Trabajo Práctico 1 - GRUPO 09

Nombre	Padrón	Mail
Josefina Marianelli Keichian	106798	jmarianelli@fi.uba.ar
Kevin Alberto Vallejo	109975	kvallejop@fi.una.ar
Lucas Oshiro	107024	loshiro@fi.uba.ar
Brenda Aylas	99575	baylas@fi.uba.ar

Introducción

En este trabajo práctico se realizaron diferentes estudios para diferentes problemas de ciencia de datos, y aplicando los contenidos vistos en la materia, poder resolverlos de forma correcta.

Para ello, se aplicarán técnicas de análisis exploratorio, preprocesamiento de datos, agrupamiento/clustering, clasificación y regresión.

Se nos presentaron 4 problemas, en los que aplicamos las técnicas previamente dichas, dependiendo del problema.

EJ 1 - Análisis Exploratorio de Datos

En este ejercicio, se realizó un análisis completo del conjunto de datos y se aplicaron técnicas de exploración y de preprocesamiento. Para este problema se nos proporcionó un set de datos sobre los viajes que se realizaron para una empresa de taxis, Yellow Cab USA, para los meses Enero, Febrero y Marzo del 2023

En primer lugar, empezamos el análisis con 3 sets de datos, uno para cada mes. A partir de ello, vimos que todos los sets tienen las mismas columnas, exceptuando para el set de Enero que en vez de tener la columna Airport_fee usaba la columna airport_fee. Para solucionar esto, cambiamos el nombre de la columna por el que tenian los otros dos sets de datos. Luego juntamos los tres datasets para tener uno solo con todos los datos.

Ya teniendo todo el dataset completo, algunos de los features mas destacables son: passenger_count, ya que representa la cantidad de pasajeros en el viaje; RatecodeId, ya que representa la tarifa del viaje según el lugar; payment_type, representando la forma de pago del viaje; y por último total_amount, que representa el valor del viaje total.

Asumimos que el total_amount se calcula haciendo la suma de los siguientes features: fare_amount + extra + mta_tax + tip_amount + tolls_amount + improvement_surcharge + congestion_surcharge + Airport_fee

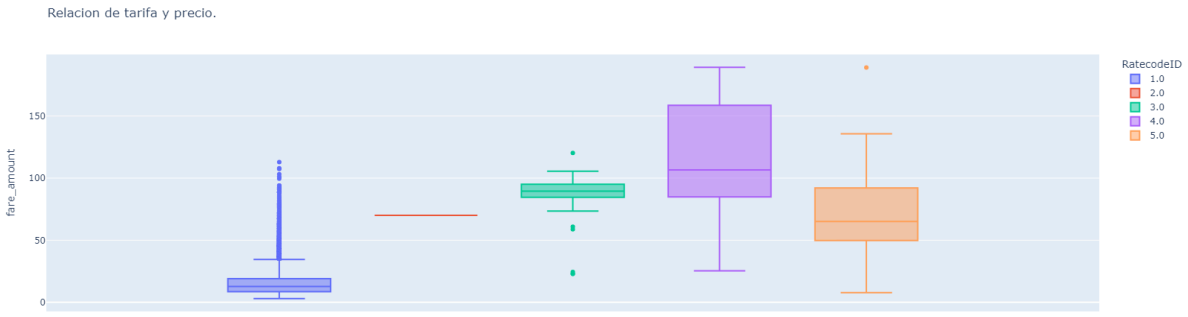
- Preprocesamiento

1) Creamos dos nuevos features que representarán la duración del viaje y la hora de inicio del viaje en horas-minutos, los cuales los calculamos al restar el datetime del fin del viaje con el datetime del inicio del viaje
2) Eliminamos los NaN y los valores con coma para el feature passenger_count ya que no podemos tener mitad de pasajeros. Además dropeamos aquellos valores que sean 0 y mayores a 7 (ya que son pocos valores y su eliminación no modifica el dataset)
3) Eliminamos aquellos viajes que su tiempo de duracion no se relacione con la distancia del viaje, es decir, si un viaje dura mucho tiempo y la distancia es una cuadra, lo eliminaremos
4) Dropeamos aquellas filas que tengan el fare_amount negativo
5) Eliminamos los que tengan VendorId igual a 6 ya que son pocos elementos y no son muy representativos
6) Para aquellos viajes que el total_amount no de igual a la suma de las variables que influyen en él, las cambiamos a la suma de los valores reales de esas mismas variables

- Análisis

Ya que era muy costoso trabajar con aproximadamente 9 millones de filas tomamos la decisión de acortar el df de manera aleatoria a 10 mil filas. Y para otros estudios un df de 700 mil. De esta manera se podía trabajar sin tener problemas de llenar la memoria RAM del Collab.
En la sección de Análisis de Variables se estudiaron la relación entre dos variables de la siguiente manera:

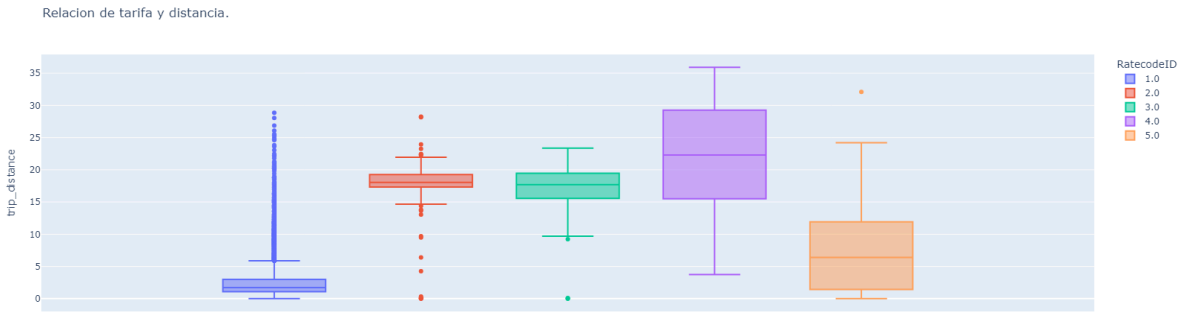
1. Tarifa y precio:



Los viajes más comunes suelen costar entre 8.6 a 20 dolares.
Los viajes al Aeropuerto JFK o alguna de las regiones más alejadas del centro de Nueva York son más caros, todos ellos en promedio de 84.9 dolares.
Todos los viajes de tarifa JFK valen exactamente lo mismo: 70 dolares.

