. El ya tener una idea general del número de ventas mensuales podría indicar que en las predicciones por ejemplo el vendedor 31 (Que es el que más vende en promedio por mes) vas a ser el que más ventas también va a tener en el mes de noviembre de 2015.

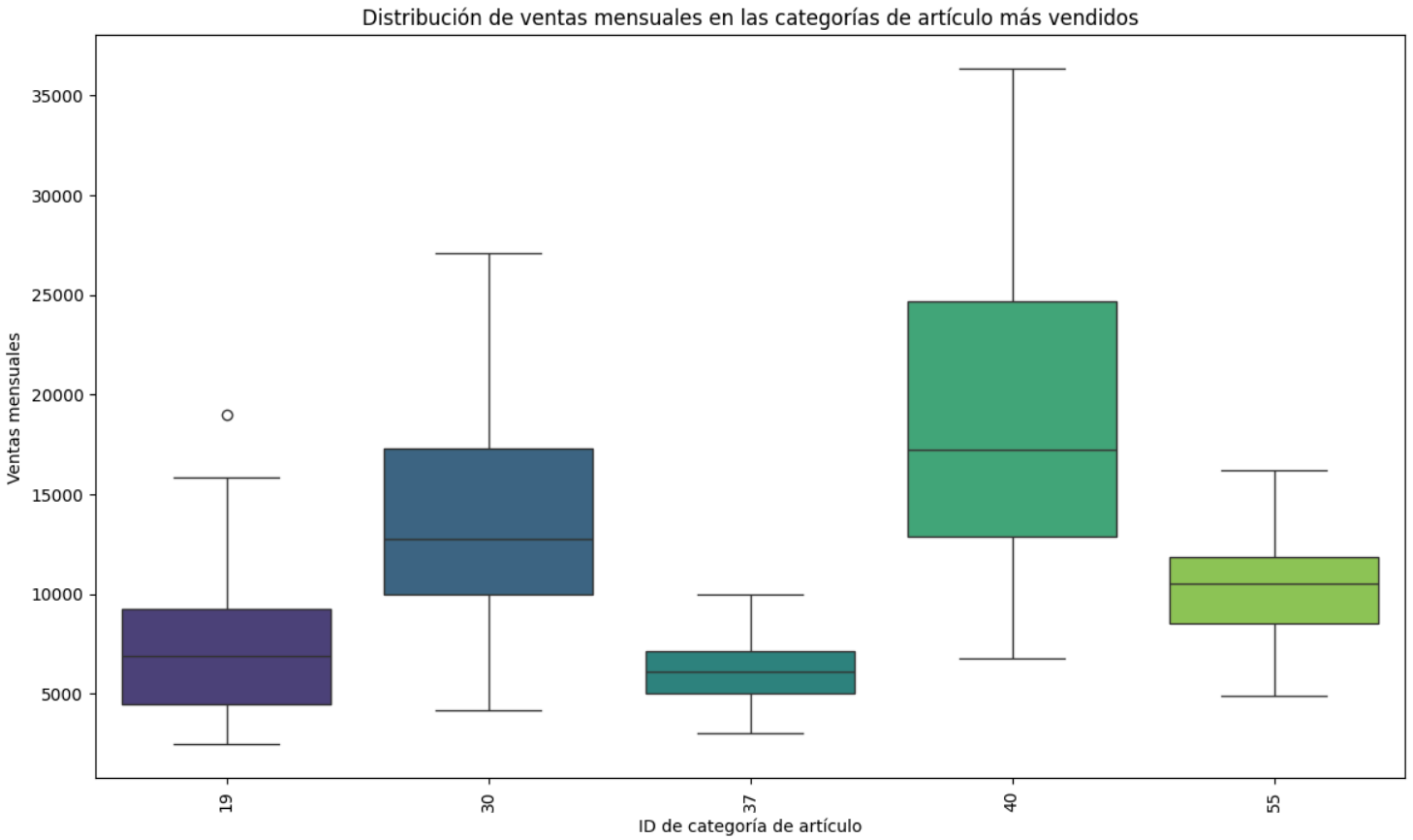
Gráfico 2: Correlación entre features

Lo que quisimos analizar es ver cómo se relacionan entre sí las distintas features que componen el dataset, del mismo nos quedamos solo con aquellas que tiene sentido analizarlas en una matriz de correlación. Nos interesaba ver si de manera básica con una matriz de correlaciones se podía detectar alguna correlación fuerte entre las variables.



