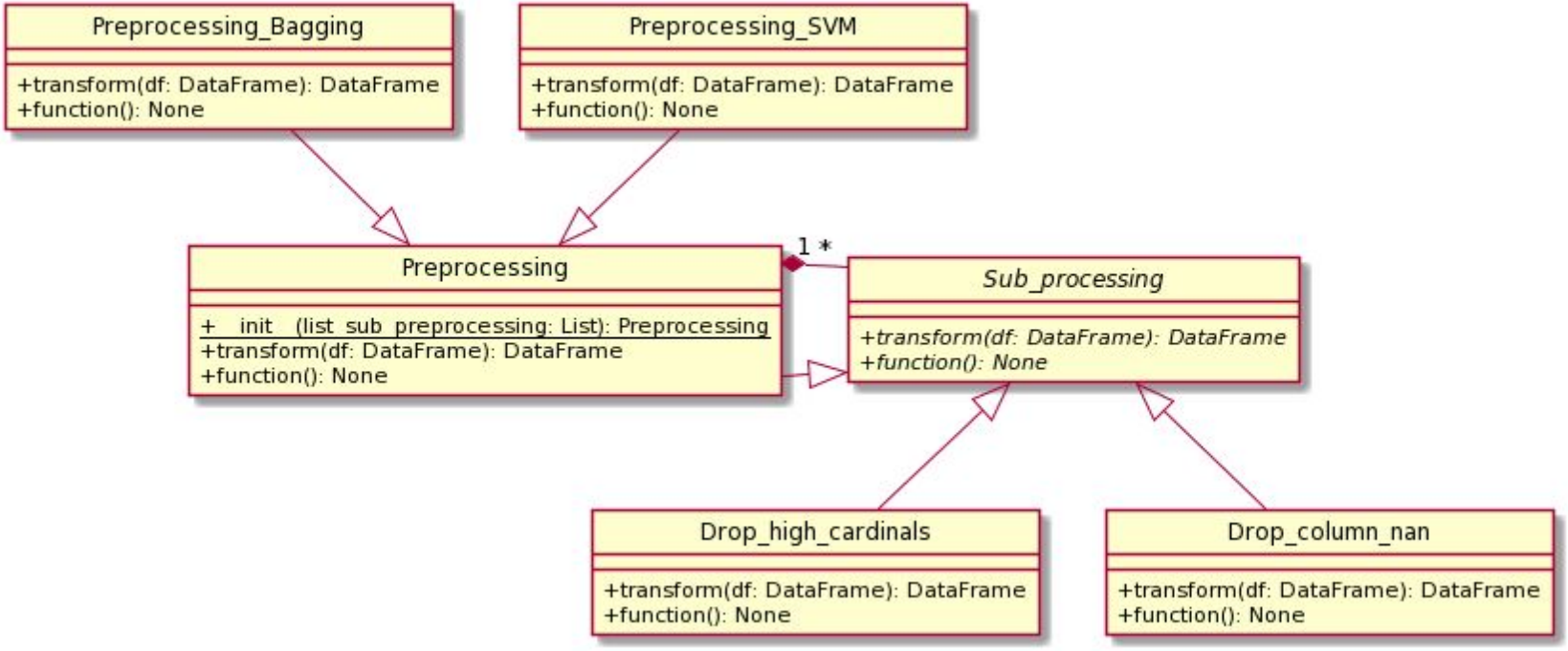
Tablas

Para los pre-procesamientos se siguió un patrón de diseño llamado composite, se dividió todos los pre-procesamientos en tareas más pequeñas llamadas sub preprocessing y luego se fueron uniendo diferentes sub preprocessing formando así pre-procesamientos más complejos.

El diagrama con algunas clases quedaría así:



1. Tabla de sub preprocesamientos:

Nombre del sub preprocesamiento	Explicación simple del funcionamiento	Nombre de la clase de python
Drop high cardinals	Elimina las columnas con alta cardinalidad del df	Drop_high_cardinals
Dummy variables	Transforma las columnas categóricas en dummy variables	Dummy_variables
Drop column nan	Elimina la columna con muchos nan (mayor al 70%)	Drop_column_nan
Nan to zero	Rellena los nan de edad con 0	Nan_to_zero
Nan to mean	Rellena los nan de edad con la media	Nan_to_mean
Nan to mode	Rellena los nan de edad con la moda	Nan_to_mode
Nan to median	Rellena los nan de edad con la mediana	Nan_to_median
Standardizer columns	Estandariza los atributos no binarios	Std_columns
Normalizer columns	Normaliza los atributos no binarios	Normalizer_columns

Se escogieron estos preprocesados por las siguientes razones:

- **Drop high cardinals:**
  Por lo analizado en la parte uno del TP, se concluyó que los features: nombre, id_usuario e id_ticket; además de tener una alta cardinalidad, estos eran identificadores (nombre e id_usuario) o tenían un formato indeterminado (id_ticket) por lo cual estos no aportaban información relevante a la variable target.