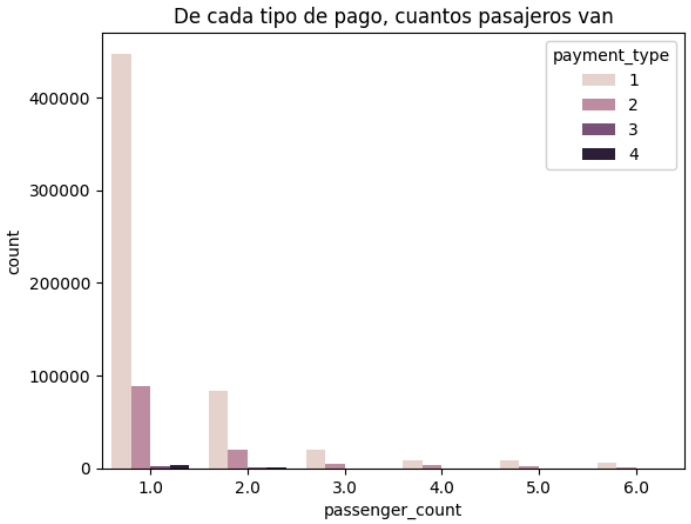
Dato Curioso: Todos los viajes de tipo Tarifa negociada son viajes con precios parecidos a los de las tarifas no ordinarias.

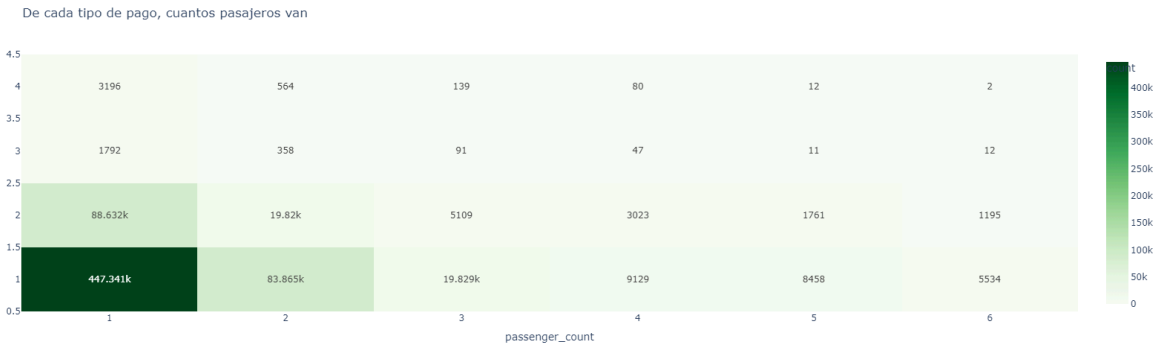
2. Tarifa y distancia



Al parecer los viajes de tarifa negociada no ponen las distancias, no usan el taximetro ya que se acordó el precio con el cliente.

3. Tarifa y duración
   tampoco se usa medición de duración en las tarifas acordadas.
4. De cada tipo de tarifa, cuantos pasajeros van
   Los viajes con tarifa standard y un pasajero son los que mas se utilizaron
5. De cada tipo de pago, cuántos pasajeros van



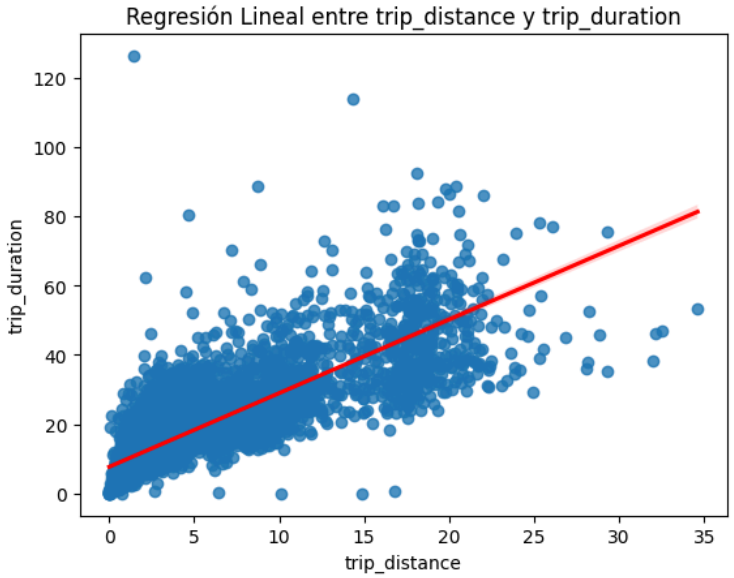


Casi siempre se paga con tarjeta, viajes de 1 persona es por lejos la más usada

Luego el pago en efectivo es el siguiente más usado, sobre todo en viajes de 1 o 2 personas

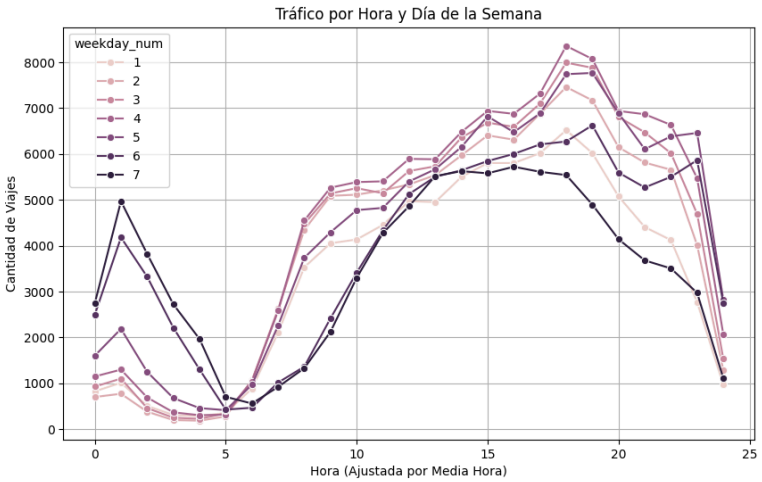
6. Interrelación entre tipo de tarifa, cant. de pasajeros y tipo de pago
   Se observa que la mayoría de viajes son de 1 pasajero, que paga con tarjeta y hace un viaje dentro de la ciudad.
   Sin importar la cantidad de pasajeros los viajes son dentro de la ciudad y en pago de tarjeta.
   Sin importar el tipo de pago la mayoría de viajes son dentro de la ciudad y de pasajero.
   Sin importar a dónde es el viaje, la mayoría son de 1 pasajero que paga con tarjeta.

