# TP1

## Análisis Exploratorio y Preprocesamiento de Datos

### Exploración inicial

* Dividimos el dataset original en train-test (80-20)
* Estudiamos el tipo de cada variable del dataset
* Preparamos los datasets (train y test)
  + Nos quedamos con
    - los tipos ‘Casa’, ‘Departamento’, ‘PH’
    - operación ‘Venta’
    - precio ‘USD’
    - ubicación ‘CABA’
  + Convertimos los campos de fechas
* Realizamos una descripción de los campos
* Calculamos medidas de resumen (media, mediana, q1, q3 y moda) para variables cuantitativas (‘property\_rooms’ y ‘property\_bedrooms’)
* Calculamos cantidades y frecuencias para variables cualitativas (‘place\_l3’ → ciudad, ‘place\_l4’ → barrio, ‘property\_type’)
* Determinamos **variables irrelevantes**
  + id → no se usa
  + operation → sólo trabajamos con ‘Venta’
  + property\_currency → sólo trabajamos con ‘USD’
  + place\_l5 → la columna estaba vacía
  + place\_l6 → la columna estaba vacía
  + property\_title → no se usa
* Visualizamos los gráficos de distribución de cada variable
* Visualizamos la **correlación** entre todas las variables con un pairplot

### Datos faltantes

#### A Nivel Columna → por cada variable

* Graficamos por cada variable el porcentaje de datos faltantes (columna vs. registro vacío)

#### A Nivel Fila → por cada registro

* Graficamos el porcentaje de filas con n cantidad de columnas con datos faltantes (fila vs. cantidad columnas vacías)

#### **Reparación** de Datos Faltantes

* + - * place\_l3 → llenamos con valores de place\_l2 bajo el método de **Hot Deck** (deberíamos haber realizado **Cold Deck** y rellenar los valores según datos externos a partir de las columnas de latitud y longitud)
      * place\_l4 → llenamos con valores de place\_l3 (ídem place\_l3)
      * place\_l5 y place\_l6 → cambiamos nulos por 0
      * latitud y longitud → imputamos a partir de la **media** agrupando por place\_l4
      * property\_surface\_total y property\_surface\_covered → los reemplazamos entre sí bajo el método de **Hot Deck**
      * property\_rooms → reemplazamos por la media según la cantidad de dormitorios
      * property\_bedrooms → reemplazamos por la media según la cantidad de ambientes
      * property\_rooms y property\_bedrooms → cuando faltan ambas, predecimos mediante **regresión lineal** property\_rooms a partir de property\_surface\_covered y luego utilizando la media a property\_bedrooms

### Valores atípicos

#### Análisis Univariado

* + - * Boxplot → estudiamos posibles outliers por encima de los límites superiores e inferiores de los cuartiles y borramos en caso de identificar valores atípicos
      * Z Score → estudiamos los posibles valores atípicos que superan la regla de oro (|Z| **> 3**) y borramos aquellos que representaban un porcentaje menor al 1%
      * Z Score Modificado → estudiamos los posibles valores atípicos que superan la regla de oro (|Z| **> 3.5**) y borramos aquellos que representaban un porcentaje menor al 1%

#### Análisis Multivariado

* + - * **Boxplot** según property\_type → estudiamos posibles outliers por encima de los límites superiores e inferiores de los cuartiles y borramos en caso de identificar valores atípicos
      * Distancia de **Mahalanobis** según property\_type → estudiamos los valores de property\_rooms y property\_bedrooms contra property\_surface\_total y property\_surface\_covered y analizamos el dispersograma resultante para observar qué valores categorizar como outliers

### Reducción de la dimensionalidad

* Para este punto ya tenemos eliminados los considerados **outliers**
* Eliminamos **variables** previamente determinadas como **irrelevantes**
* Encontramos 3 pares de **variables** altamente **correlacionadas** y eliminamos una
* Estudiamos variables con **varianza baja** pero no eliminamos ninguna

## Agrupamiento

* Transformamos las variables cualitativas con One Hot Encoding
* Estudiamos si existe **tendencia al clustering** con la estadística de **Hopkins** (para valores cercanos a 0 significa que hay tendencia)
* Utilizamos **K-Means** junto al método del **codo** para definir qué cantidad de clusters utilizar
* Utilizamos el método de **Silhouette** para determinar qué tan óptima es la cantidad de clusters que elegimos a partir del método del codo}
* Analizamos la clasificación obtenida según cada variable y obtenemos que las más importantes para la clusterización son: ubicación, precio y superficie.

## Clasificación

### Construcción del target

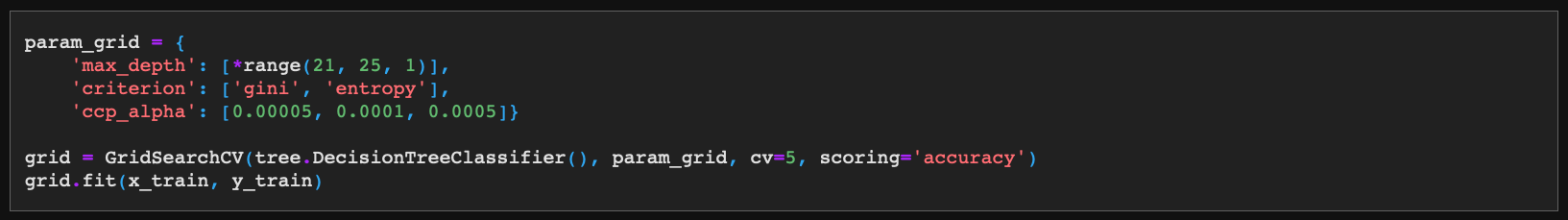
* Construimos la variable **tipo\_precio** evaluando la relación entre precio y metro cuadrado de tres categorías: alto, medio y bajo.
* **Dividimos** la variable pxm2 en 3 intervalos utilizando 3 alternativas distintas:
  + tipo\_precio\_1 → ordenamos los valores y partimos el datset en 3 partes iguales
  + tipo\_precio\_2 → evaluamos según cuartiles y asignamos q1, q2, q3 a bajo, medio, alto respectivamente
  + tipo\_precio\_3 → como la técnica anterior pero agrupando por tipo de propiedad
* Estudiamos cómo se comportan las distribuciones de las variables tipo\_precio\_1, tipo\_precio\_2 y tipo\_precio\_3 y vimos que la **mediana** del precio es **distinta** según el **tipo de propiedad**
* Concluimos que la técnica más efectiva es la que utilizamos para tipo\_precio\_3
* Agrupamos por KMeans con 3 clusters y por el target y vemos resultados similares
  + Los PH se distribuyen casi iguales a lo largo de las 3 categorías
  + El grupo de precios altos abarca casi todo el mapa
  + Es mucho menor la cantidad de registros de precios bajos

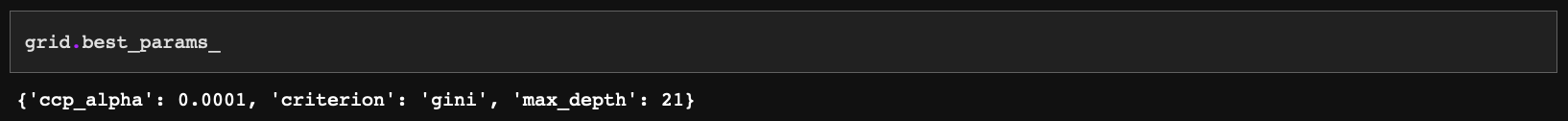
### Entrenamiento y predicción

* **Eliminamos** de train y test las **variables** de tipo\_precio y cualquier otra que contenga **información del precio** de venta
* Dejamos como target tipo\_precio\_3
* Realizamos One Hot Encoding para las variables cualitativas
* Buscamos los mejores **hiperparámetros** con GridSearchCV en base al Accuracy porque es la métrica más intuitiva y queremos maximizar el porcentaje de observaciones