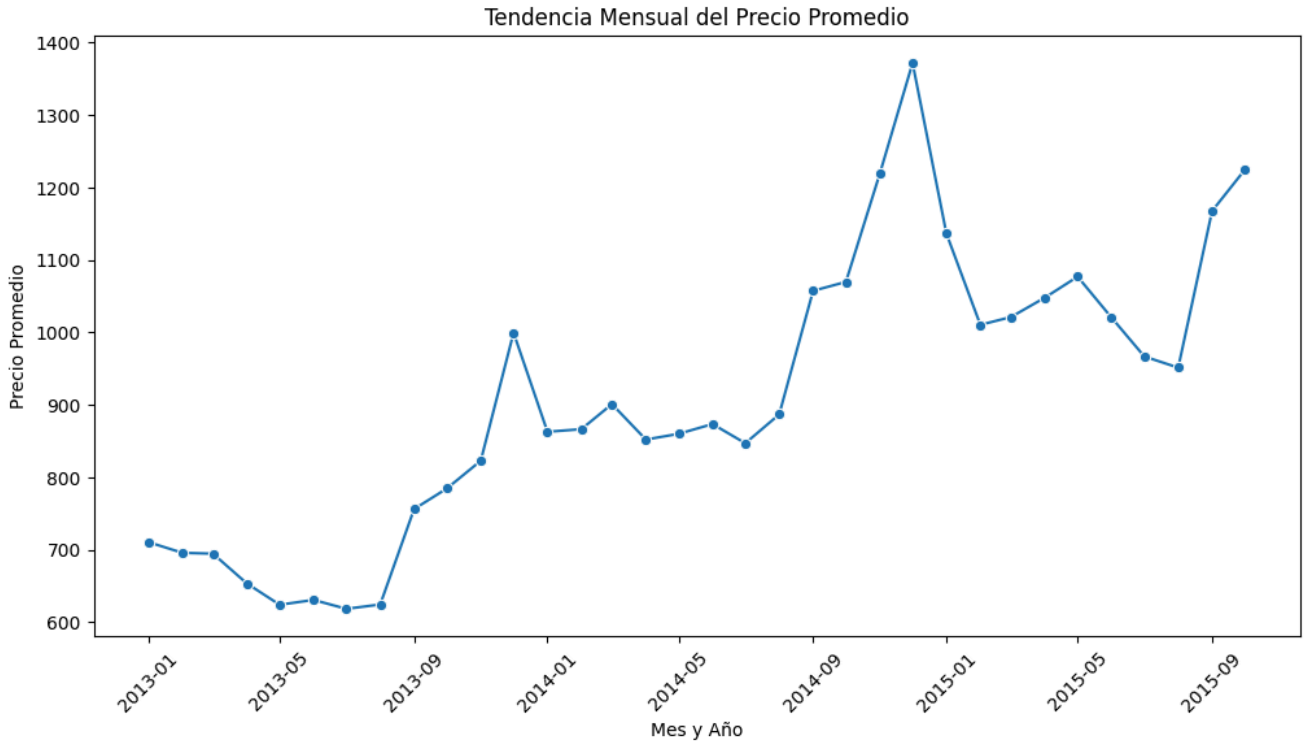
Como se puede observar aunque prácticamente todas las variables tienen una correlación positiva esta no resulta lo suficientemente alta. Creemos que esto sucede por la simplicidad de la matriz de correlaciones y que los modelos si serán capaces de detectar esas relaciones entre los features.

Gráfico 3: Distribución de Ventas Mensuales por Categoría de Artículo más vendidos.
Nos ayuda a ver que aquellos productos que pertenecen a estas categorías muy probablemente tengan un número de ventas en noviembre similar a los que se puede apreciar en los box plots.



Por ejemplo para un item de la categoría 37 es esperable que sus ventas para el mes de noviembre ronden un poco más de quince mil.