   En general se mantiene esa jerarquía del tipo 1 al último sin importar el tipo específico donde observemos. Excepto el tipo de tarifa que sigue el siguiente orden: 1, 2, 5, 3, 4.
   Ordinaria, JFK, negociada, Newark, Nassau o Westchester.
7. Tiempo y Distancia
   Se encontró una correlación muy baja: Correlación: 0.066
   Por lo que en el siguiente análisis se hace una limpieza y una vez limpio la correlación tiene sentido:



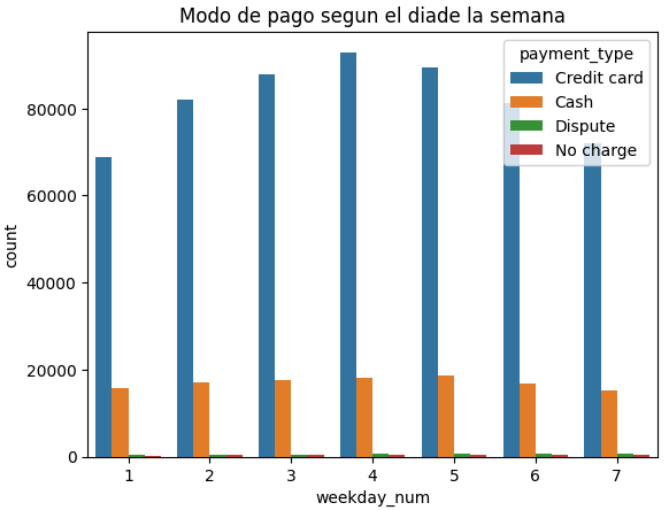
8. Datos con relaciones que hacen ruido
   Viajes en los que no se recorrió ni un centímetro, pero duraron distitnos tiempos y sí se cobró.
   Un 20% de estos viajes son pactados con el pasajero y no usan el taximetro. Y si a eso le sumamos el 15% de no identifcado tipo de tarifa y viajes al aeropuerto tenemos un 35% confirmado que se hizo un viaje no ordinario.
9. Comparando días (Ver los gráficos del Collab)
   En esta sección creamos una variable para cada día.

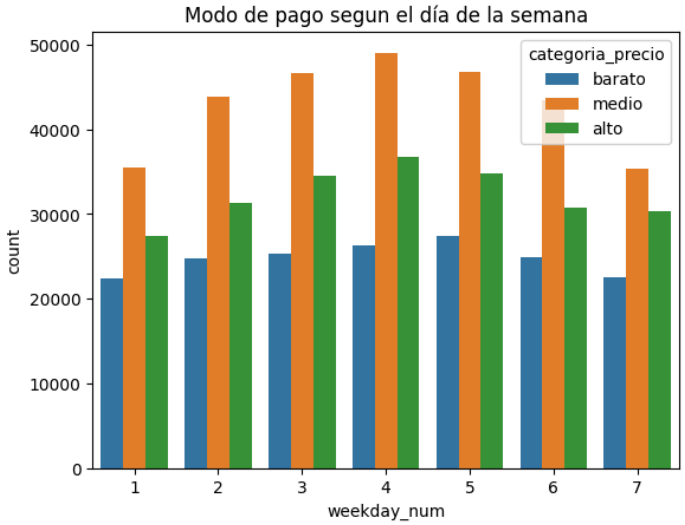
También una variable días_laborales y fin_de_semana
El siguiente gráfico muestra según el día (1=lunes) las horas de más viajes.



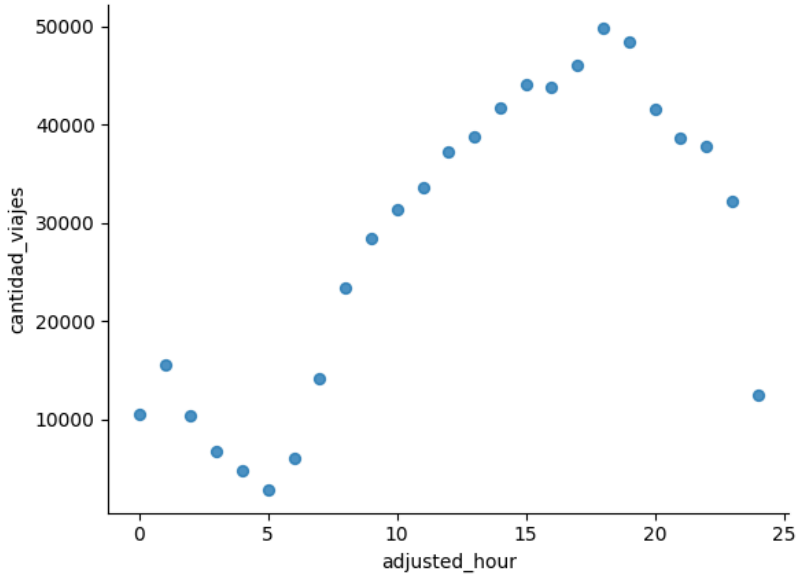
Los domingos se hacen viajes más caros.
La relación de los precios se mantiene sin importar el día.
Contrario a lo que uno esperaría, los lunes es el día con menos tráfico.
El Jueves es el peor día para dar un paseo.

Más gráficos del análisis 9:





Tráfico de taxis según la hora del día



EJ 2 - Clasificación

En este problema, el objetivo es predecir si lloverá o no al día siguiente, teniendo en cuenta los datos de distintas estaciones meteorológicas de Australia. Los datos se obtuvieron en el siguiente link: Weather Dataset. De esos datos, nos encargaremos de estudiar para las locaciones de Nueva Gales del Sur y Victoria.

El dataset originalmente contaba con 23 columnas y 145460 filas. Luego de filtrar por las locaciones indicadas nos quedamos con 54865 registros para poder trabajar. Continuamos con la limpieza de datos y preparación previa para poder utilizar los modelos vistos en clase.