- **Dummy variables:**
  Dado que los modelos solo entienden números, las variables categóricas fueron encodeadas de esta forma.

- **Drop column nan:**
  Si se tiene un feature con un porcentaje de valores NaN mayor del 70%, se puede decir que se tiene muy poca información de este dato, por lo cual ignorar esta columna puede ser una buena decisión.

- **Nan to “x”:**
  Como en la feature edad se mostraron pocos valores NaN, con este preprocesado se trata de rellenar esos NaN con algún valor (cero, media, moda y mediana) y ver que respuesta tienen los distintos modelos a estos cambios.

- **Standardizer columns:**
  Dado que se vio en las clases anteriores, existen algunos modelos que funcionan mejor cuando los datos están estandarizados, por lo cual este sub-preprocesado busca hacer eso y ver qué impacto produce en dichos modelos.

- **Normalizer columns:**
  Al igual que el sub-preprocesado anterior; este preprocesado busca normalizar para ver qué impacto produce en los modelos.

2. Tabla de preprocesamientos:

Nombre del preprocesamiento	Explicación simple del funcionamiento	Nombre de la clase de python
Preprocessing Dessicion Tree Classifier and Bagging	Elimina las columnas con alta cardinalidad. Elimina las columnas con un porcentaje de valores nan mayor igual al 70%. Transforma las variables categóricas en dummy variables. Rellena los nan de edad con la media.	Preprocessing_Tree_Bagging
Preprocessing K Nearest Neighbors and Support Vector Machine	Elimina las columnas con alta cardinalidad. Elimina las columnas con un porcentaje de valores nan mayor igual al 70%. Transforma las variables categóricas en dummy variables. Rellena los nan de edad con la moda. Estandariza los atributos no binarios.	Preprocessing_KNN_SVM

Preprocessing Naive Bayes, Boosting and Stacking Classifier	Elimina las columnas con alta cardinalidad. Elimina las columnas con un porcentaje de valores nan mayor igual al 70%. Transforma las variables categóricas en dummy variables. Rellena los nan de edad con la moda.	Preprocessing_NB_Boosting_Stacking
Preprocessing Random Forest Classifier	Elimina las columnas con alta cardinalidad. Elimina las columnas con un porcentaje de valores nan mayor igual al 70%. Transforma las variables categóricas en dummy variables. Rellena los nan de edad con 0.	Preprocessing_RF

3. Tabla de los modelos:

En el caso de Neural Network Classifier no se pudo asegurar la reproducibilidad, por eso no posee un preprocesamiento dado que este varía cada vez que corre el notebook, las métricas son las últimas obtenidas.

Para todos los modelos, lo datos fueron divididos en 3:

- X_train: Para entrenar los modelos mediante cross validation usando un k-fold de 5 y escoger los mejores hiperparámetros usando Grid Search o Random Search dependiendo el caso para un preprocesado.
- X_val: Para elegir el mejor modelo en una familia de modelos con el mejor preprocesado.
- X_val_2: Para elegir el mejor modelo entre todos los modelos hechos.

Nombre del modelo	Nombre del pre procesamiento	AUC-ROC	Accuracy	Precision	Recall	F1
Dessicion Tree Classifier	Preprocessing_Tree_Bagging	0.843226	0.802469	0.826087	0.612903	0.703704
KNN Classifier	Preprocessing_KNN_SVM	0.869355	0.827160	0.869565	0.645161	0.740741
Naive Bayes Classifier	Preprocessing_NB_Boosting_Stacking	0.794839	0.728395	0.695652	0.516129	0.592593
Support Vector Machine Classifier	Preprocessing_KNN_SVM	0.846774	0.839506	0.909091	0.645161	0.754717
Neural Network Classifier	-	0.844194	0.827160	0.793103	0.741935	0.766667
Bagging Classifier	Preprocessing_Tree_Bagging	0.857419	0.839506	0.875000	0.677419	0.763636
Random Forest Classifier	Preprocessing_RF	0.867097	0.814815	0.807692	0.677419	0.736842
Boosting Classifier	Preprocessing_NB_Boosting_Stacking	0.874194	0.839506	0.846154	0.709677	0.771930
Stacking Classifier	Preprocessing_NB_Boosting_Stacking	0.872903	0.827160	0.840000	0.677419	0.750000
Baseline TP parte 1	-	0.802581	0.839506	0.909091	0.645161	0.754717

Ahora se explicará lo más importante de cada Notebook:

Dessicion Tree Classifier:
Para evitar overfitear con este modelo se usó aproximadamente la mitad de los atributos que hay para el rango del hiperparámetro max_depth en el entrenamiento, además se usó Grid Search debido a que eran pocos valores a probar.

KNN Classifier:
Para este modelo se buscó; además de preprocesar como en el modelo anterior; estandariza y normaliza los datos para ver qué impacto produce en el modelo, dando como resultado una mejora significativa.