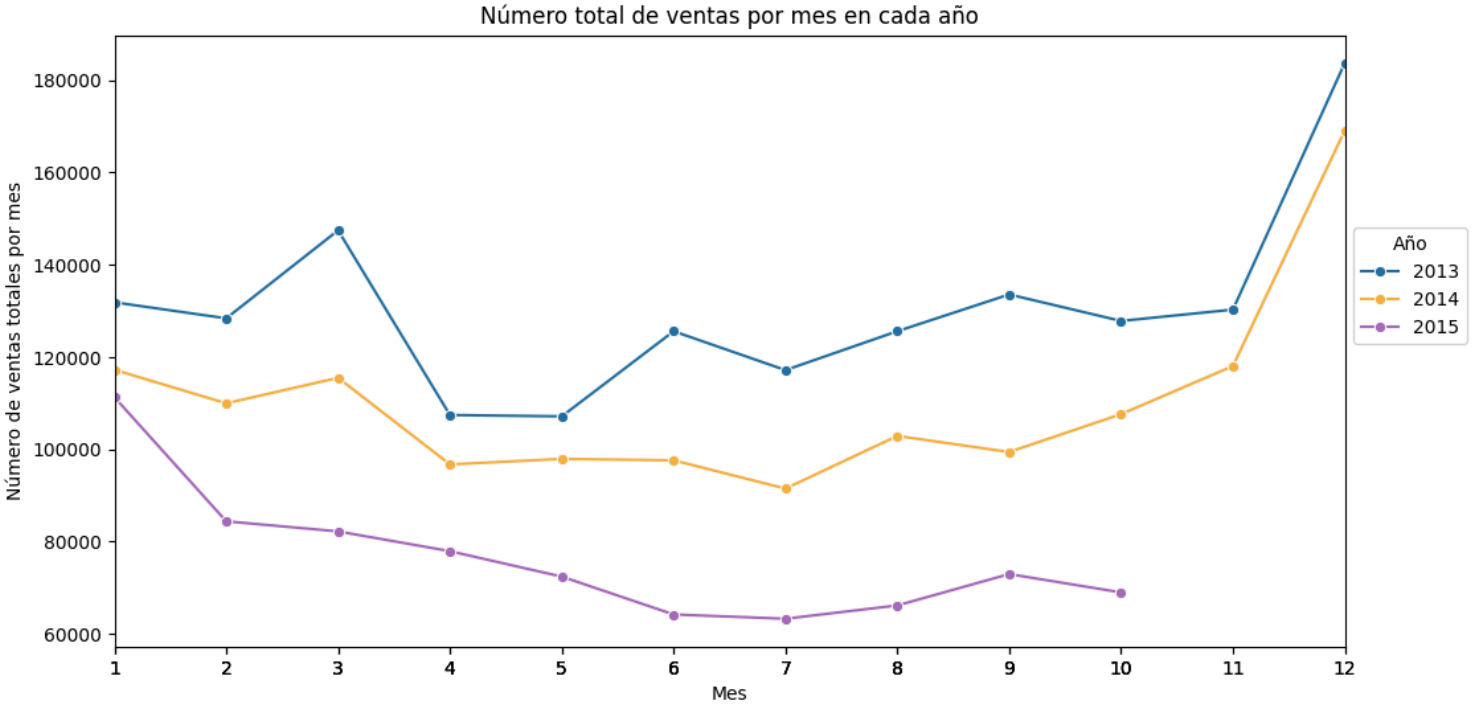
Gráfico 4: Tendencia Mensual del Precio Promedio
Identificar la tendencia de los precios a lo largo del tiempo ayuda a entender cómo fluctúan las tendencias de las ventas.



Se puede ver que cuando se tiende a fines de año y época de fiestas los precios de venta se disparan. Es algo que se repite a fin de año de 2013 como también de 2014. Esto podría marcar que durante estas épocas se da esa subida debido a un crecimiento en el número de ventas y el accionar de los vendedores que tienden a subir los precios para aprovechar ese escenario.