Árbol de Decisión:

Este primer árbol es la base a utilizar por otros métodos. En nuestro caso contábamos con variables categóricas en el dataset original pero sabemos de las clases teóricos que los modelos de tipo árbol no trabajan con textos por lo que tuvimos que transformarlas usando la función de get_dummies, obteniendo así columnas nuevas con ceros y unos; éstas correspondían a la dirección del viento en diferentes momentos del día. De forma similar se trabajó con Location.
Creamos también variables nuevas provenientes de la columna Date que son “season” y “month” ya que nos pareció una buena forma de agrupar registros según características parecidas según el clima correspondiente, sin importar si es el mes de agosto de un año u otro.

Para la optimización de hiperparámetros se utilizaron 5 folds y se trabajó con Random Search CV obteniendo los siguientes resultados:
{'min_samples_split': 7, 'min_samples_leaf': 9, 'max_depth': 7, 'criterion': 'gini', 'ccp_alpha': 0.0} con score F1 de: 0.564.

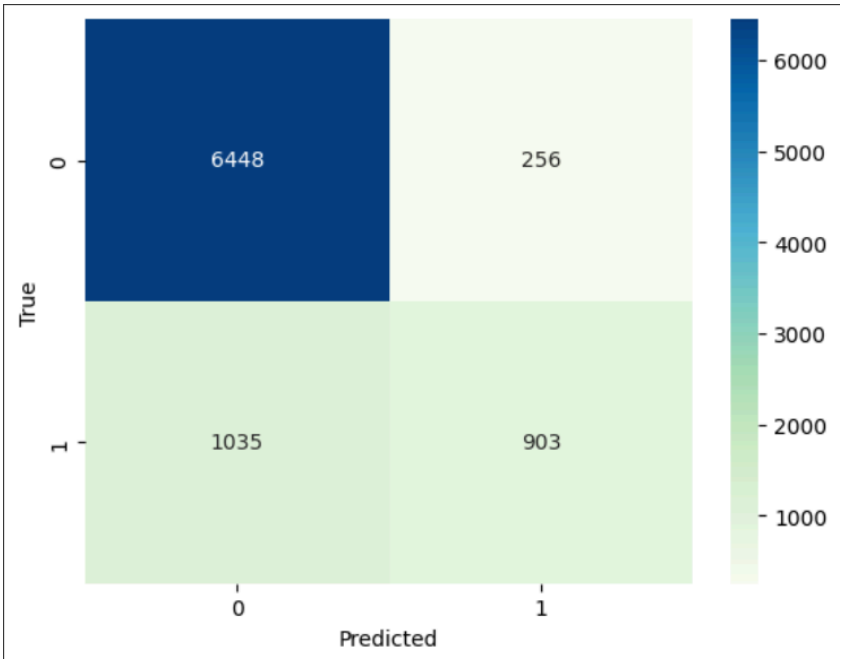
Random Forest:

Se desarrollaron tres versiones del modelo clasificador **Random Forest** con el objetivo de predecir si lloverá mañana (Rain Tomorrow: sí/no). Cada versión fue diseñada para mejorar la precisión y el rendimiento del modelo:

Versión 1.0:

En esta versión, se utilizaron hiper parámetros seleccionados de manera aleatoria, sin un proceso de optimización específico.

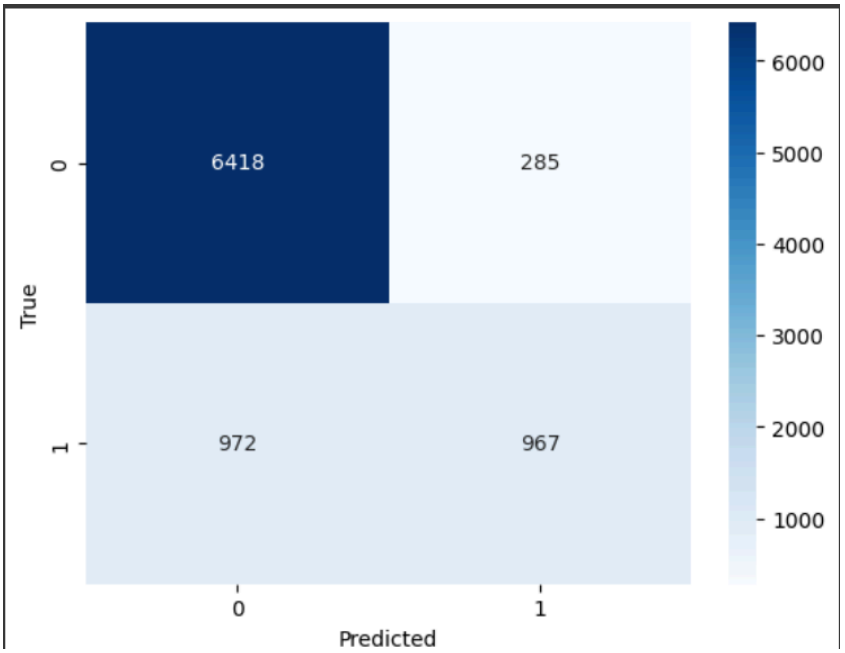
Matriz de confusión:



<u>Versión 2.0</u>:

Para esta versión, se buscó optimizar los hiper parámetros aplicando **Random Search Cross Validation** '**multimétrico**'(en nuestro caso chequeamos con **'accuracy', 'f1' y 'roc auc'**) durante la creación del clasificador. Este enfoque permitió explorar de manera más eficiente el espacio de hiper parámetros y mejorar el rendimiento del modelo.

