Trabajo práctico 1: predicción del precio de una vivienda en CABA.

Grupo 01 - Los Defensores de Pearson

Análisis Exploratorio

El dataset en crudo que se nos otorga cuenta con más de 450 mil registros y 20 columnas.
Las características más destacables del mismo son:

- **property_price:** El precio de publicación es el dato que buscamos que nuestro modelo pueda predecir por lo que es necesario entrenarlo conociendo este dato. Es de tipo numérico.
- **property_currency:** Sirve para realizar el filtrado inicial. Es de tipo string. Una vez filtrado por este dato, la columna es descartable ya que tendría el mismo valor en todas las filas (“USD”).
- **operation:** Sirve para realizar el filtrado inicial. Es de tipo string. Una vez filtrado por este dato, la columna es descartable ya que tendría el mismo valor en todas las filas (“Venta”).
- **property_type:** Sirve para realizar el filtrado inicial. Determina qué es lo que está siendo publicado (Casa, local comercial, etc). Es de tipo string. Una vez filtrado nos quedan 3 tipos (Casa, PH, Departamento) los cuales pueden llegar a tener correlación con el precio.
- **Property_rooms** y **property_bedrooms:** Indican la cantidad de habitaciones y ambientes (respectivamente). Son de tipo numérico. Puede existir una relación entre estas cantidades y el precio de la publicación.
- **property_surface_total** y **property_surface_covered:** Indican los metros cuadrados del total de la propiedad y los cubiertos (respectivamente). Son de tipo numérico. Puede existir una relación entre estas cantidades y el precio de la publicación.

- **place_l2, place_l3, place_l4...:** Indican la ubicación de la propiedad. Sea provincia, localidad, barrio. Son de tipo string. Estos datos son de alto interés ya que los precios de las propiedades suelen estar altamente influenciados por la ubicación. Existe una grán proporción de datos nulos cuanto más específica es la ubicación.
- **property_title**: Indica el título con el cual se realizó la publicación. Este puede llegar a aportar información útil al momento de tomar decisiones sobre las otras columnas, ya que generalmente en cada registro aporta información sobre la cantidad de ambientes y/o locación. Es de tipo string.

Analizamos las columnas para filtrar según los criterios indicados buscando irregularidades que causen un filtrado menos preciso. Por ejemplo, que la columna **property_currency** (la cual tenemos que filtrar por el valor “USD”) no tenga algunos registros en mayúscula y otros en minúsculas, o no diga “Dólares” en lugar de “USD‘ y por ello no sea captada por el filtro. Ninguna de las columnas que son criterio de filtrado presentó problemas de este tipo.

Al filtrar obtenemos un dataframe de más de 94 mil registros. Este se divide en conjunto de entrenamiento y prueba en una proporción 80/20, teniendo como resultado 75 mil registros de entrenamiento y casi 19 mil de prueba.

Todo el análisis se realiza sobre el conjunto de entrenamiento.

Hipótesis:

- Las casas tienden a tener 1 ambiente más que el total de habitaciones (el living).
- Las coordenadas (latitud y longitud) son mucho más propensas a presentar un grave error de precisión a comparación del barrio.
- La superficie cubierta es menor o igual a la superficie total.
- Las propiedades con un exagerado número de habitaciones (mansiones por ejemplo) son un mercado paralelo y no rigen los mismos criterios que para la propiedad promedio.

Preprocesamiento de Datos

Tras filtrar el dataset según los criterios indicados, nuestro dataset presenta ciertas características:

1. **Columnas eliminadas:**
    a. place_l4: +96% nulls
    b. place_l5: 100% nulls
    c. place_l6: 100% nulls
    d. start_date: No aporta información
    e. end_date: No aporta información
    f. created_on: No aporta información
    g. property_currency: Todos los registros tienen el mismo valor
    h. operation: Todos los registros tienen el mismo valor