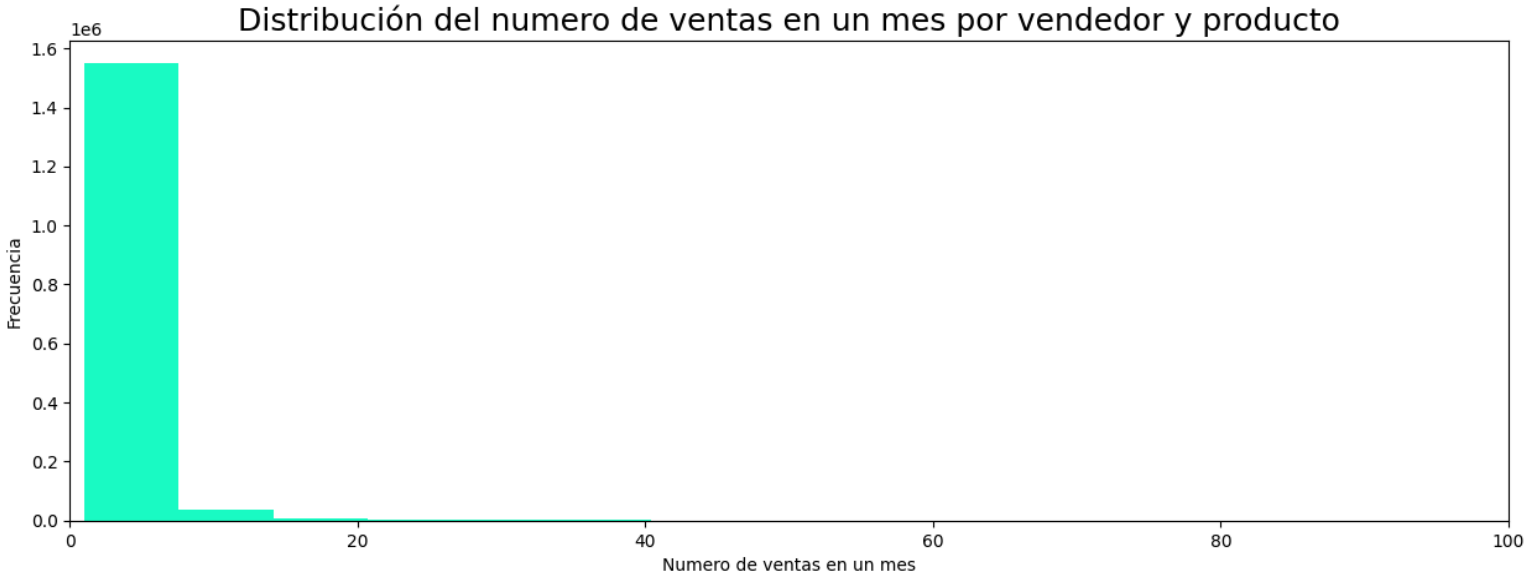
—----------

Gráfico 5: Número total de ventas por mes en cada año
Comparar las ventas mensuales de diferentes años ayuda a identificar patrones estacionales y anuales. También se confirma lo que esperábamos del plot anterior. Podemos notar como tanto en 2013 como 2014 la tendencia es que a partir de noviembre las ventas se disparan, probablemente debido a las festividades de fin de año.



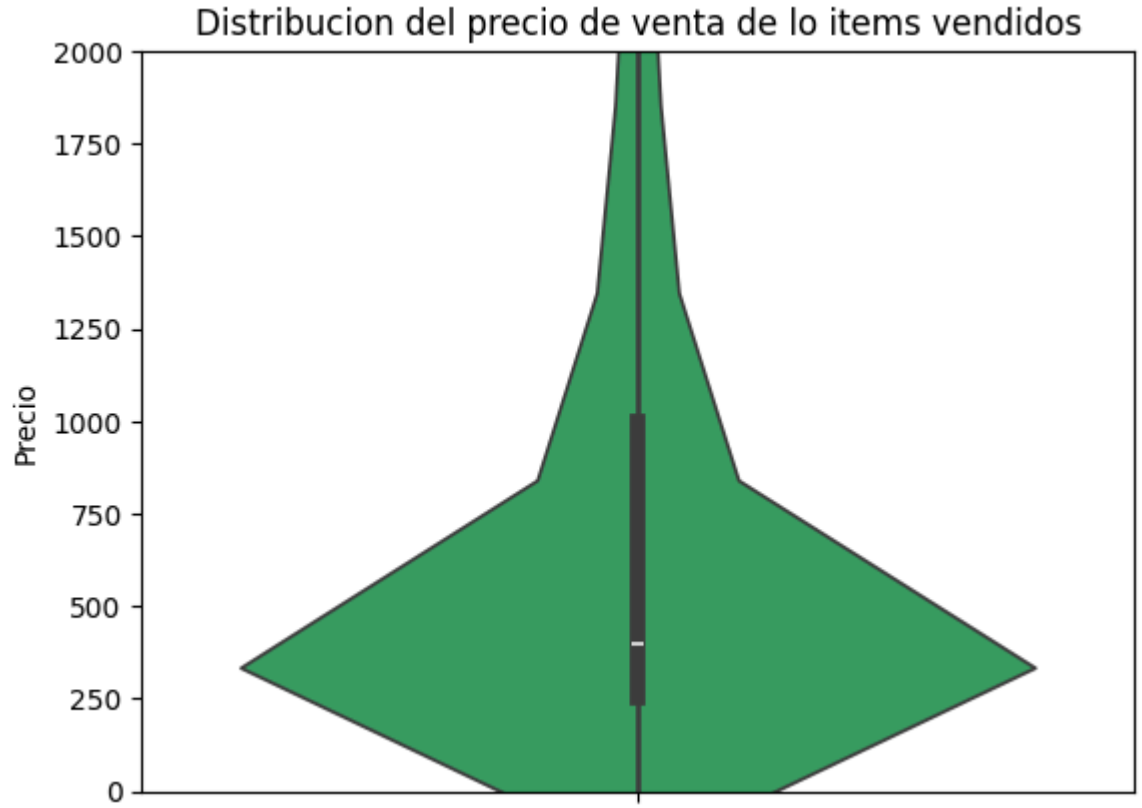
Con esto podemos intuir que los números de ventas que vamos a predecir con los modelos van a ser mayores respecto a los de los meses anteriores a noviembre del año 2015.