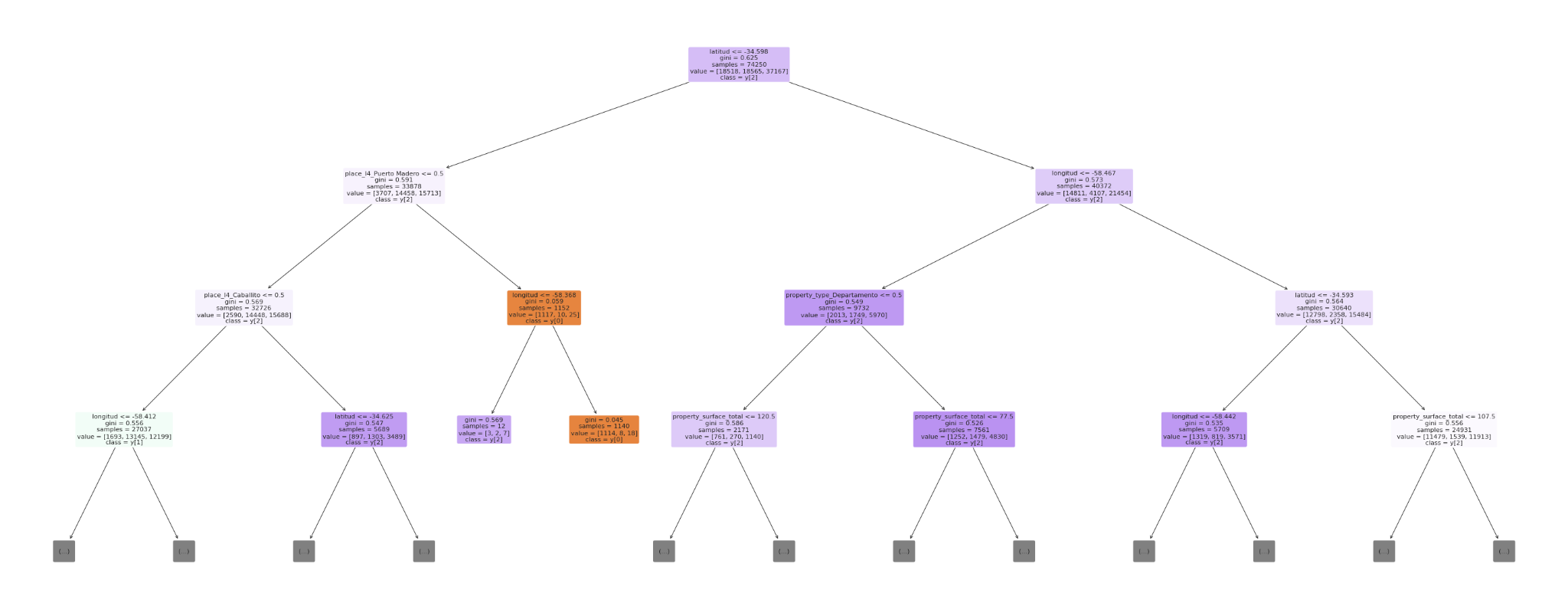
#### Modelo 1 → Árbol de Decisión

* Buscamos los mejores hiperparámetros:



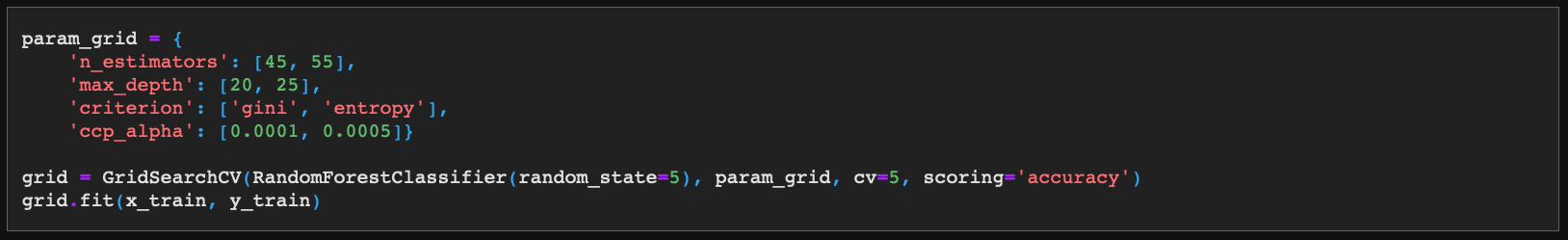


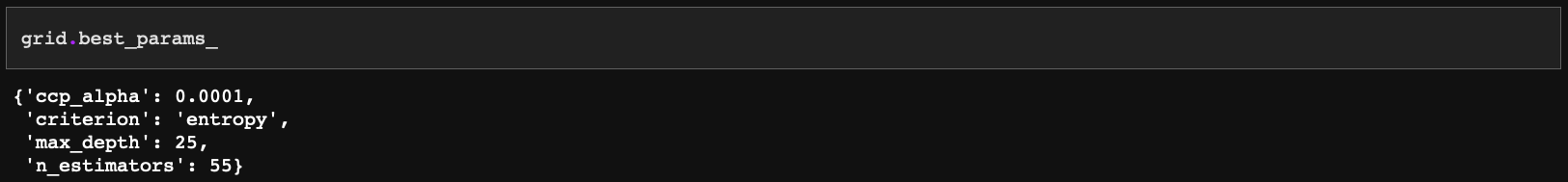
* Predecimos el set de entrenamiento y obtenemos:
  + Accuracy: 0.66
  + Precision: 0.68
  + Recall: 0.61
  + F1 Score: 0.63
* Predecimos el set de test y obtenemos:
  + Accuracy: 0.64
  + Precision: 0.66
  + Recall: 0.59
  + F1 Score: 0.61
* Vemos que las métricas dan muy similares
* Exportamos el árbol:



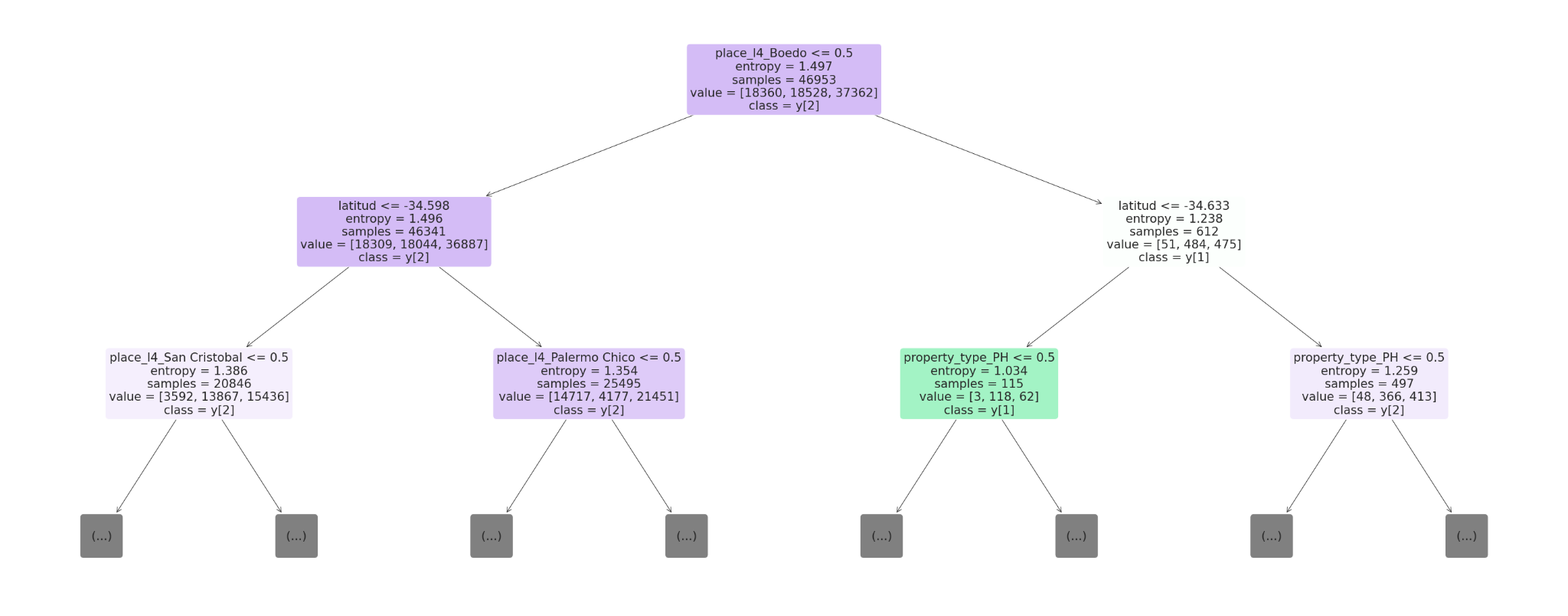
#### Modelo 2 → Random Forest

* Buscamos los mejores hiperparámetros:



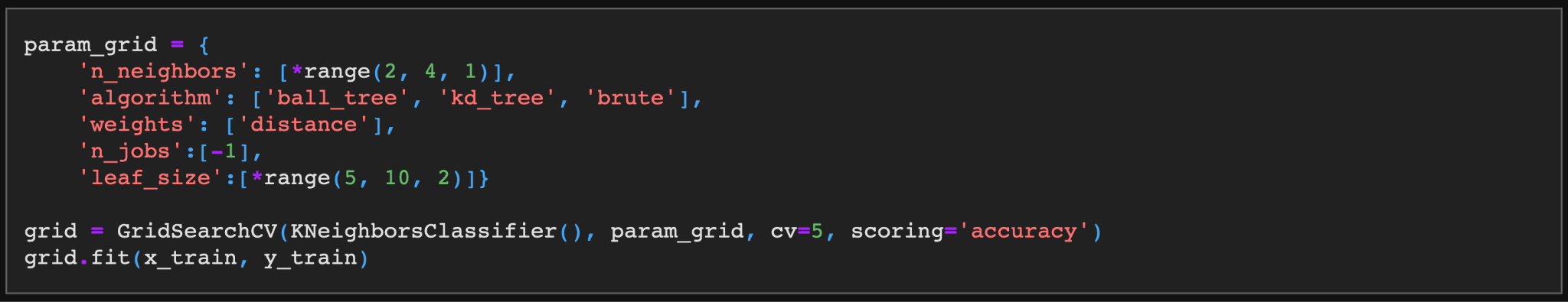


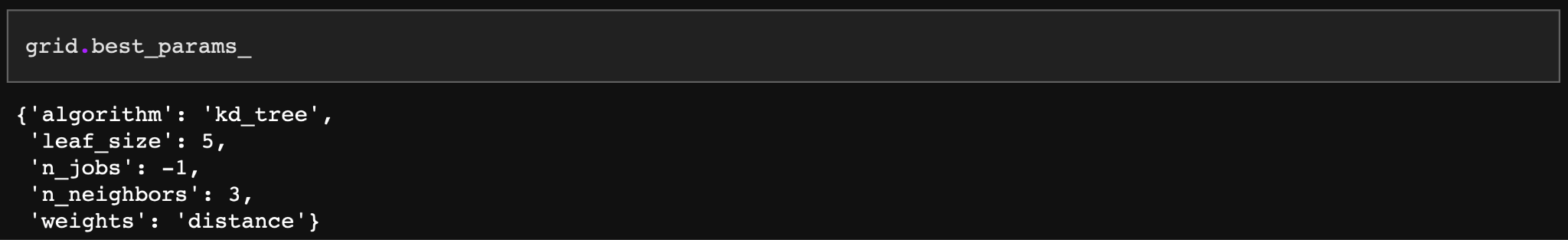
* Predecimos el set de entrenamiento y obtenemos:
  + Accuracy: 0.75
  + Precision: 0.79
  + Recall: 0.71
  + F1 Score: 0.73
* Predecimos el set de test y obtenemos:
  + Accuracy: 0.68
  + Precision: 0.71
  + Recall: 0.63
  + F1 Score: 0.65
* Vemos que las métricas dan muy similares
* Exportamos el árbol:



#### Modelo 3 → K Nearest Neighbours

* Buscamos los mejores hiperparámetros:





* Predecimos el set de entrenamiento y obtenemos:
  + Accuracy: 1.0
  + Precision: 1.0
  + Recall: 1.0
  + F1 Score: 1.0
* Predecimos el set de test y obtenemos:
  + Accuracy: 0.48
  + Precision: 0.44
  + Recall: 0.44
  + F1 Score: 0.44

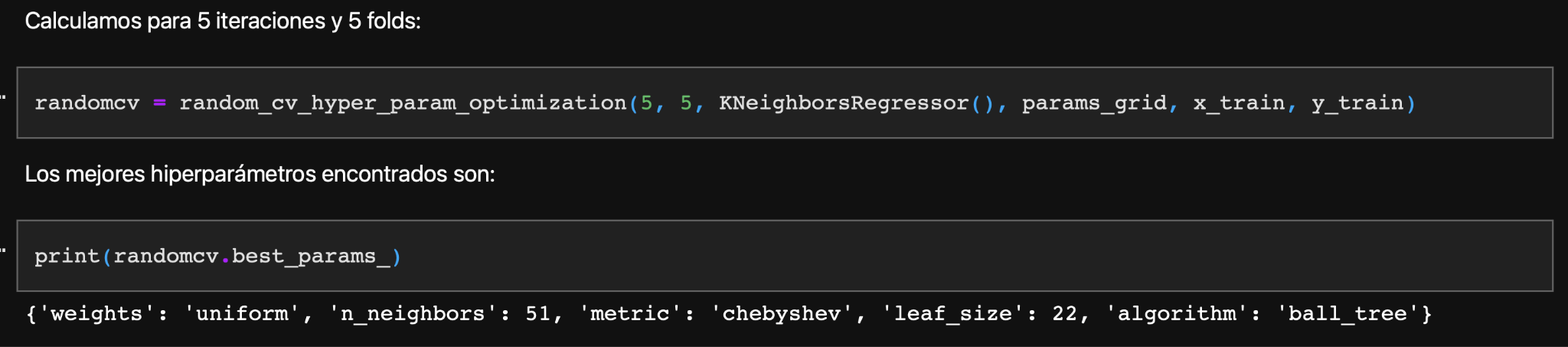
## Regresión

### Modelo 1 → KNN

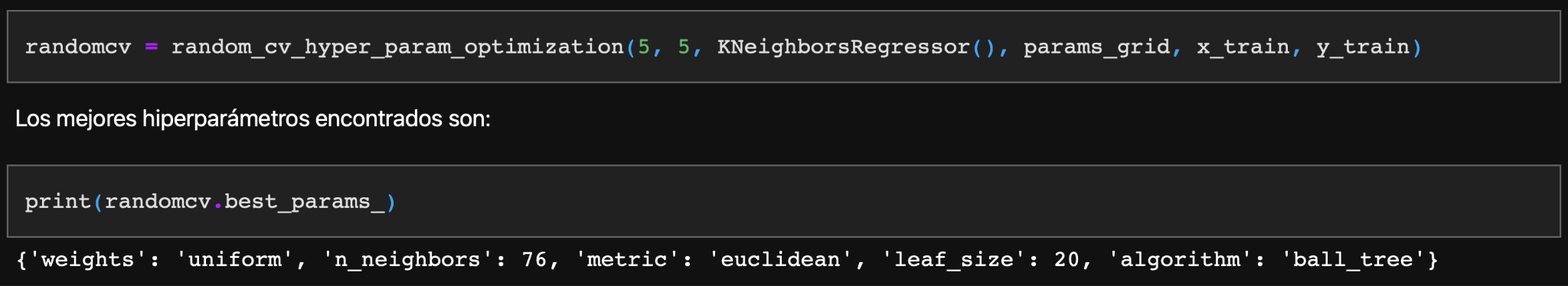
* Entrenamos el dataset de train:
  + Accuracy: 0.193%
  + MSE: 72 billones
  + RMSE: 269 mil
  + Score: 0.424
* Entrenamos el dataset reducido:
  + Accuracy: 0.15%
  + MSE: 82 billones
  + RMSE: 286 mil
  + Score: 0.349
* Observamos que los modelos no son buenos predictores
* Optimizamos hiperparámetros usando RandomSearchCV con los parámetros:



* Buscamos para el dataset de train:



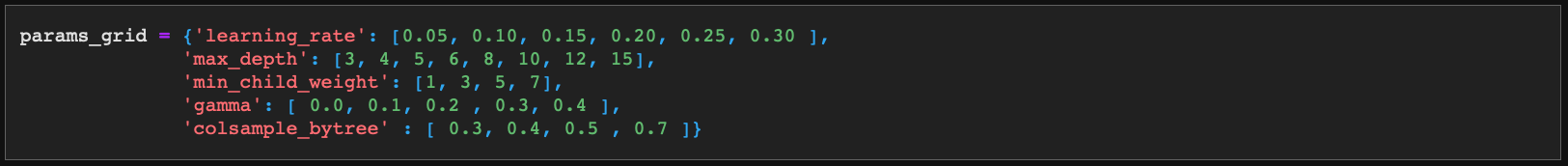
* Entrenamos y obtenemos:
  + Accuracy: 0.0%
  + MSE: 70 billones
  + RMSE: 265 mil
  + Score: 0.441
* Buscamos para el dataset reducido:



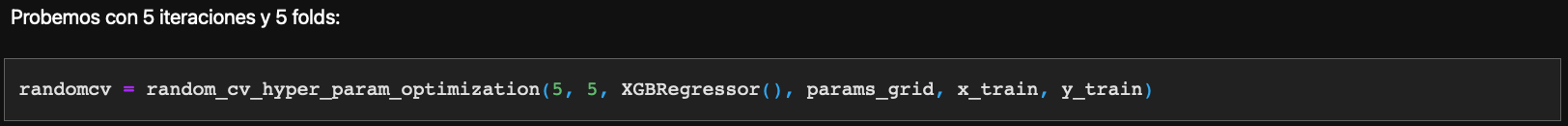
* Entrenamos y obtenemos:
  + Accuracy: 0.0%
  + MSE: 78 billones
  + RMSE: 280 mil
  + Score: 0.376
* Vemos que no se obtienen buenos puntajes de predicción a pesar de que mejoraron luego de la optimización de hiperparámetros

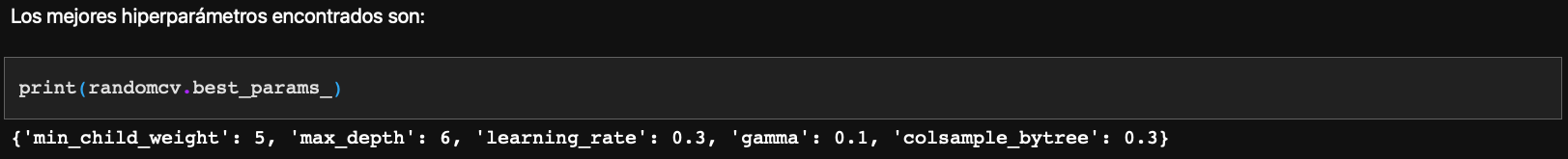
### Modelo 2 → XG Boost

* Entrenamos el dataset de train:
  + Accuracy: 0.0%
  + MSE: 53 billones
  + RMSE: 228 mil
  + Score: 0.424
* Entrenamos el dataset reducido:
  + Accuracy: 0.0%
  + MSE: 56 billones
  + RMSE: 237 mil
  + Score: 0.555
* Observamos que los modelos no son buenos predictores
* Optimizamos hiperparámetros usando RandomSearchCV con los parámetros:

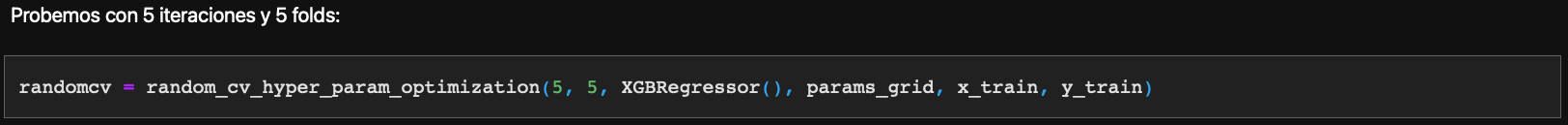


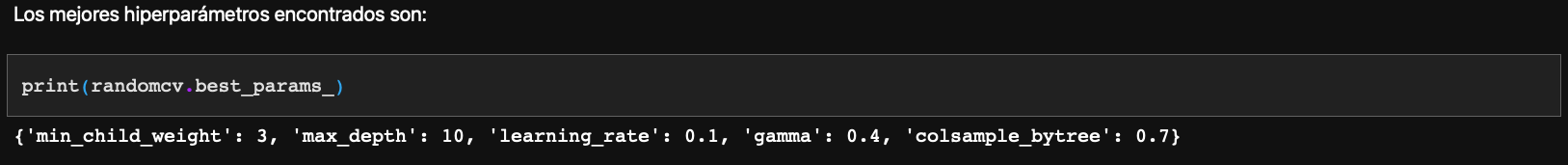
* Buscamos para el dataset de train:





* Entrenamos y obtenemos:
  + Accuracy: 0.0%
  + MSE: 52 billones
  + RMSE: 229 mil
  + Score: 0.584
* Buscamos para el dataset reducido:

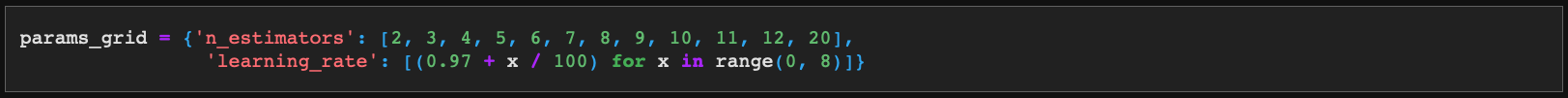




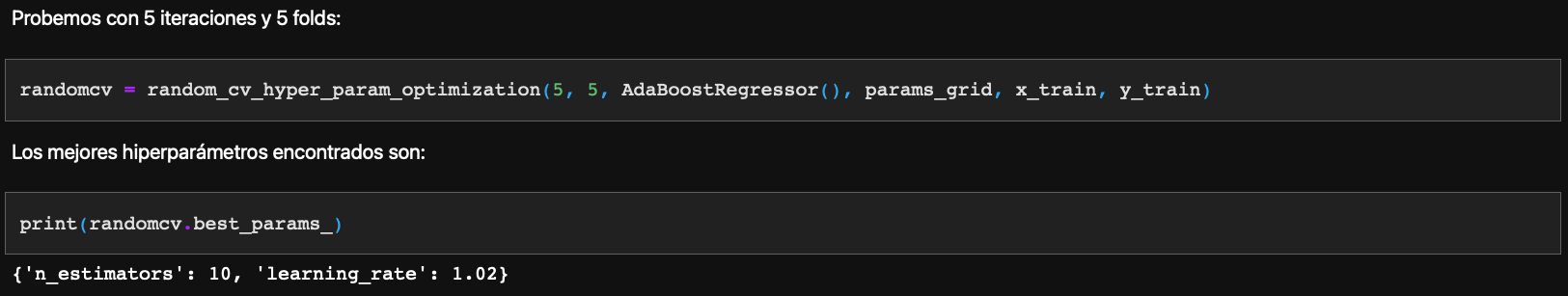
* Entrenamos y obtenemos:
  + Accuracy: 0.0%
  + MSE: 51 billones
  + RMSE: 227 mil
  + Score: 0.591
* Observamos que el Score del modelo alcanza aproximadamente un 60% y resulta mejor la regresión con el dataset reducido que con el de train

### Modelo 3 → AdaBoost

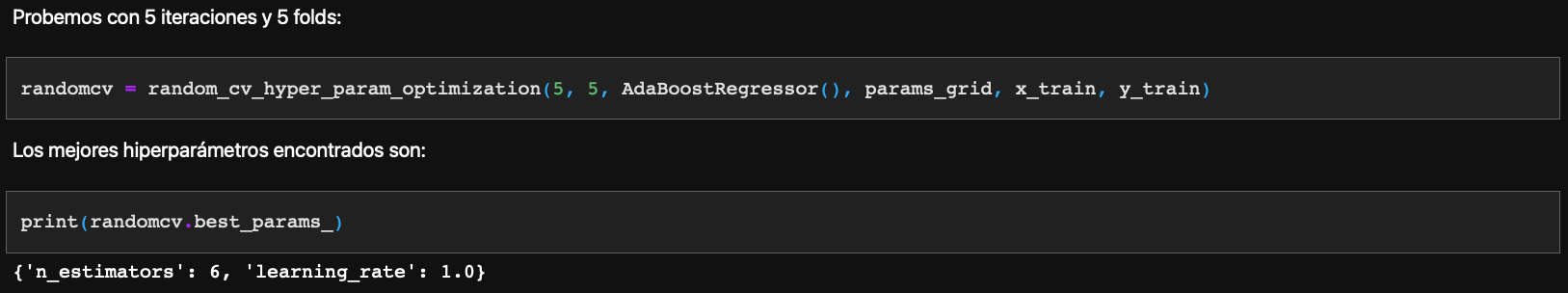
* Entrenamos el dataset de train:
  + Accuracy: 0.0%
  + MSE: 102 billones
  + RMSE: 319 mil
  + Score: 0.191
* Entrenamos el dataset reducido:
  + Accuracy: 0.0%
  + MSE: 785 billones
  + RMSE: 280 mil
  + Score: 0.378
* Observamos que los modelos no son buenos predictores
* Optimizamos hiperparámetros usando RandomSearchCV con los parámetros:



* Buscamos para el dataset de train:



* Entrenamos y obtenemos:
  + Accuracy: 0.0%
  + MSE: 65 billones
  + RMSE: 256 mil
  + Score: 0.481
* Buscamos para el dataset reducido:



* Entrenamos y obtenemos:
  + Accuracy: 0.0%
  + MSE: 73 billones
  + RMSE: 271 mil
  + Score: 0.417
* Observamos que las estimaciones mejoran pero no alcanzan el Score de los dos previos → es el menos óptimo

# 

# TP2

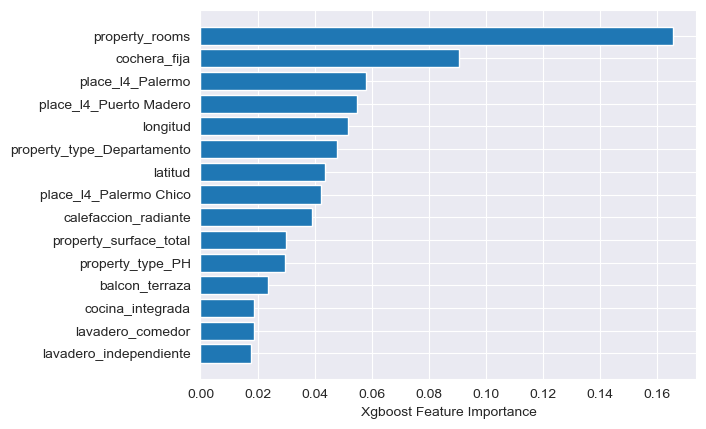
## Procesamiento del Lenguaje Natural

### Ampliación del Dataset

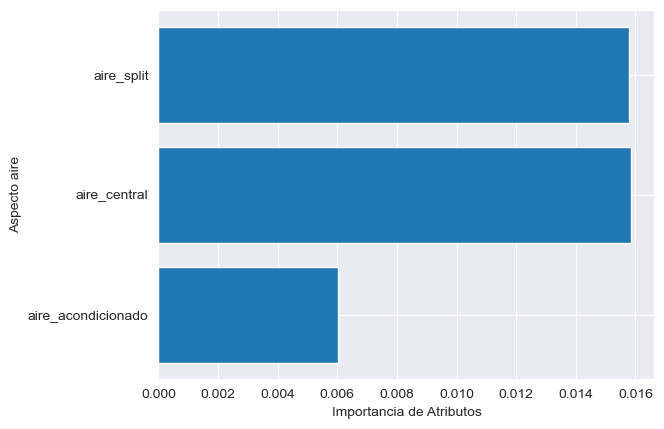
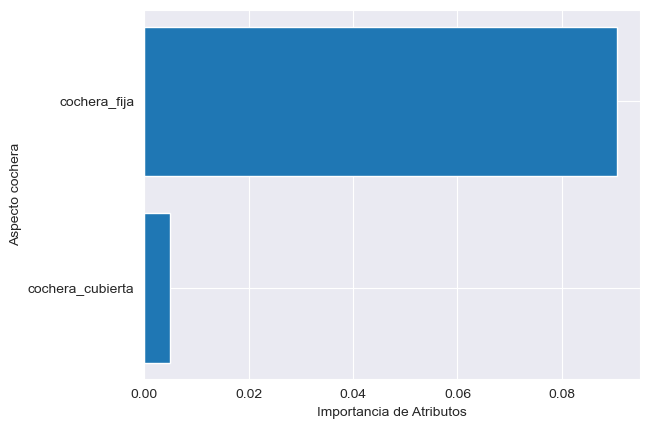
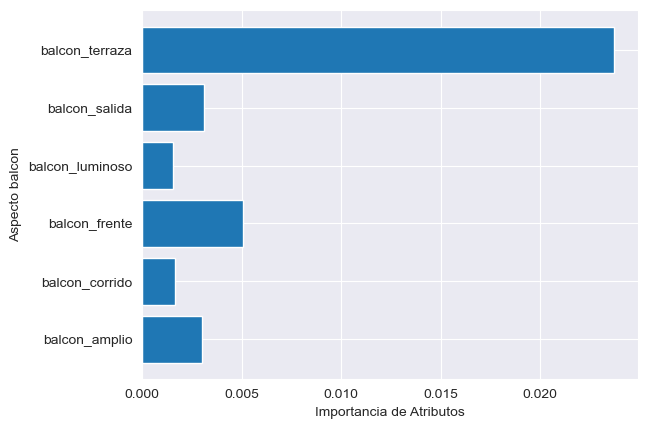
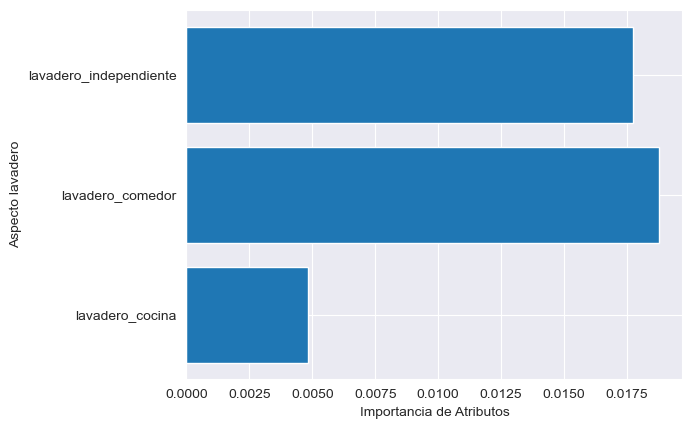
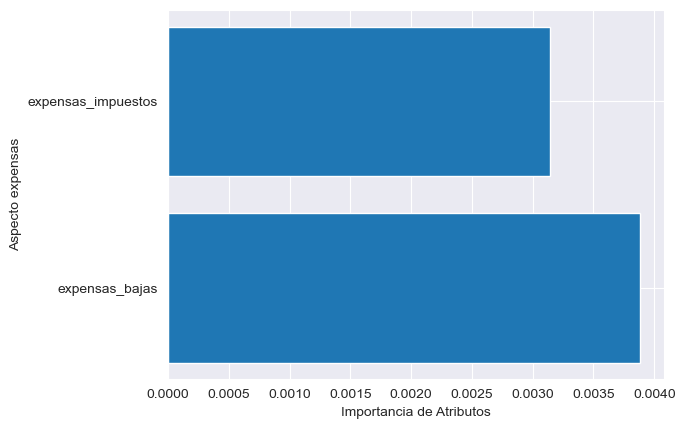
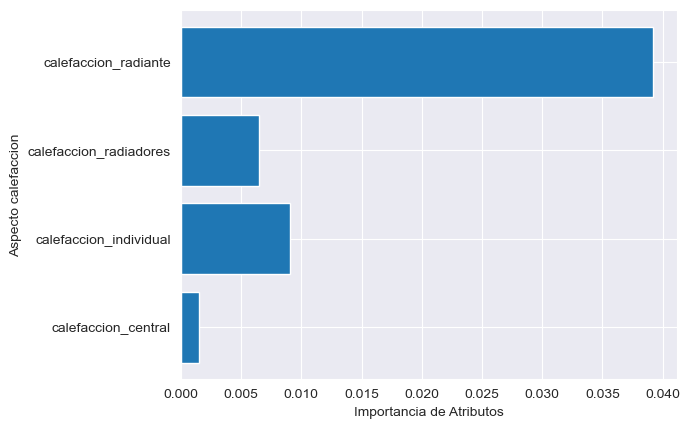
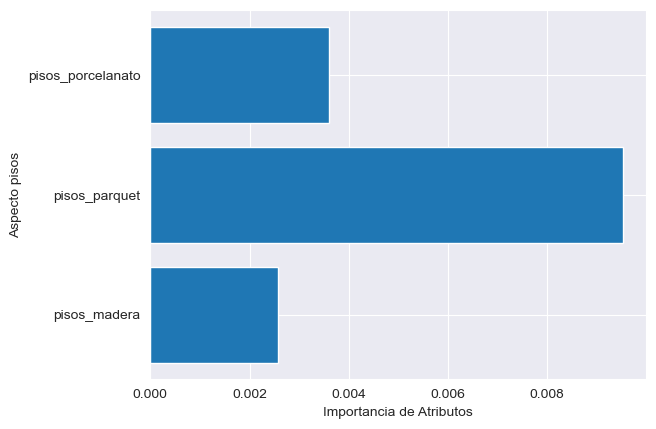
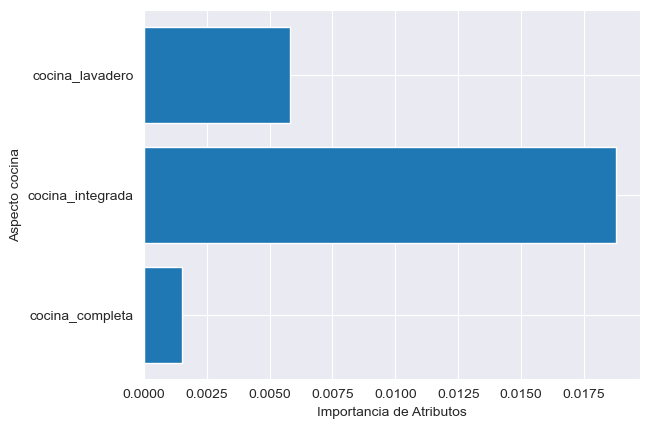
* Ampliamos el dataset utilizado para entrenamiento en el TP1 y adicionamos la columna **property\_description**
* Estudiamos cuáles son las 100 palabras más **frecuentes** (Minqing Hu y Bing Liu) en la columna property\_description y vemos que hay etiquetas html, letras en mayúsculas y minúsculas, con tilde, números, etc.
* Filtrando con **ReGex**, utilizamos técnicas de reducción de la dimensionalidad:
  + eliminamos etiquetas html
  + transformamos las palabras a minúsculas
  + quitamos los tildes
  + eliminamos símbolos
  + eliminamos espacios múltiples
  + eliminamos palabras sin significado (stop words)
* Nuevamente estudiamos cuáles son las 100 palabras más comunes y **seleccionamos** las que consideramos relevantes: cocina, pisos, calefaccion, expensas, lavadero, balcon, cochera y aire
* Estudiamos cuáles son las palabras más comunes **para cada uno de estos aspectos** y seleccionamos los que consideramos como relevantes
* **Agregamos al dataset** de train cada uno de los aspectos como columnas y como valores el que se encontró para el aspecto en cuestión