Matriz de confusión:

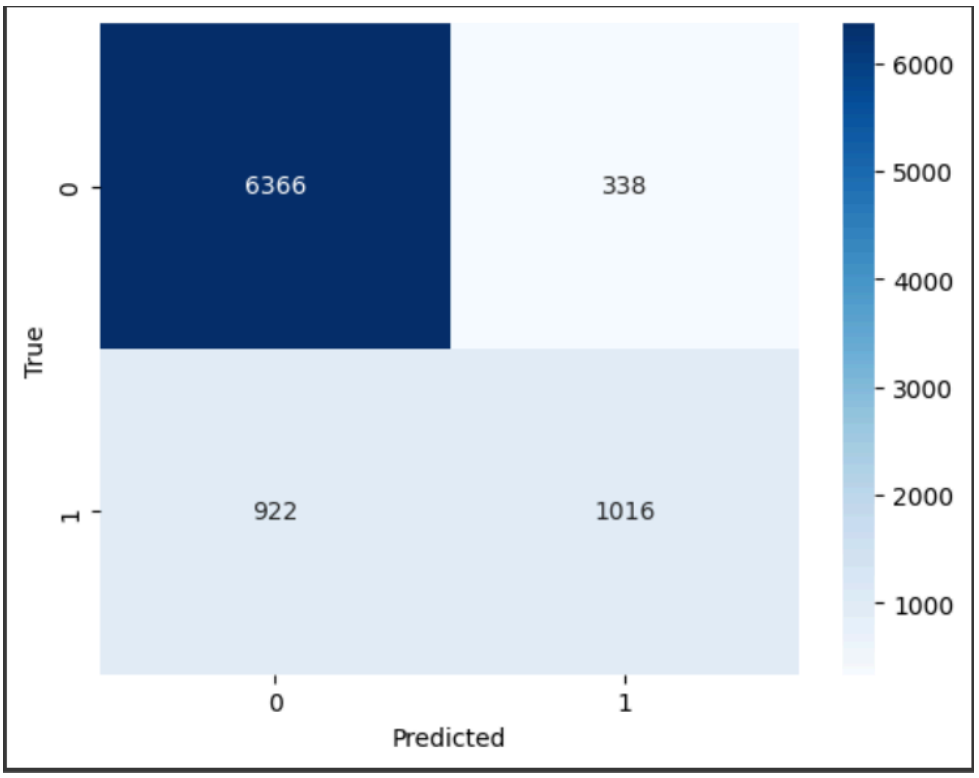


Versión 3.0:

En esta etapa, se llevó a cabo un análisis de la importancia de los atributos. A partir de esta información, se decidió eliminar todas las columnas (atributos) que tenían un peso insignificante en el resultado, conservando únicamente aquellos con mayor relevancia.

Posteriormente, se optimizaron nuevamente los hiper-parámetros utilizando **Random Search Cross Validation** en un enfoque **multimétrico**, evaluando específicamente las métricas de **accuracy**, **F1 Score** y **ROC AUC**. Este proceso permitió crear el clasificador 3.0, buscando mejorar el rendimiento general del modelo.

Matriz de confusión:



A continuación, se presentan los detalles de cada versión del clasificador en los siguientes cuadros:

Modelo	Precisión	Accuracy	F1 Score	Recall
Random Forest 1.0 version	0.77	0.85	0.58	0.46
Random Forest 2.0 version	0.77	0.85	0.60	0.49
Random Forest 3.0 version	0.75	0.85	0.62	0.52

Modelo	Criterion	Min Samples Leaf	Min Samples Split	N Estimators	Max Depth	Max Features	Cross Validation	Folds
Random Forest 1.0	gini	1	2	100	10	sqrt	None	-
Random Forest 2.0	entropy	1	2	100	30	sqrt	Random Search	5
Random Forest 3.0	entropy	1	2	125	20	log2	Random Search	3

En cuanto a los folds seleccionados en las versiones 2.0 y 3.0, la diferencia entre ambas se relaciona con la cantidad de atributos utilizados. En la versión 2.0, se emplearon todos los atributos disponibles, mientras que en la versión 3.0 se trabajó con menos de la mitad, enfocándose únicamente en aquellos atributos considerados "importantes", seleccionados tras analizar el array de importancia de atributos. Esta estrategia buscó concentrar el modelo en las variables más relevantes para la predicción.

Se observa una mejora en las métricas de **F1 Score** y **Recall** al pasar de la versión 1.0 a la versión 3.0. Esta mejora está relacionada con la optimización de los hiper-parámetros aplicada en la versión 2.0, así como con la selección de atributos específicos basados en su importancia, combinada con la optimización de hiper-parámetros en la versión final.

En cuanto a la **Precision**, notamos que las versiones iniciales presentan un puntaje equivalente, mientras que la versión final muestra una leve disminución. Esto podría atribuirse a la reducción en el número de atributos utilizados en la última versión. Por otro lado, el **Accuracy** se mantiene constante a pesar de las diferencias entre las versiones.

Podemos concluir que la optimización de hiper-parámetros mediante **Random Search Cross Validation** tuvo el mayor impacto en las métricas de rendimiento. Aunque la selección de atributos es un factor que puede contribuir a la mejora del modelo, en este caso, el recorte significativo que dejó menos de la mitad de los parámetros originales resultó ser una decisión desfavorable. Un enfoque menos drástico podría haber sido más beneficioso.

Respecto a los puntajes de las métricas, si bien logramos mejorarlos, la desproporción en la cantidad de casos de "Rain Tomorrow" entre las clases "No" y "Yes" limita nuestra capacidad para mejorar aún más estos resultados, dado que no contamos con suficientes datos para la clase positiva.

En conclusión, consideramos que la mejor versión del clasificador Random Forest es la 2.0.

Ensambles Stacking:

Se utilizaron los siguientes modelos base:

knn_clas =KNeighborsClassifier (n_neighbors=1)
rf_clas = RandomForestClassifier(n_estimators=100, random_state=13)
xgb_reg = XGBRFClassifier(random_state=13)

final estimator = RandomForestClassifier(n_estimators=100, random_state=13)

resultados obtenidos con este modelo:

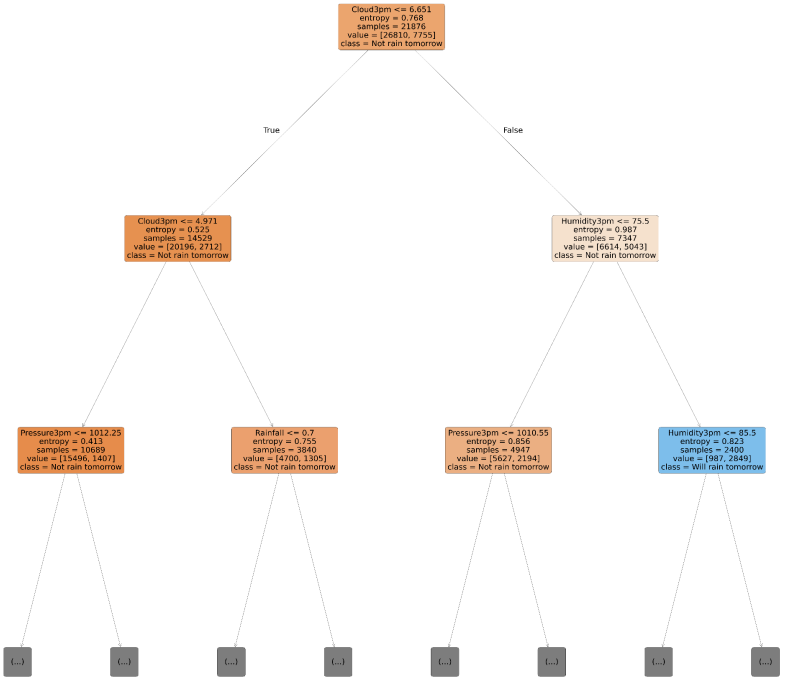
```
Accuracy: 0.8031705623698218
F1 Score: 0.5450655255415887
Precision: 0.5517054683270168
Recall: 0.5385835095137421

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.88      0.87      6750
           1       0.55      0.54      0.55      1892

    accuracy                           0.80      8642
   macro avg       0.71      0.71      0.71      8642
weighted avg       0.80      0.80      0.80      8642
```