Gráfico 6: Distribución del número de ventas mensuales por vendedor y producto
Entender la distribución del número de ventas mensuales por vendedor e item nos da una idea de donde se va a encontrar el rango de los valores de ventas por mes que vamos a predecir en los modelos.



Como se puede apreciar por muchísima diferencia la mayoría de pares vendedor ítem tiene por mes un número muy bajo de ventas rondando valores menores a 10. Con esto podemos tener una idea de que los valores que obtengamos de predicción muy probablemente estén en ese rango de cifras.

Gráfico 7: Distribucion del precio de venta de lo ítems vendidos.
Ver la distribución de los precios nos permite saber que precio ronda la amplia mayoría de los ítems vendidos.



Aca podemos ver que la mayoría ronda un precio entre 250 y 500 lo que significa que si los precios de los productos para los que se va a predecir su número de ventas mensual tienen un precio que ronde este rango es esperable que se mantenga su número de ventas.

Definiendo set de entrenamiento, validación y test

Como los datos son una serie temporal, osea están ordenados cronológicamente y guardan una relación de tiempo, el realizar una partición aleatoria usando el método de sklearn train_test_split rompería esta dependencia temporal. Hacer uso de esta herramienta podria llevar a que el modelo aprenda patrones incorrectos, ya que por ejemplo se podria dar un escenario en el que el modelo se entrena con datos futuros y deba predecir datos pasados lo cual es incorrecto y se conoce como ‘time traveling’.
Teniendo en cuenta esto para abordar adecuadamente la separación de del set de datos, separamos la información en un punto temporal específico. Teniendo en cuenta que se recomienda por lo general un proporción de 80% o 85% para el set de entrenamiento y el

20%-15% restante para validación, usamos los datos hasta de Abril de 2015 para el set de entrenamiento de los modelos y los datos después de Abril hasta Octubre inclusive para el set de validación.
Así quedan finalmente definidos X_train y X_validation nos quedamos con todas las columnas menos item_cnt_month. y_train e y_validation solo consisten de la columna item_cnt_month c/u con los datos del periodo de tiempo correspondiente.
Ahora toca preparar el X_test, lo que hicimos fue agarrar el dataset **test.csv** e intentamos igualarlo a las columnas de X_train, asignándole el date_block_num = 34, mes en Noviembre (número 11) y año en 2015.
Con respecto a los precios de los ítems como no habia informacion, tomamos la decisión de llenarla con el valor promedio de item_price para cada ítem dentro de **sales_train.csv**, lo mismo hicimos con total_sales_item, nos pareció una mejor medida que llenar con ceros.
En los que sí nos parecía que no había otra opción más que llenarlos con ceros son las columnas 'day', 'item_cnt_day' así como también los días encodeados con OneHotEncoding de '0' a '6'. Sacamos las columnas que estában de más como resultado de los merges como 'shop_name' e 'item_name'. También sacamos 'ID' que no era necesario.
No agregamos item_cnt_month por porque eso es justamente lo queremos predecir.

Ya preparados todos los sets de datos finalmente los guardamos en CSVs y ya están listos para ser usados para entrenar los modelos.

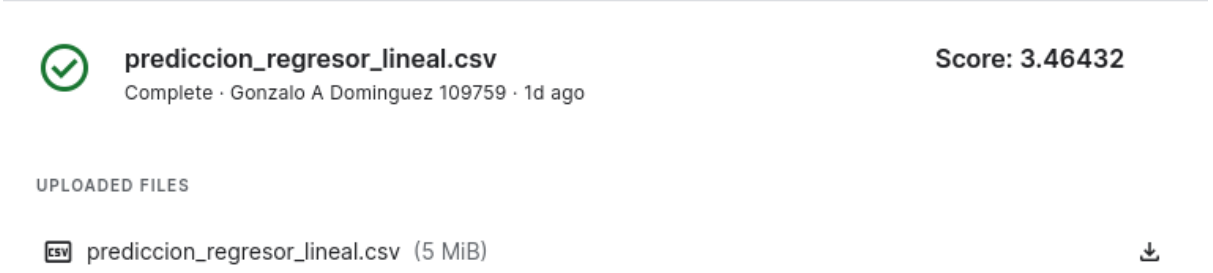
Modelos

Regresor Lineal como Baseline regresor_lineal(baseline).ipynb

Como punto de partida implementamos un regresor lineal, como resultado nos dio el