### Modelos

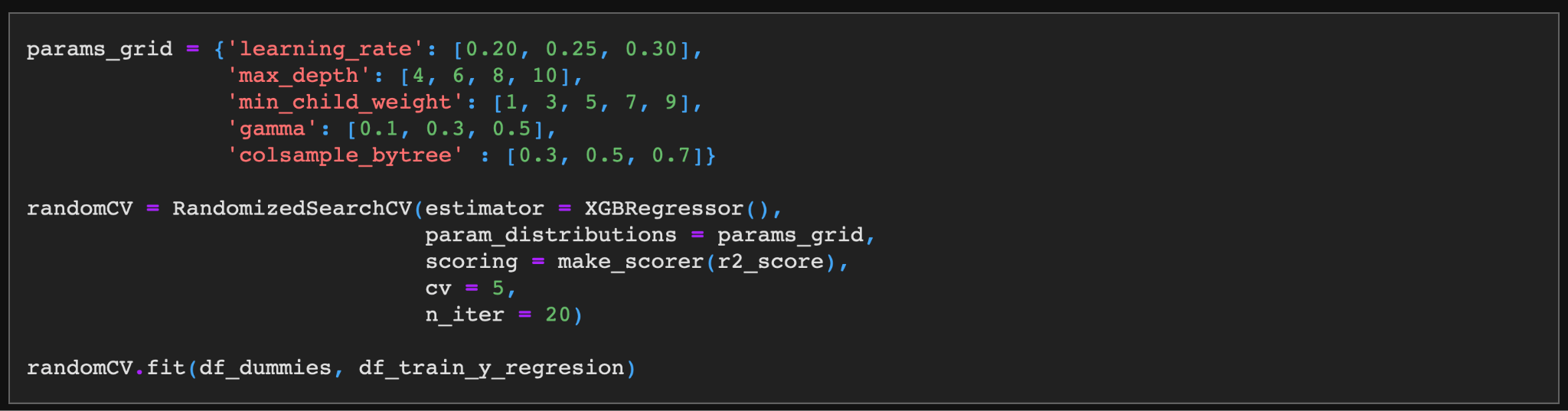
* Entrenamos un modelo **XGBoost** para regresión con el nuevo dataset ampliado utilizando los mismos hiperparámetros que TP1:
* min\_child\_weight = 5
* max\_depth = 6
* learning\_rate = 0.3
* gamma = 0.1
* colsample\_bytree = 0.3
  + - MSE: 53 billones
    - RMSE: 230 mil
    - Score: 58.0%
* Estudiamos la importancia de features:



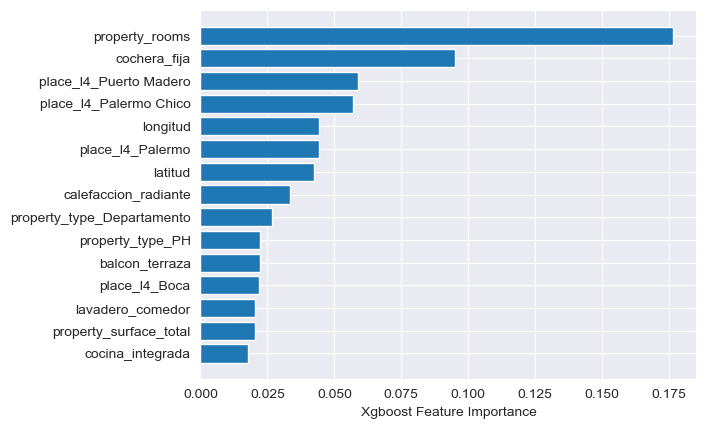
* Estudiamos la importancia de features por aspecto:



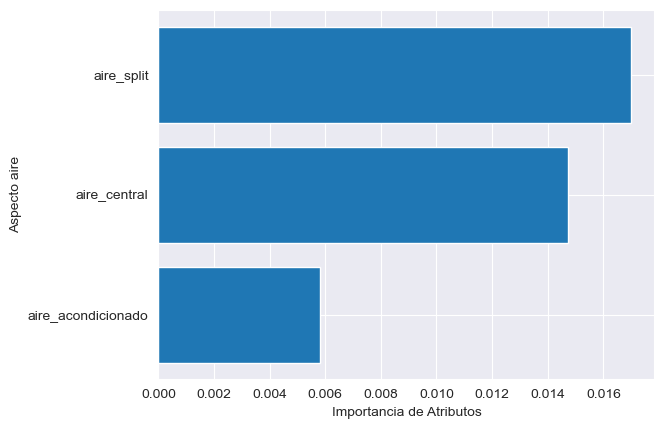
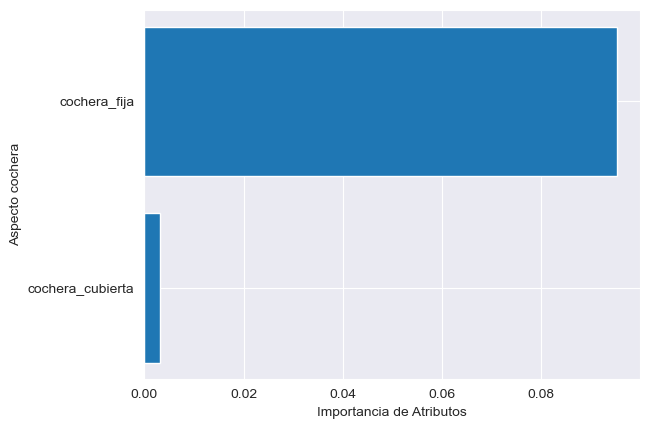
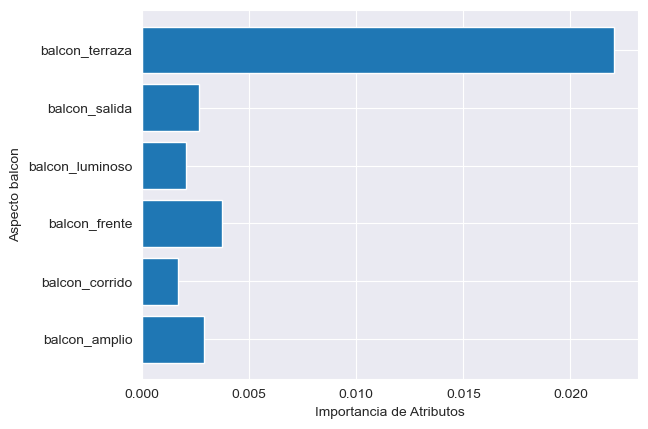
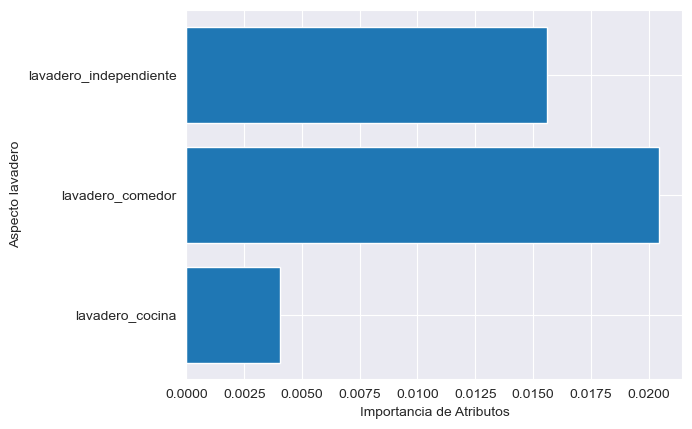
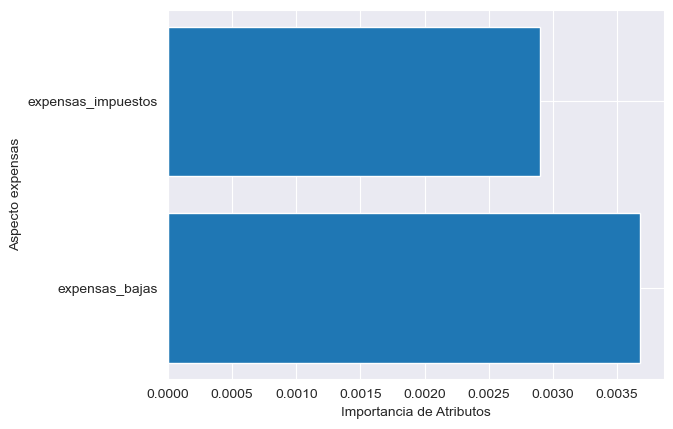
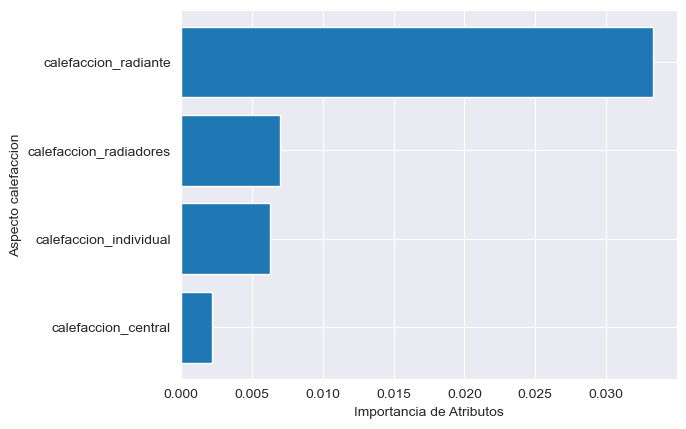
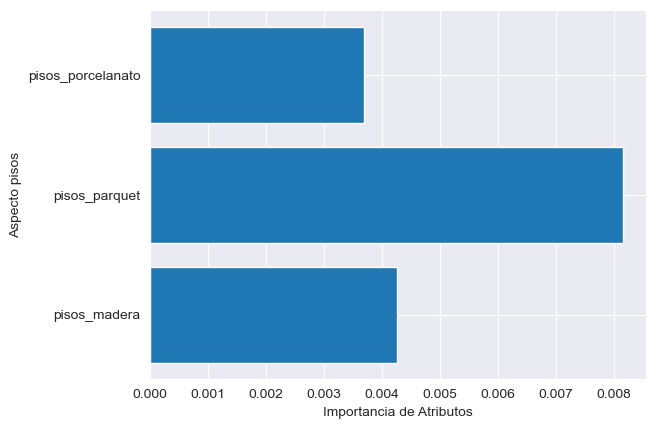
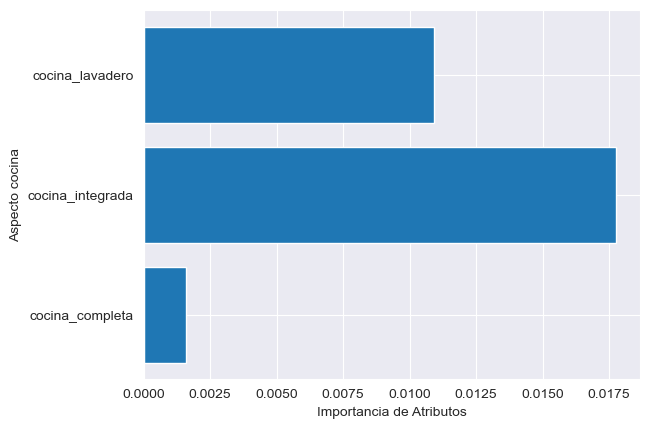
* Realizamos una optimización de hiperparámetros:



* min\_child\_weight: 1
* max\_depth: 8
* learning\_rate: 0.3
* gamma: 0.1
* colsample\_bytree: 0.3
  + MSE: 53 billones
  + RMSE: 230 mil
  + Score: 57.9%
* Estudiamos la importancia de features:



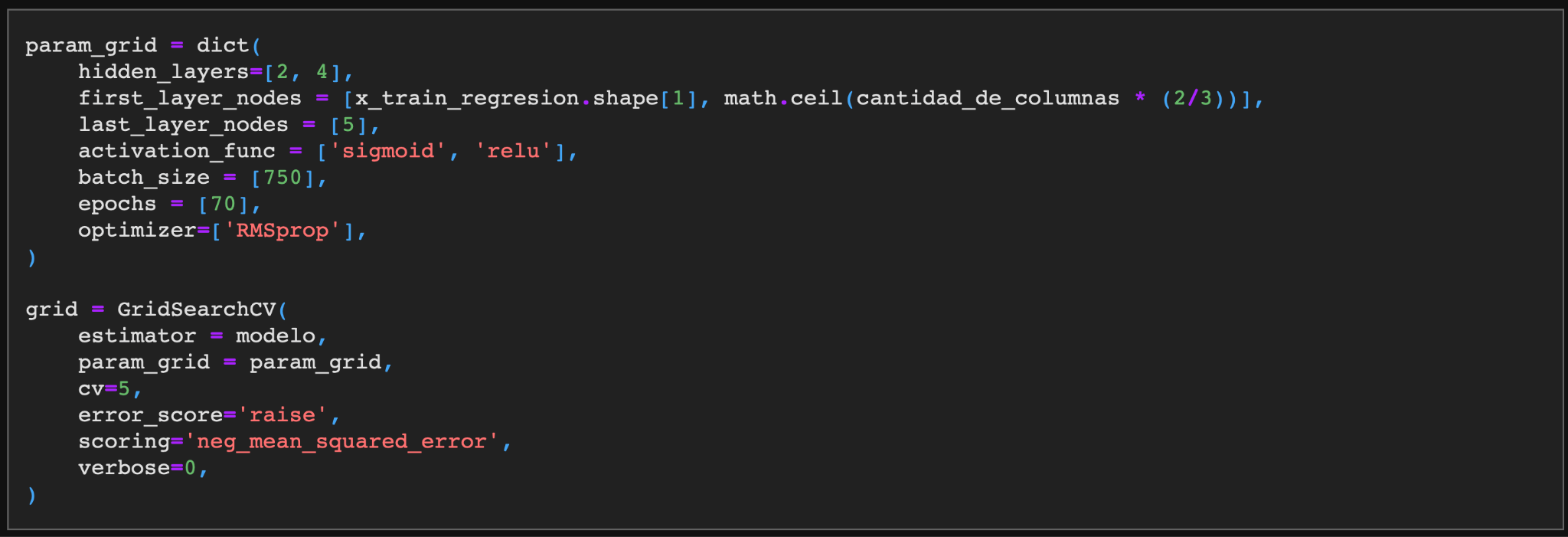
* Estudiamos la importancia de features por aspecto:



## Redes Neuronales

### Regresión

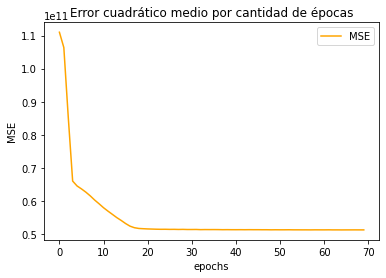
* Sacamos columnas de ids
* Normalizamos todos los valores con StandardScaler
* Buscamos el mejor modelo con GridSearchCV:



* activation\_func: relu
* batch\_size: 750
* epochs: 70
* first\_layer\_nodes: 69
* hidden\_layers: 4
* last\_layer\_nodes: 5
* optimizer: RMSprop
* Error absoluto promedio: 65.5%
* Error absoluto cuadrado promedio: 51 millones

#### Predecir precio de la propiedad

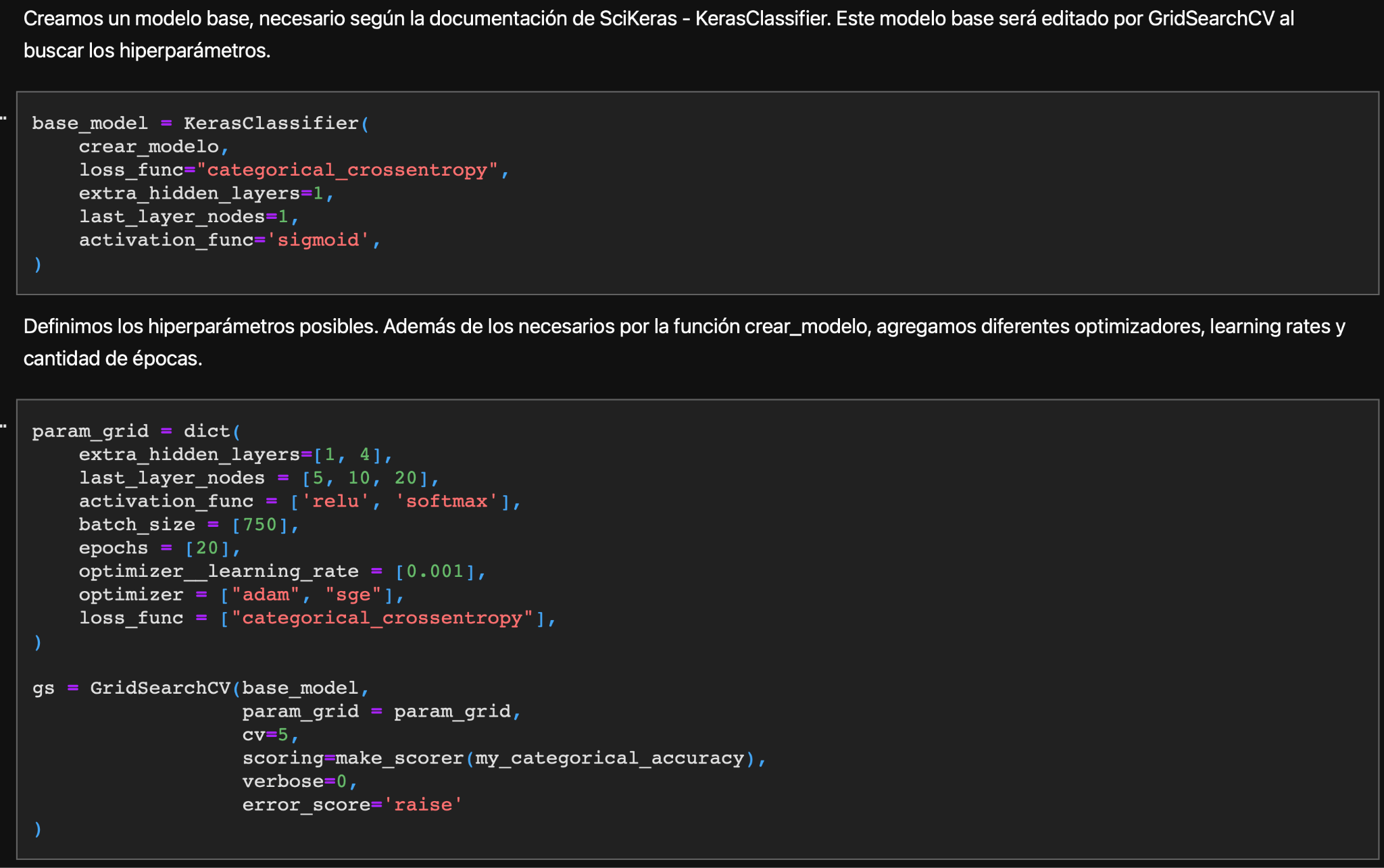
* Predecimos el dataset de train y obtenemos:



* + MSE: 107 millones
  + RMSE: 327 mil

### Clasificación

* Sacamos columnas de ids
* **Normalizamos** todos los valores con Z Score
* Aplicamos One Hot Encoding a la columna target para que tenga 3 y coincida con el output del modelo
* Buscamos el mejor modelo con **GridSearchCV**:



* Obtenemos un modelo secuencial de 4 capas densas

#### Predecir tipo precio

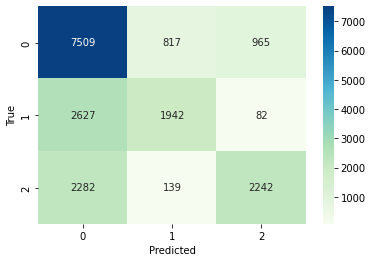
* Entrenamos el dataset de train y obtenemos:
  + Accuracy: 0.57
  + Precision: 0.58
  + Recall: 0.5
  + F1 Score: 0.51

## Ensamble de Modelos Híbridos

* Preparamos la variable target bajo la misma convención que para el TP1 → pxm2 agrupando por tipo de propiedad en 3 intervalos, 25% a bajo, 50% a medio y el otro 25% restante a alto

### Ensamble 1 (Clasificación) → Voting

* Usamos el dataset **reducido** del TP1
* Usamos el tipo de votación hard (**regla de la mayoría**)
* Definimos el ensamble con los mismos empleados en el TP1:
  + **Árbol de decisión**
    - ccp\_alpha: 0.0001
    - criterion: gini
    - max\_depth: 21
  + **Random forest**
    - ccp\_alpha: 0.0001
    - criterion: entropy
    - max\_depth: 25
    - n\_estimators: 55
  + **KNN**
    - algorithm: kd\_tree
    - leaf\_size: 5
    - n\_jobs: -1
    - n\_neighbors: 3
    - weights: distance
* Entrenamos, predecimos y obtenemos:
  + Accuracy: 0.63
  + Precision: 0.65
  + Recall: 0.57
  + F1 Score: 0.59



### Ensamble 2 (Regresión) → Stacking

* Usamos el dataset **reducido** del TP1
* Modelos base:
  + **KNN**
    - weights: uniform
    - n\_neighbors: 51
    - metric: chebyshev
    - leaf\_size: 22
    - algorithm: ball\_tree
  + **XGBoost**
    - min\_child\_weight: 5
    - max\_depth: 6
    - learning\_rate: 0.3
    - gamma: 0.1
    - colsample\_bytree: 0.3
  + **Adaboost**
    - n\_estimators: 10
    - learning\_rate: 1.02
* Meta-modelo: **Gradient Boosting**
* Entrenamos, predecimos y obtenemos:
  + MSE: 74 millones
  + RMSE: 272 mil
  + R2 Score: 0.410
* Vemos que el **MSE se redujo** a la mitad que en el TP1

# Conclusiones Métricas

* Precision: Cuántos de los que predije son verdaderos.
* Recall: A cuántos de los que son verdades le pegue.
* Accuracy: Que porcentaje de casos acerte (No es buena si la matriz está desbalanceada).
* **F1 Score**: Combina Precision y Recall.