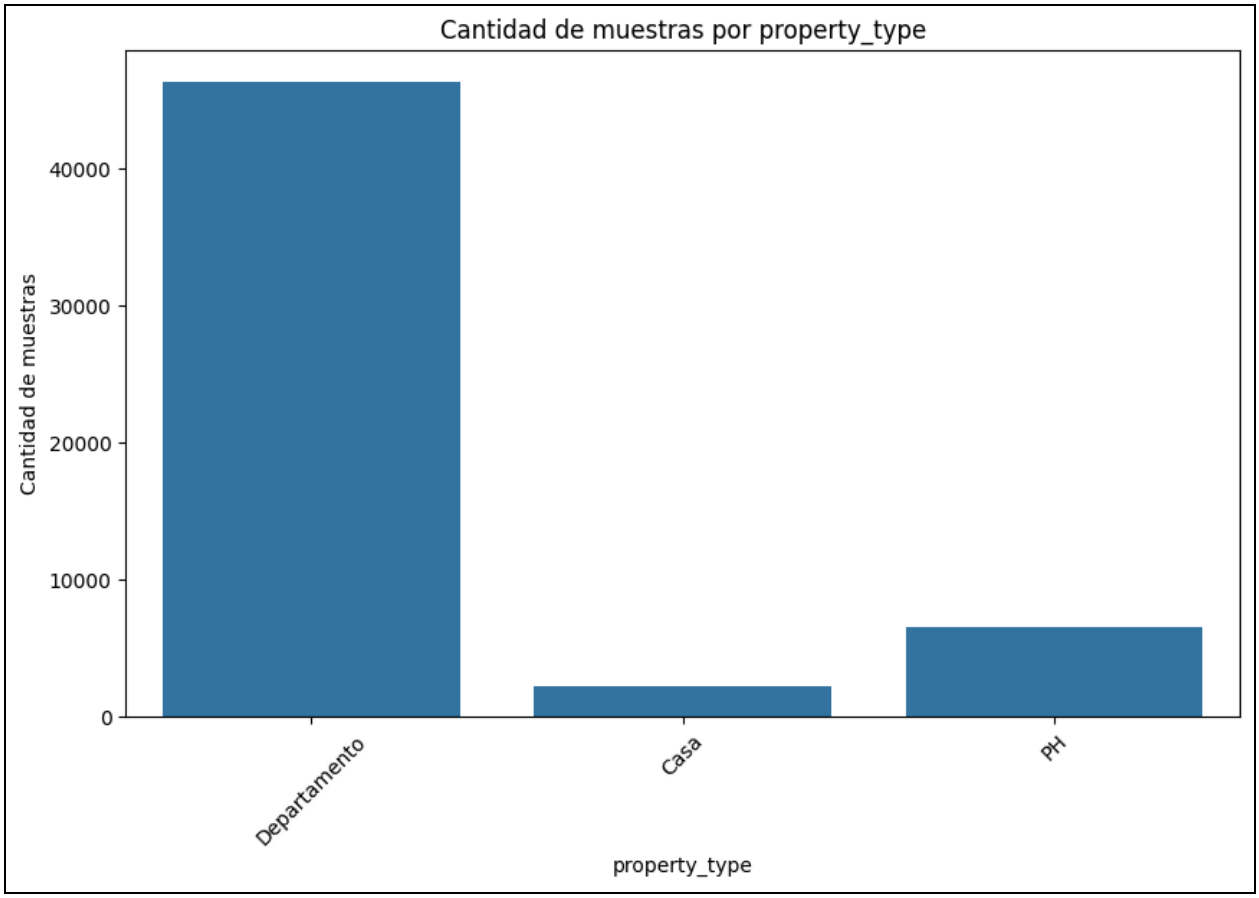
2. **Correlaciones:**
    a. Cantidad de habitaciones y cantidad de ambientes: 0.87. Esto confirma nuestra hipótesis sobre la tendencia de las propiedades de tener 1 ambiente más que la cantidad de habitaciones.
    b. Superficie cubierta y superficie total por tipo de propiedad:
    En PH's y casas la correlación es mayor a 0.9. En departamentos ronda el 0.55.
    En casi todos los barrios los departamentos tienen correlación +0.8, lo cual tiene mucho sentido ya que generalmente un departamento es en su mayoría superficie cubierta.
    Este dato nos es útil para imputar las filas donde uno de los datos es nulo y el otro no.

3. **Nuevas features:**
    a. total_m2_price: Precio por metro cuadrado de superficie total.
    b. covered_m2_price: Precio por metro cuadrado de superficie cubierta.
    c. Casa, departament y PH: Aplicando One Hot Encoding reemplazamos la columna "property_type" por estas 3 nuevas columnas dummies
    d. Palermo, Belgrano, Caballito, Otros: Aplicando One Hot Encoding reemplazar la columna "place_l3" por dummies de las ubicaciones con mayor presencia. Útil para el análisis de grupos.