- **RMSE de set de entrenamiento de:** `4.919432`
- **RMSE de set de validación de:** `4.578861`
- **Al entregar en Kaggle nos dio como resultado un RMSE de: 3.46**

Finalmente generamos y_test prediciendo con este regresor lineal limitando los valores entre cero y veinte y los entregamos.
En kaggle el resultado fue el siguiente:



Modelos con búsqueda de hiperparametros, consideraciones

Durante el entrenamiento de los modelos se utilizó el método RandomizedSearchCV de la librería sklearn. El método de búsqueda de hiperparametros RandomSearch al probar de manera aleatoria a lo largo de una serie N de ciclos distintos combinatorias de valores para los hiperparametros resulta más rápido que GridSearch.
Si en cambio hubiésemos utilizado GridSearch por ejemplo si tenemos 10 hiperparámetros, cada uno con 10 posibles valores, se tendría que evaluar 10^{10} combinaciones, lo cual implicaría mucho tiempo, si es cierto que obtiene la mejor combinatoria.
Sin embargo, RandomizedSearchCV hace cross validation y esto puede tener un impacto negativo en los modelos ya que los datos guardan una relación temporal. Es por ello que para hacer cross validation hicimos uso de TimeSeriesSplit, el cual también forma parte de sklearn, para que de esa manera RandomizedSearchCV pueda usar cross validation pero de forma que se garantice que se va respetar el orden temporal entre los datos.
En cuanto a ser reproducibles, configuramos el random_state para que se pueda correr el notebook varias veces sin afectar el resultado.

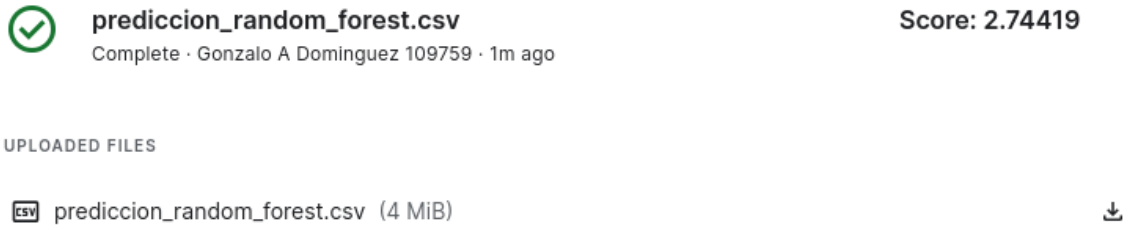
Modelo 1: Random Forest con búsqueda de hiperparametros

random_forest.ipynb

Implementamos RandomForest como nuestro primer modelo con búsqueda de hiperparametros.
En cuanto a los parametros tuvimos en cuenta las siguientes cosas:
n_estimators = [5, 10, 15]: Se eligieron valores pequeños para hacer una búsqueda inicial rápida y evitar tiempos de entrenamiento prolongados cosa que sucede en random forest con valores grandes de este hiper parámetro.
max_depth = [None, 5, 10]: Este parámetro define la profundidad máxima de cada árbol. Con None permitimos la posibilidad de tener un numero ilimitado en profundidad, mientras que queríamos ver qué pasaba con valores limitados que ayuden a evitar el (overfitting) sobreajuste.
min_samples_split = [5, 10]: Valores más altos pueden ayudar a prevenir el sobreajuste al requerir que los nodos tengan un número mínimo de muestras antes de dividirse.
min_samples_leaf = [1, 2]:Un valor de 1 permite hojas con una sola muestra, lo que puede llevar al sobreajuste. Al probar con 2, cada hoja a tener al menos dos muestras.

