Lista de Pre Procesamientos

Nombre	Descripción	Función Python
KNN Imputer	Completado de missings utilizando el modelo k-Nearest Neighbors. Cada missing es completado con el promedio de los valores de los n-vecinos	knn_imputer
Mean Imputer	Completado de missings utilizando la media de los datos en cada feature	mean_imputer
One Hot Encoding (OHE)	Para cada feature categórico crea x-1 columnas booleanas siendo x la cantidad de categorías del feature	one_hot_encoding
Hashing Trick	Aplica una función de hash a las categorías de los features categóricos	hashing_trick_encoding
Binary Encoding	Se aplica a los features categóricos, funciona convirtiendo cada categoría a una tira de bits	binary_encoding
Standard Scaler	Escala los datos de forma tal que los features queden con esperanza cero y varianza uno	standar_scaler
Robust Scaler	Escala los datos de forma tal que sea robusto ante puntos outliers	robust_scaler

Lista de Modelos

Modelo	Pre Procesamiento	AUC ROC	Accuracy	Precisión	Recall	F1 Score
Random Forest	Knn Imputer - Standar Scaler - OHE - Hashing Trick	0.71	0.84	0.74	0.45	0.57
Random Forest	Knn Imputer - Standar Scaler - OHE - Binary Encoding	0.70	0.85	0.74	0.45	0.56
Random Forest	Mean Imputer - Robust Scaler - OHE - Binary Encoding	0.71	0.85	0.76	0.46	0.57
Árbol de Decisión	Knn Imputer - Standar Scaler - OHE - Hashing Trick	0.70	0.83	0.70	0.46	0.55

Árbol de Decisión	Knn Imputer - Standar Scaler - OHE - Binary Encoding	0.70	0.83	0.70	0.46	0.55
Árbol de Decisión	Mean Imputer - Robust Scaler - OHE - Binary Encoding	0.69	0.84	0.72	0.43	0.54
Naive Bayes	Knn Imputer - Standar Scaler - OHE - Hashing Trick	0.68	0.82	0.67	0.41	0.51
Naive Bayes	Knn Imputer - Standar Scaler - OHE - Binary Encoding	0.68	0.82	0.67	0.41	0.51
Naive Bayes	Mean Imputer - Robust Scaler - OHE - Binary Encoding	0.70	0.81	0.59	0.51	0.55
Red Neuronal	Knn imputer - Robust scaler - Hashing Trick - OHE	0.500	0.781	0	0	0
Red Neuronal	Mean imputer - Standard scaler - Binary Encoding - OHE	0.500	0.84	0	0	0
XGBoost	Knn imputer - Robust scaler - Hashing Trick - OHE	0.870	0.850	0.720	0.503	0.593
XGBoost	Mean imputer - Standard scaler - Binary Encoding - OHE	0.872	0.851	0.733	0.500	0.595
KNN	Knn imputer - Robust scaler - Hashing Trick - OHE	0.854	0.834	0.715	0.444	0.548
KNN	Mean imputer - Standard scaler - Binary Encoding - OHE	0.855	0.842	0.737	0.434	0.541

Conclusión

Mirando la **matriz de confusión** de todos los modelos, vemos que los seis tienen la misma tendencia, tienden a predecir 0 sobre 1. Además, sus matrices tienen características muy similares salvo que en la de la Red Neuronal no se predijo casos negativos y la de XGBoost posee la diagonal principal más “saludable” de todas, por lo que concluímos que es el modelo en el cual debemos confiar más.

El modelo que tuvo la métrica AUC-ROC más alta fue **XGBoost** con un valor de 0.872, por lo tanto será el mejor modelo para este problema.
Consideramos que este modelo es el mejor para este problema porque minimiza una función objetivo regularizada (L1 y L2) que combina una función de pérdida convexa (basada en la diferencia entre las salidas previstas y objetivo) y un término de penalización para la complejidad del modelo. El entrenamiento al proceder iterativamente, agrega nuevos árboles que predicen los residuos o errores de árboles anteriores que luego se combinan con árboles anteriores para hacer la predicción final. Se llama gradient boosting porque utiliza un algoritmo de descenso de gradiente para minimizar la pérdida al agregar nuevos modelos, y esto hace al modelo muy eficiente.

Por último, si quisiéramos tener la menor cantidad de falsos positivos, deberíamos seleccionar el modelo que tenga más alta *Precisión*, en nuestro caso sería la **Random Forest** utilizando (Mean Imputer - Robust Scaler - OHE - Binary Encoding) que nos dio 0.76.
Si quisiéramos predecir todos los días que potencialmente lloverán hamburguesas al día siguiente sin preocuparse demasiado si metemos en la misma días que realmente no llovieron hamburguesas al día siguiente deberíamos elegir el modelo que más alto *Recall* tenga, en nuestro caso sería la **XGBoost** que nos dio 0.503.