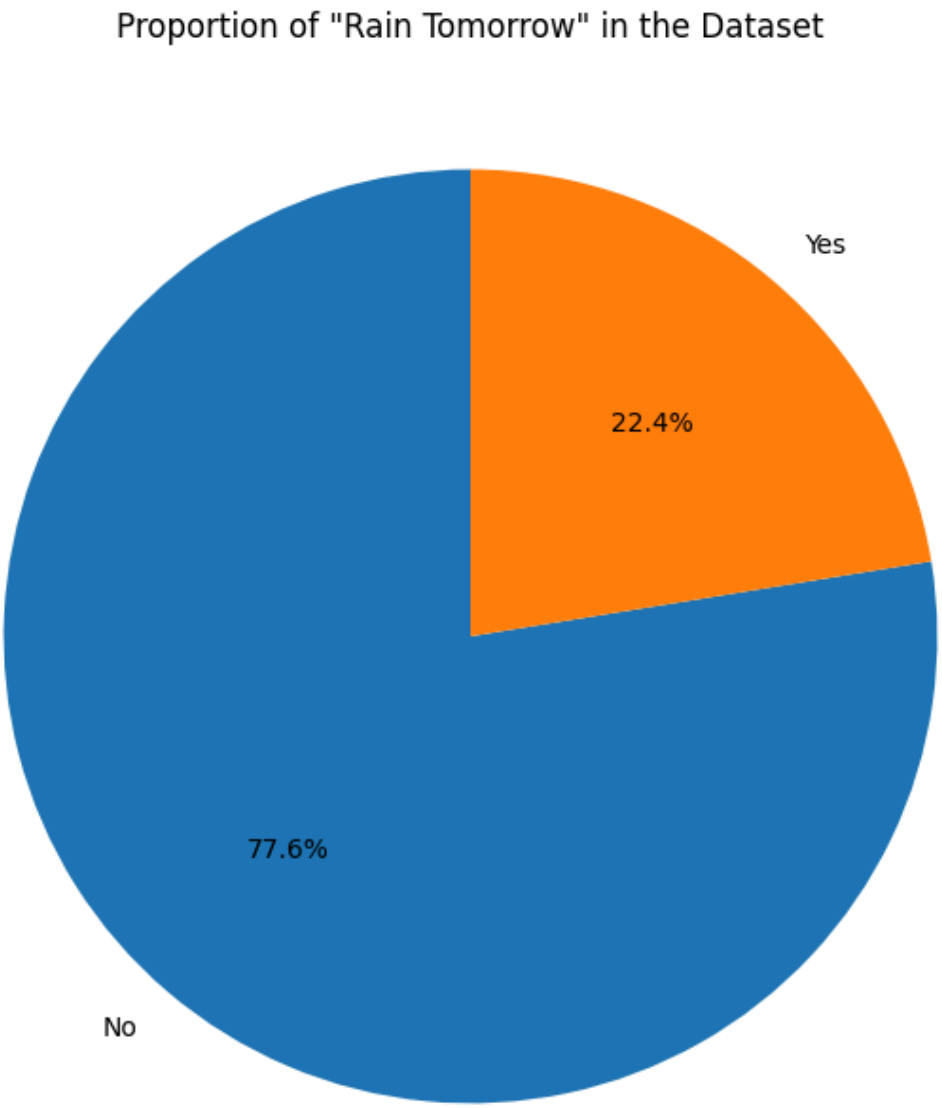
Conclusión ejercicio 2:

Modelo	Precisión	Accuracy	F1 Score	Recall
Árbol de Decisión	0.55	0.80	0.55	0.54
Random Forest	0.75	0.85	0.60	0.49
Ensemble Stacking	0.55	0.80	0.54	0.53

En conclusión el modelo con mejor performance es el de Random Forest (ver 3.0)

Un árbol del Random forest seleccionado:


Cloud3pm <= 6.651
entropy = 0.769
samples = 21876
value = [26810, 7755]
class = Not rain tomorrow
True
False
Cloud3pm <= 4.971
entropy = 0.525
samples = 14529
value = [20196, 2712]
class = Not rain tomorrow
Humidity3pm <= 75.5
entropy = 0.987
samples = 7347
value = [6614, 5043]
class = Not rain tomorrow
Pressure3pm <= 1012.25
entropy = 0.413
samples = 10689
value = [15496, 1407]
class = Not rain tomorrow
Rainfall <= 0.7
entropy = 0.755
samples = 3840
value = [4700, 1305]
class = Not rain tomorrow
Pressure3pm <= 1010.55
entropy = 0.856
samples = 4947
value = [5627, 2194]
class = Not rain tomorrow
Humidity3pm <= 85.5
entropy = 0.823
samples = 2400
value = [987, 2849]
class = Will rain tomorrow
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)



Consideramos que los scores contaban con una mejoría limitada pese a las optimizaciones aplicadas porque contábamos con una mayor cantidad de registros de clase No Rain Tomorrow que Yes Rain Tomorrow. Esto también se refleja en las matrices de confusión de los modelos.