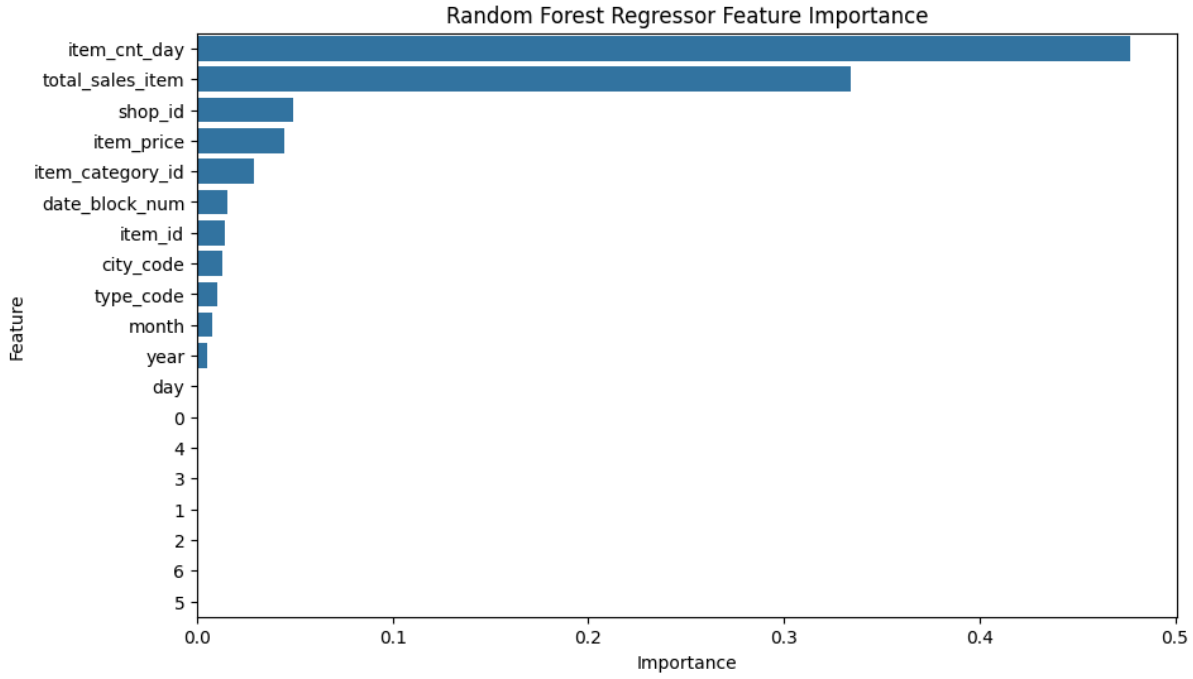
Nos dieron como resultado:

- **RMSE de set de entrenamiento de:** `3.452478`
- **RMSE de set de validación de:** `3.600107`
- **El modelo tardó: 25 Minutos**
- **Al entregar en Kaggle nos dio como resultado un RMSE de: 2.74**



Análisis de Feature Importance de Random Forest

En el caso de random forest el feature importance se mide con el coeficiente de Gini, tanto ventas diarias como total de ventas tuvieron más ponderación durante el entrenamiento del modelo, después de estos le siguen los id de las tiendas, el precio y la categoría del ítem. En cambio, los días de la fecha y días de la semana encodeados por OneHotEncoding, no tuvieron importancia alguna.



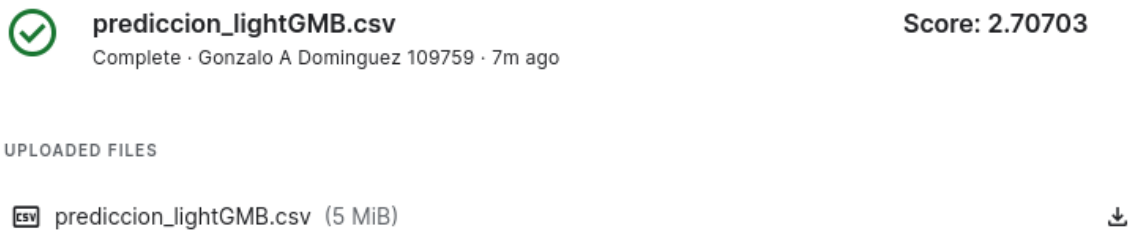
Modelo 2: LightGBM con búsqueda de hiperparametros más Avanzado lightGMB.ipynb

Implementamos LightGBM como nuestro segundo modelo con búsqueda de hiperparametros. Decidimos indagar más aún y probar con más hiperparametros:
'n_estimators': [100, 125, 150] :Configuramos estos valores por arriba del numero basico que es 100 para obtener mejores resultados.
'num_iterations': [75, 100, 150]:Probamos con un rango que incluye un valor un poco más bajo (75) permite ver si el modelo puede tener buenos resultados con menos iteraciones, lo

que podría reducir el tiempo de entrenamiento. Y 150 que es bastante mas que el valor base de 100.
'feature_fraction': [0.5, 0.7, 1.0]: Probamos con fracciones de 50% a 100% a identificar si el uso de casi todas las características en cada iteración es beneficioso o si es mejor utilizar solo una parte de ellas para evitar el sobreajuste.
'max_depth': [-1, 5, 10, 15]: Usamos -1 ya que permite una profundidad ilimitada, mientras que valores de 5, 10 y 15 prueban diferentes niveles de profundidad para controlar la complejidad del modelo.
'num_leaves': [20, 31, 37, 45, 50]: Usamos desde valores más conservadores (20) hasta más altos (50), para ver cómo afecta la capacidad del modelo para ajustarse a los datos.
'min_data_in_leaf': [10, 15]: Utilizamos valores relativamente pequeños, pero que aseguran que cada hoja tenga un número mínimo de muestras suficiente para evitar sobreajuste.

