Prácticas que vi en el notebook que creo que deben ser corregidas

1. **Doble for loop para reemplazar datos categóricos a numéricos en un dataframe:**
   1. La idea es buena, convertir datos categóricos a datos numéricos, pero se puede realizar de una manera más eficiente con .map() o .replace(). Basta con hacer un diccionario de los elementos a reemplazar y usar el .map() en cada columna, así evitamos el doble for loop innecesario e ineficiente.
2. **Redundancia en los coeficientes de correlación:**
   1. Se muestran los coeficientes de correlación tanto en una tabla con los puros datos numéricos y como heatmap. Yo creo que basta con usar el heatmap.
3. **Dividir el dataset por ciudades:**
   1. No hay sentido en dividir el dataset por ciudades, sólo a menos que se haya pedido crear modelos predictivos por ciudad. Esto sólo crea redundancia y alta cardinalidad; al modelo deseado final se le hará difícil generalizar.
4. **No probar distintos modelos de clasificación:**
   1. Tal vez se realizó, pero al menos en el notebook no se nota que haya probado distintos modelos de clasificación que sea el más adecuado para este problema.
5. **Imputar valores sin intentar ver por qué hay valores nulos.**
   1. Hay columnas que son altamente correlacionadas al target feature, pero tienen la característica de que son columnas con altos niveles de valores nulos. El científico tomó la decisión de imputarlos con la media o moda dependiendo del tipo de datos que sea la columna. Pudo haber buscado el porqué un buen porcentaje de datos está faltando. Por ejemplo, estuve viendo por fechas, pero no parece haber un patrón; son valores probablemente faltantes al azar.
   2. También pudo haber utilizado algún modelo inmune a los datos nulos en lugar de afectar la varianza de los datos, como XGBoost, que es lo que se realizó en la refactorización.
6. **Falta de comentarios dentro de las celdas de código.**
   1. Los comentarios que se hacen a veces se encuentran incluso en celdas de código en el mismo notebook, ni siquiera poniendo el # o ‘’’ ‘’’ para hacerlos comentarios y clarificar mejor las líneas de código.
7. **Importe de librerías sin usar.**
   1. Cargar librerías ocupa espacio en la memoria RAM, un recurso indispensable para el procesamiento de datos. Se encontró que se importaron librerías sin usar y esto es ineficiente.

Notas

* Tomar el feature de *location* es valioso para hacer un modelo robusto, pero requiere de más investigación para lidiar con el problema de alta dimensionalidad (tiene 49 ciudades distintas y hacer one-hot encoding aumentaría demasiado la cardinalidad del dataset), como hacer clústers de ciudades o tomar mejor sus altitudes y latitudes. Me salté este paso porque no viene al caso con el objetivo del challenge. Sólo se tomó en cuenta el mes del feature *date*.
* Después de hacer LazyPredict se determinó que el mejor modelo para esta tarea es el XGBoost Classifier.
* Con los datos nulos, se imputó con la media en las columnas. No tuvo mucha mejoría, pero de todos modos se dejaron los datos imputados.
* Se revisó con el feature\_importance de XGBoost si vale la pena quedarse con todas las columnas o si se pueden remover. Encontré que dejar todas las columnas conlleva al mejor desempeño del modelo tanto en accuracy como en f1 score.

Estructura del proyecto

Elegí esta estructura de organización de folders para tener un mejor orden en el proyecto:

MLProject

├── data

│ ├── processed

│ │ └── data\_preprocessed.csv

│ └── raw

│ └── weatherAUS.csv

├── models

│ ├── model1.bin

│ └── ...

├── notebooks

│ ├── EDA\_y\_Desarrollo.ipynb

│ └── modelo\_DS.ipynb

├── notes

│ └── notas\_y\_correcciones.docx

├── src

│ ├── data\_splitting.py

│ ├── main.py

│ ├── preprocess\_data.py

│ └── train.py

├── .gitignore

├── modelo\_DS.zip

├── README.md

└── requirements.txt

¿Cómo interactúa la estructura de las carpetas con el ambiente productivo?

Mediante el manejo de una terminal, se puede correr el archivo main.py que se encuentra en el folder src utilizando la librería *argparse*, de tal manera que no es necesario abrir los scripts o notebooks y correrlos desde un editor de código. Así se pueden correr experimentos o jobs de una manera más práctica y eficiente.

Este archivo main.py toma otros scripts como el de preprocess\_data.py, train.py, y data\_splitting.py como modularización del pipeline de un ciclo de vida de MLOps. Esto permite debuggear fácilmente, y se podrían correr estos archivos por separado usando un “if \_\_name\_\_ = ‘\_\_main\_\_’”.

Algo importante también en un proyecto de MLOps es el monitoreo de los resultados de los experimentos antes de mandarlos a producción. Para esto utilicé la herramienta WandB, la cual nos ayuda a llevar un recuento de los experimentos, y se puede ver aquí 4 modelos que corrí con diferentes parámetros: <https://wandb.ai/mario-alan-16180/WalmartChallenge>. No sólo sirve para el momento del entrenamiento sino también para verlo en una etapa en producción.

Además, hay una carpeta dedicada para guardar los datos procesados, así como también hay una carpeta que guarda los notebooks para hacer análisis exploratorio de los datos, una etapa importante. Los modelos se guardan en su carpeta dedicada también, y en el folder principal se muestran los requerimientos para poder correr el archivo main.py.