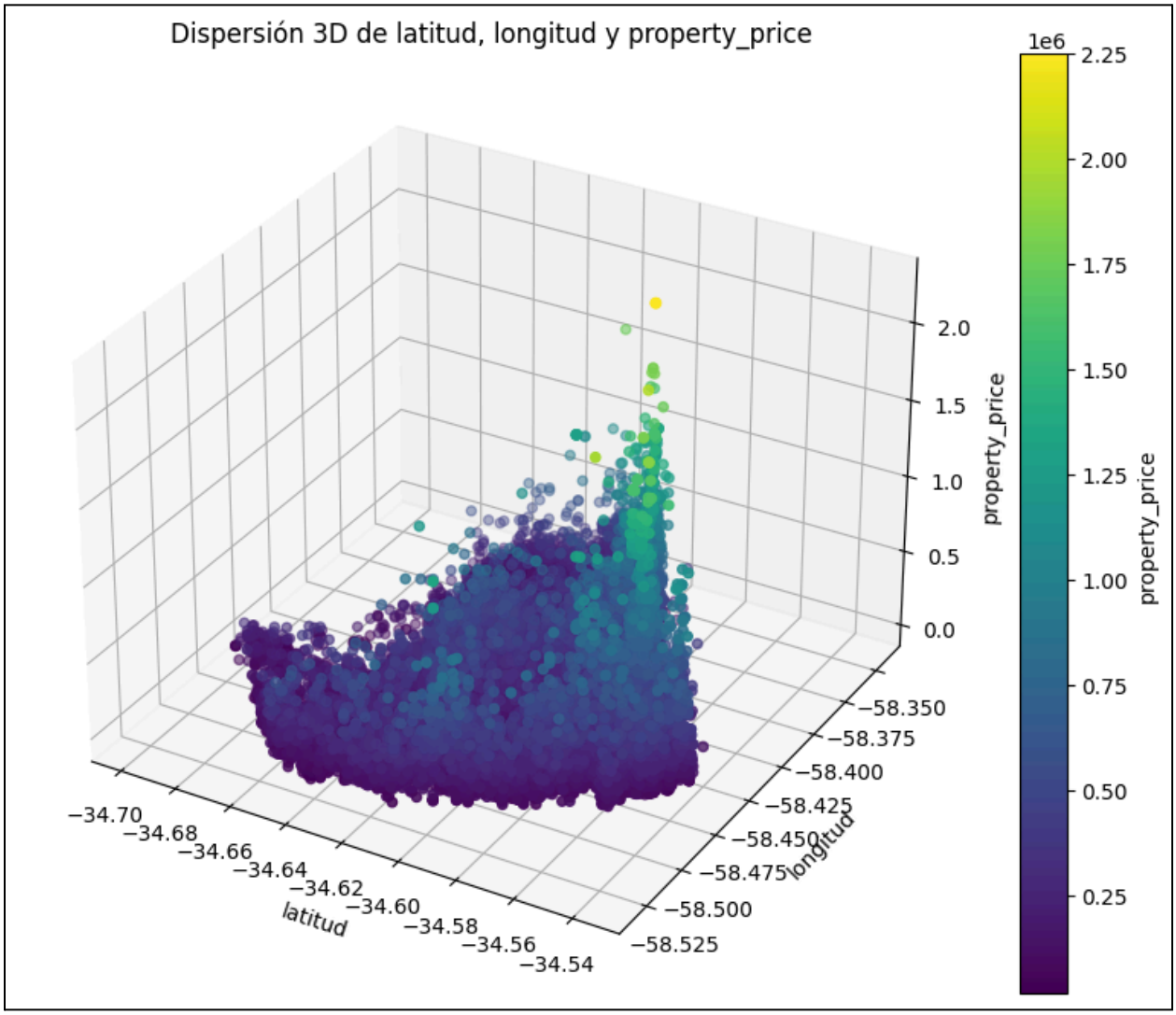
4. **Valores atípicos:**
    a. Superficie total menor a superficie cubierta: Este es un absurdo en el cual optamos por invertir los valores, asignando la superficie cubierta como superficie total y viceversa. Esto incrementó en grán medida la correlación entre estos datos para propiedades del tipo Casa.