EJ 3 - Regresión

En este ejercicio, nos centraremos en predecir el precio del alquiler de un AirBnB, en la ciudad de Madrid, en función de un dataset con los alquileres que se encuentran en el siguiente link: Datos Alquileres Madrid, utilizando el Detailed Listings Data para Madrid

- Preprocesamiento

1) En primer lugar, nos quedamos con las columnas que nos parecieron más representativas a la hora de predecir el precio del alquiler, según la página de las definiciones de los features, la cual está en el siguiente link, Definicion de features
2) Nos quedamos con: listing_url, host_since, host_response_time, host_response_rate, host_acceptance_rate, host_is_superhost, host_verifications, host_has_profile_pic, neighbourhood_group_cleansed, latitude, longitude, property_type, room_type, accommodates, bathrooms, bedrooms, beds, price, number_of_reviews, number_of_reviews_ltm, number_of_reviews_l30d, first_review, last_review, review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value
3) Luego para las variables accommodates, bathrooms, bedrooms y beds, les eliminaremos las filas con Nan y dropearemos los valores que creemos que son outliers, según el valor mínimo que establecimos para cada uno. Se aplican en estos features porque creemos que son los más importantes a la hora de predecir el precio del alquiler
4) En la mayoría de los casos, los NaN fueron dropeados. En muy pocos casos, los modificamos por un dato que signifique algo dentro de esa variable
5) Luego hicimos un análisis entre el tipo de propiedad con los features de accommodates, beds, bathrooms, bedrooms y fuimos modificando o dropeando según la relación que nos pareció adecuada. Un ejemplo: si una habitación compartida decía que tenía diez baños, nos parecía algo erróneo, por lo que dropeamos esa observación o bien, le cambiamos el valor a uno mínimo que sea adecuado
6) Para el tipo de propiedad, juntamos en grupos a aquellas que nos parecieron muy similares, y a esos grupos les asignamos un valor categórico. Se hizo este mismo paso, para la variable neighbourhood_group_cleansed, solo que en este caso, se asignó un valor que representen la locación

- Modelos

1) Regresión Lineal

Para este modelo, se implementaron varias combinaciones de features para comparar los resultados de las métricas. Como resultado, obtuvimos que las mejores variables a utilizar son: accommodates neighbourhood_group, bathrooms, reviews_scored_location y property_type. A este modelo lo llamamos Modelo lineal múltiple 5.
Las métricas que obtuvimos, son las siguientes:

Métrica	Set de entrenamiento	Set de testeo

MSE	2727.01	3008.63
RMSE	52.22	54.85
R^2	0.46	0.44

Como podemos observar, el error medio para el set de entrenamiento y el set de testeo no varía tanto, a lo sumo 2.63 euros

Para el set de entrenamiento, el R^2 explica un 46% el problema, mientras que para el set de testeo, un 44%.

2) XGBoost

Para el caso de XGBoost, también se usaron varias combinaciones de features para conseguir el mejor modelo. En este caso, las mejores variables a usar son aquellas que tienen mejor correlación con Price, las cuales son: accommodates, bathrooms, bedrooms, beds, neighbourhood_group y property_category_mapped.
Luego, se realizó un k-fold cross-validation con el algoritmo RandomizedSearchCV utilizando varios valores de K. En este caso, el que mejor score (Mean cross-validated score) nos dió fue usando K = 7. Y los hiperparametros que nos dio son:

```
Best parameters:
{'objective': 'reg:squarederror', 'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.1, 'lambda': 1, 'gamma': 100}

Best score:
0.5838344380265295

Number of folds for the best score:
7
```

A este modelo lo llamamos Modelo XGB 1
Las métricas que obtuvimos son:

Métrica	Set de entrenamiento	Set de testeo
MSE	1894.76	2126.83
RMSE	43.53	46.12
R^2	0.63	0.60

Al igual que con el modelo lineal, las el RMSE para este modelo no varió tanto y para el R^2se mantuvieron en valores cercanos, entre 0.63 y 0.60, lo que representa que el modelo explica un 60% de la variabilidad del problema.

3) Modelo a elección con Cross-validation

Para el último modelo, construimos un Random Forest, usando varias combinaciones de features para encontrar el mejor modelo. Al igual que en XGBoost, se utilizó RandomizedSearchCV para buscar los mejores hiperparametros para el modelo. En este caso, se llegó a que el mejor K-fold sería 7. A partir de ello, se obtuvieron los siguientes hiperparametros

```
Best parameters:
{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_depth': 10}

Best score:
0.5771686625705511

Number of folds for the best score:
7
```

Además los features que se utilizaron para este modelo de Random Forest también fueron los que mejor correlación tenían con respecto a la variable Price: accommodates, bathrooms, bedrooms, beds, neighbourhood_group, property_category_mapped. Al modelo lo llamamos Modelo RF 1
Las métricas obtenidas son:

Métrica	Set de entrenamiento	Set de testeo
MSE	1891.72	2116.52
RMSE	43.49	46.01
R^2	0.63	0.60

En este modelo ocurre algo parecido a los anteriores, el RMSE de ambos está en un rango similar, lo cual significa que el error medio del modelo es 46.01. En cuanto al R^2, se obtuvo 0.60, igual valor que para XGBoost.

4) Modelos adicionales

Nosotros decidimos realizar algunos modelos más, y el que más nos llamó la atención es el modelo de Stacking. Para ello, utilizamos como modelos bases alguno de los que obtuvimos con mejores métricas. En este caso, se utilizaron el Modelo XGB 1 y el Modelo RF 1.
Como resultado obtuvimos las siguientes métricas:

Métrica	Set de entrenamiento	Set de testeo
MSE	2001.15	2074.41
RMSE	44.73	45.55
R^2	0.61	0.61

Como se puede observar, en la métrica R^2 se obtuvo el mejor valor de entre todos los modelos que encontramos.

