Resumiendo los puntos más importantes del trabajo, el inicio de nuestro análisis se basó en explorar, transformar y visualizar datos.

Para la exploración y transformación de datos buscamos datos nulos que podamos reemplazar por valores más significativos (como *'not company'* para los valores NaN de *company*) y valores atípicos que podamos eliminar o reemplazar por su media dependiendo de su significancia. Estos valores fueron separados en outliers univariados, que encontramos usando Boxplot y Z-Score, y multivariados, que encontramos usando Isolation Forest y Mahalanobis. Además, encontramos valores sin sentido como entradas con cantidad de adultos nula o *adr* negativo.

Para la visualización de datos usamos varios métodos de seaborn y matplot, como por ejemplo bar plots para ver la frecuencia de las variables, histogramas y gráficos de densidad para hallar su distribución, scatter plots para hallar relaciones entre ambas, entre otros. Sumado a esto último, la relación entre las variables fue hallada usando un heatmap hecho a partir de la matriz de correlación de las variables cuantitativas de nuestro set de datos, de esta forma vimos las que más se relacionaban para hacer sus respectivos dispersogramas.

Luego de esta parte introductoria comenzamos a usar árboles de decisión para predecir nuestra variable target (*is_canceled*). Sin embargo, necesitamos que nuestras variables tomen valores numéricos para poder utilizarlos. Esto nos lleva a utilizar dummies para que los valores que toman las variables categóricas de nuestro conjunto sean 0 o 1, pero hacer tal cosa nos trajo un problema, y es que ahora tenemos un número gigantesco de columnas. La solución a esto fue crear una función que calcule la probabilidad de que la entrada tome el valor *is_canceled* teniendo en cuenta el valor de la columna en cuestión, por ejemplo, en vez de tener 149 columnas para cada país y que solo haya un 1 en la columna "Portugal", creamos una sola columna que tome el valor de la probabilidad de que una reserva con valor "Portugal" sea cancelada. El cálculo consiste simplemente en aplicar Laplace al conjunto de datos, por lo que debemos suponer que el conjunto de datos es un espacio equiprobable (lo cual no es así), esto genera cierto error, pero viendo el resultado de nuestras predicciones suponemos que fue un buen *tradeoff* entre calidad de la predicción y tiempo de entrenamiento, ya que este último mejoró drásticamente después del cambio.

En esta parte también comenzamos con el uso de Cross Validation para optimizar los hiper parámetros del árbol. Aquí descubrimos que no nos convenía usar *GridSearch* ya que si bien nos iba a dar el resultado más preciso, el tiempo de optimización iba a ser demasiado grande, por lo que tomamos la decisión de trabajar con *RandomSearch*. Luego de optimizar los hiperparámetros y entrenar el modelo logramos observar cuales eran las variables más importantes para nuestro modelo, las cuales resultaron ser *deposit_type, agent_prob_is_canceled,* si el *customer_type* es *Transient-Party*, *country_prob_is_canceled, lead_time,*

previous_cancellations, entre otras. Para ver todas las correlaciones creamos la función *caracteristicas_importantes*, que recibe el modelo y la cantidad de variables a mostrar.

Finalizada la parte de árboles de decisión comenzamos a utilizar ensambles y analizar el rendimiento de cada uno, sus ventajas y desventajas. Comenzando por K-Nearest-Neighbors, obtuvimos un ensamble balanceado en términos de recall y precisión que generaliza lo suficientemente bien para nuevos modelos, sin embargo, no era tan bueno como el resto de ensambles, esto posiblemente se deba a que KNN es sensible a conjuntos desbalanceados y con presencia de outliers. El ensamble más problemático fue SVM, que a pesar de mostrar buenas predicciones cuando se cambia únicamente su kernel (siendo el más óptimo el radial), su optimización resultó muy tediosa debido al tiempo de entrenamiento necesario, por lo que su matriz de confusión resultó muy desbalanceada y su puntaje F1 bajo en comparación al resto. El tercer mejor modelo que entrenamos fue Random Forest, pero para su entrenamiento se tuvo que asignar un número muy pequeño de iteraciones para evitar tiempos de ejecución inviables. El segundo modelo fue XGBoost, y corroboramos su gran utilidad para evitar *overfitting* nombrada en las clases teóricas. Para finalizar esta entrega, creamos dos ensambles híbridos, Voting y Stacking, y comprobamos que su rendimiento es mejor que el de los modelos que los componen por separado, por lo que cumplen con su función (siendo Stacking el mejor de los modelos entrenados).

Para la última entrega, realizamos tres redes neuronales distintas, en las que probamos optimizar y utilizar distintos regularizadores, como Early Stop, L2, L1 y sin regularizador. Utilizamos para las neuronas de entrada, tantas como la cantidad de características que tiene nuestro dataframe, también usamos cross validation para la optimización de hiperparametros. Como se suponía, el modelo sin regularización realiza un sobreajuste a los datos de entrenamiento, el resto logramos un modelo con resultados similares para test y train.
Además mostramos la matriz de confusión de cada modelo y realizamos diferentes gráficos para entender mejor lo que ocurre en nuestro modelo, como por ejemplo que a partir de las 10 epochs, la optimización que hace el entrenamiento comienza a empeorar, y luego de las 40, ya no es tan relevante seguir entrenando.

Como comentario final del trabajo, queremos decir que nos encontramos varias veces con el problema de la velocidad en la que corre google colab, estuvimos demasiado tiempo esperando que nuestros modelos puedan optimizar hiperparametros, por más que se utilice Random Search y que se pongan pocas iteraciones, se tarda mucho y resulta complicado a veces poder avanzar a un buen ritmo. Para la próxima deberíamos utilizar jupyter desde nuestra propia pc para evitar esto, ya que realmente se hace muy tedioso. Aún así, cabe destacar que nos gustó este trabajo, en donde podemos aplicar los conceptos vistos en las clases y “jugar” con ellos intentando mejorar nuestro modelo.