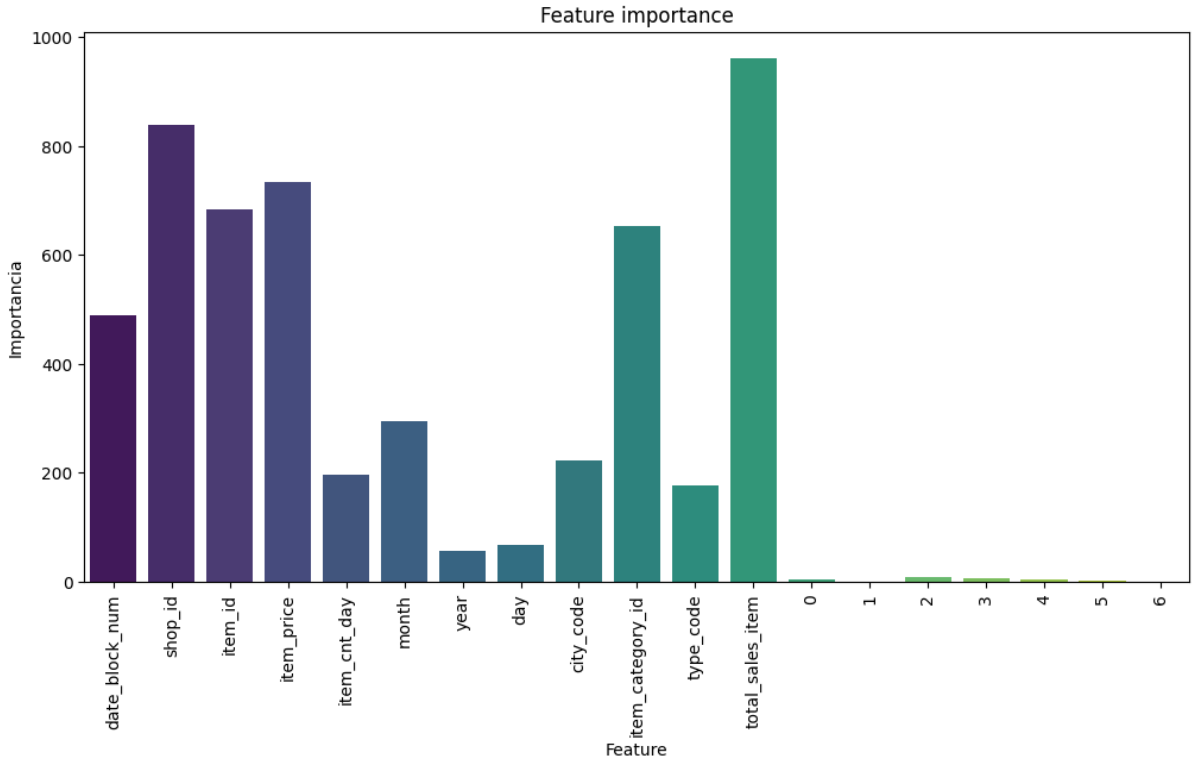
Nos dieron como resultado:

- **RMSE de set de entrenamiento de:** `2.881115`
- **RMSE de set de validación de:** `3.281104`
- **El modelo tardó:** 16 minutos
- **Al entregar en Kaggle nos dio como resultado: 2.70**



Análisis de Feature Importance de LightGBM más Avanzado

En este modelo también se considera a total de ventas como una feature importante durante el entrenamiento del modelo, a diferencia de Random Forest, ventas diarias no fue el feature más significativo, en cambio está como segundo puesto el id de las tiendas. Después de estos le siguen el precio y el id de los ítems.
En este caso, Light GBM pudo aprovechar los features encodeados por One Hot Encoding y darle una importancia para predecir el target.



Conclusión

Construimos un modelo base utilizando regresión lineal, y entrenamos dos modelos avanzados, incluyendo LightGBM y Random Forest ambos con búsqueda de hiperparámetros.
En cuanto a rendimiento el mejor fue LightGMB no solo por el rmse logrado sino que también por el relativamente bajo tiempo de ejecucion requerido.

El mejor score en la competencia hasta el momento es de 0.73726 en términos de RMSE. Nuestro mejor modelo, utilizando LightGBM optimizado, alcanzó un RMSE de 2.70703, buscando “Grupo 6” en el leaderboard de la competencia de kaggle nos podrán encontrar.