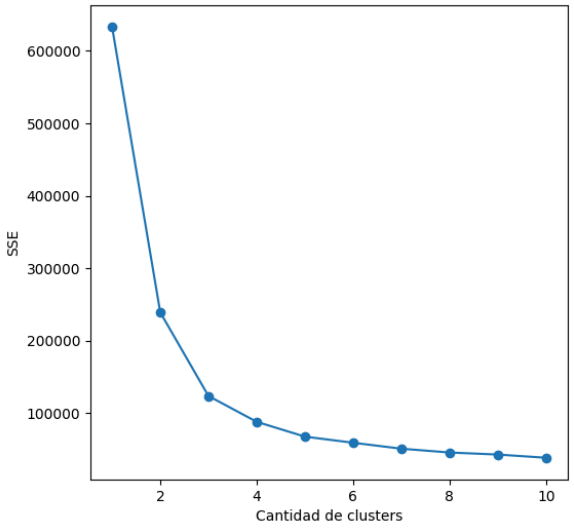
- **Cuadro de resultados**

Métricas para el set de testeo			
Modelo	MSE	RMSE	R^2
Modelo lineal múltiple 1	3008.63	54.85	0.44
XGBoost (Modelo XGB 1)	2126.83	46.12	0.60
Random Forest (Modelo RF 1)	2116.52	46.01	0.60
Stacking	2074.41	45.55	0.61
KNN	2571.14	50.71	0.51

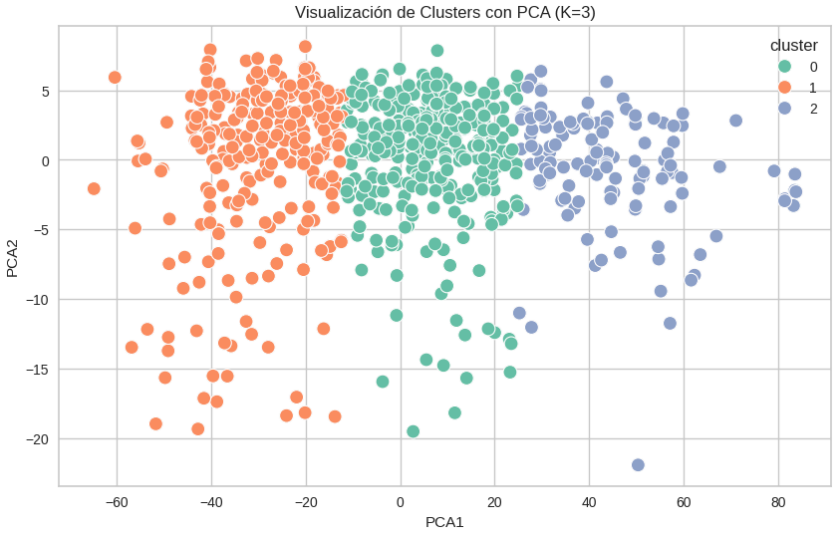
Un dato a tener en cuenta, es que durante las búsquedas del mejor modelo para XGBoost, utilizando todos los features del dataset se obtuvo un modelo que tenía como R^2 = 0.66, el mejor valor de todo el estudio. Sin embargo, nosotros decidimos descartarlo, ya que la diferencia que había entre las métricas para el set de entrenamiento y para el set de testeo, variaron demasiado en comparación a los demás modelos. Por ejemplo, en R^2, para el set de entrenamiento nos dio 0.75 y para el set de testeo 0.66.
Volviendo al cuadro de resultados, en cuanto a RMSE, los mejores modelos de los requeridos son el Modelo XGB 1 y el Modelo RF 1, y por muy poco el mejor de los dos es el Modelo RF.
Sin embargo, el mejor modelo de todos es el Stacking ya que tiene las mejores métricas de todos los modelos. Pudo influir que se utilizaron los otros dos mejores modelos como modelos bases, pero habiendo probado con otros modelos como modelos bases, obtuvimos valores cercanos a esos, haciendo parecer que este modelo de por sí ya es muy bueno.
Como conclusión, si no se quiere implementar más de un modelo, los mejores para predecir el precio son el Modelo XGB 1 o el Modelo RF 1. Si se tiene la capacidad para implementar más de un modelo, el mejor es el Ensamble Stacking.

EJ 4 - Clustering

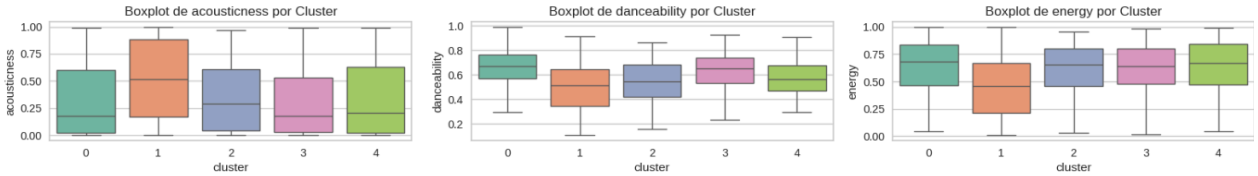
El df estaba totalmente limpio. Creamos nueva variable de minutos en base a la de milisegundos.
La cantidad de grupos que se deben de formar la tomamos del gráfico del codo, donde la cantidad de información que aporta cada nueva separación se vuelve muy poca.



Se estudió K-Means con k=3,4, y 5
El gráfico con el más óptimo k=3:



Vemos las medias de cada columna para poder ver cómo está agrupando. Y dependiendo de esos valores podemos categorizarlos en distintos géneros musicales o estados de ánimo.
Gráfico de 3 distintas variables en K=5



Haciendo la observación de esos promedios se pueden categorizar los distintos grupos.

Tiempo Dedicado

Integrante	Tarea	Prom. Hs Semana
Kevin Vallejo	Ejercicio 1: Collab (En todo) y parte del reporte. Ejercicio 3: Collab. Parte de limpieza. Modelos: Lineal simple, múltiple, KNN, Ensambles. Ejercicio 4: Collab (En todo) y reporte.	
Josefina Marianelli Keichian	**Ejercicio 1**: Collab. **Ejercicio 2**: Collab: Introducción, Limpieza, Información/resumen de columnas del dataset, parte del preprocesamiento del dataset post-limpieza, Random Forest + Ensambles Stacking + Conclusión + Cuadro comparativo de modelos + Parte del reporte (modelos random forest y ensambles stacking) **Ejercicio 4**: Collab.	4-6
Lucas Oshiro	**Ejercicio 1**: Collab. Parte del reporte **Ejercicio 3**: Collab: Introducción, limpieza y análisis, modelos XGBoost y Random Forest (k-fold cross validation) Reporte **Ejercicio 4**: Collab.	4-5
Brenda Aylas	Ejercicio 1: collab - preprocesamiento de datos Ejercicio 2: Collab - preprocesamiento y transformación de datos. Árbol de decisión y ensambles stacking. Ejercicio 4: Collab - detalle de columnas y preprocesamiento de datos. KMeans base, gráficos en general. Armado de reporte.	3-4