Naive Bayes Classifier:
En este modelo se vio un rendimiento menor al resto de notebooks hechos, esto puede deberse a que el presupuesto que hace naive bayes respecto a que las variables predictoras son independientes entre sí, tiene un error alto.

Support Vector Machine Classifier:
Al igual que con KNN se preprocesó normalizando y estandarizando los datos para ver qué impacto produce en el modelo, dando como resultado una ligera mejora.

Se pudo ver que este modelo tuvo un score menor que KNN Classifier, esto puede deberse a que en el caso de SVM se usó Random Search debido a la alta cantidad de hiperparámetros a usar obteniéndose así un óptimo local o también puede deberse al dicho de la Navaja de Ockham, que estos datos puedan ser clasificados de forma sencilla y por eso KNN siendo un modelo más simple que SVM se adapta mejor a los datos.

Neural Network Classifier:
Debido a la complejidad de entrenamiento de este modelo se creó una función en la cual se podía añadir las capas, optimizers, regularizers, dropout y funciones de activación que quisieras al modelo; para así poder buscar las mejores combinaciones de hiperparámetros.

Bagging Classifier:
Con este modelo se obtuvo buenos resultados, sin embargo no parece mostrar una mejora significativa, incluso KNN tiene un auc roc score mayor; esto puede deberse a que el dataset obtenido tiene pocos atributos predictores fuertes lo cual lleva a que todos los árboles formados por bagging sean parecidos.

Random Forest Classifier:
Se pudo observar que este modelo tuvo mejores resultados que Bagging lo cual puede indicar que soluciona el problema de los pocos atributos predictores, sin embargo sigue siendo ligeramente más bajo que KNN.

Boosting Classifier:
Se pudo observar que este modelo tuvo mejores resultados que Random Forest Classifier, incluso más alto que KNN, además el resto de las métricas son muy buenas.

Stacking Classifier:
Para este modelo se usó los mejores modelo y los mejores preprocesados de los anteriores notebooks, primero se probó los modelos más básicos (Dessicion Tree Classifier, KNN, SVM y NB) y luego los más potentes (Bagging, Random Forest y Boosting) teniendo como ganador el modelo que usa los más potentes obteniéndose así un buen resultado con las métricas.

Baseline TP parte 1:
Para este modelo se usó dividió los datos como en el resto de las notebooks y se calculó las métricas en base al segundo validation, obteniéndose así un resultado relativamente bueno, lo cual nos lleva a las siguientes:

- **¿Por qué tuvo buenos resultados?**
- **¿Estos resultados son confiables?**

Respuestas en las conclusiones.

4. Conclusiones:

Al hacer este TP se pudo observar que todos los modelos obtuvieron un **Recall** bajo en comparación al resto de las métricas, esto se debe a que el modelo fue escogido a base de la métrica auc roc score y esta métrica busca tener un alto **False Positive Rate** y **True Positive Rate**, donde este último es la **Precision** el cual tiene una relación inversa con el **Recall**.

Al final los mejores modelos resultaron ser:

1. KNN Classifier
2. Boosting Classifier
3. Stacking Classifier

El mejor modelo parece ser el **Boosting Classifier** dado que no solo tiene el mejor **AUC-ROC** score, sino que también tiene el resto de las métricas con un valor alto. Por eso recomendaría el **Bagging Classifier.**

Como se vió en el punto anterior el baseline tuvo buenos resultados y se plantearon las siguientes preguntas:

- **¿Por qué tuvo buenos resultados?**
  Para responder esta pregunta debemos recordar que dicho baseline fue hecho analizando el dataset completo, en otras palabras se puede decir que se entrenó con todos los datos y luego se calculó las métricas en base a una parte de estos; por eso tuvo buenos resultados porque se calculó la métrica con una parte del training, esto da pie a la respuesta de la segunda pregunta.

- **¿Estos resultados son confiables?**
  Como se calculó la métrica con una parte del training, estos resultados no solo muestran lo que aprendió si no también lo que memorizó por lo tanto estas métricas obtenidas son desconfiables. Para poder saber que tanto aprendió el baseline se necesitaría una porción de datos que no estuviesen incluidas en el análisis/training.

A todo esto cabe recalcar que como el baseline fue hecho a mano es posible que tenga un sesgo muy alto debido a mis presupuestos basados en el análisis hecho de los datos a ojo, todo esto hace que no solo métrica calculada sea desconfiable si no que el baseline también lo sea.

En cambio el mejor modelo obtenido, además de tener mejores resultados de las métricas en general, hizo la respectiva separación del dataset por lo cual sus resultados son más confiables que los del baseline, además nuestro mejor modelo final fue hecho usando un ensamble con el cual está demostrado que es mejor a cualquier modelo por sí solo además de tener una base matemática que lo sustenta en el estimador base que es un **Decision Tree Classifier**.