b. Superficie total exageradamente grande: Haciendo un gráfico de dispersión entre la superficie total y la cubierta se ve un claro cúmulo en una esquina del gráfico. Eliminamos las publicaciones cuya superficie total es desde 7 veces mayor a la mediana.
c. Precio por metro cuadrado:
   En algunas propiedades obtuvimos valores por metro cuadrado irrisorios, de más de 5 veces la mediana inclusive. Las eliminamos.
d. Cantidad de habitaciones y de ambientes:
   En aquellos casos donde existen más habitaciones que ambientes asignamos el valor de ambientes al de habitación.
   Eliminamos aquellos registros donde la cantidad de habitaciones es más de 7 veces la mediana.

Visualizaciones



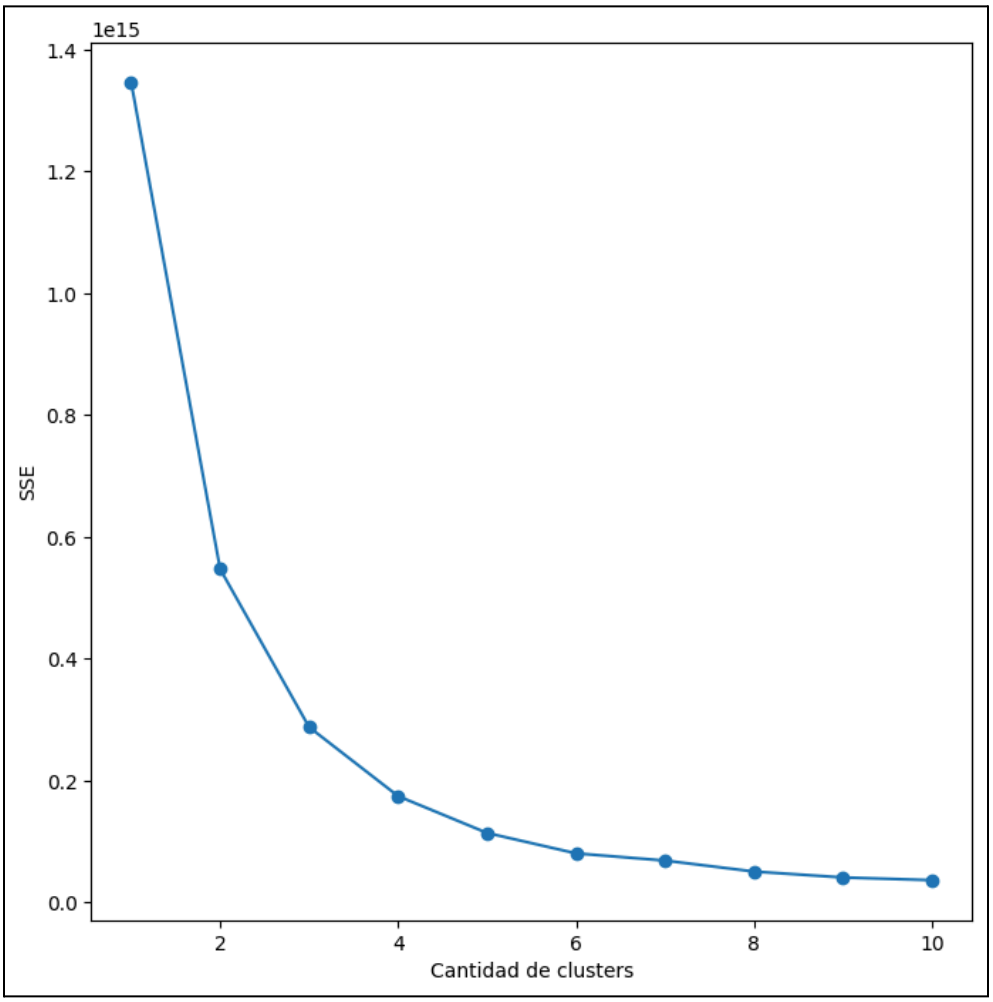
En el gráfico visualizamos la cantidad de muestras por “propety_type” la cual nos permite apreciar que en nuestro dataset poseemos una mayor distribución de Departamentos, lo cual es lo que uno esperaría de Capital Federal. Se eligió mostrar este gráfico ya que nos permite observar si un dato predomina sobre los otros y tenerlo más en cuenta por su peso



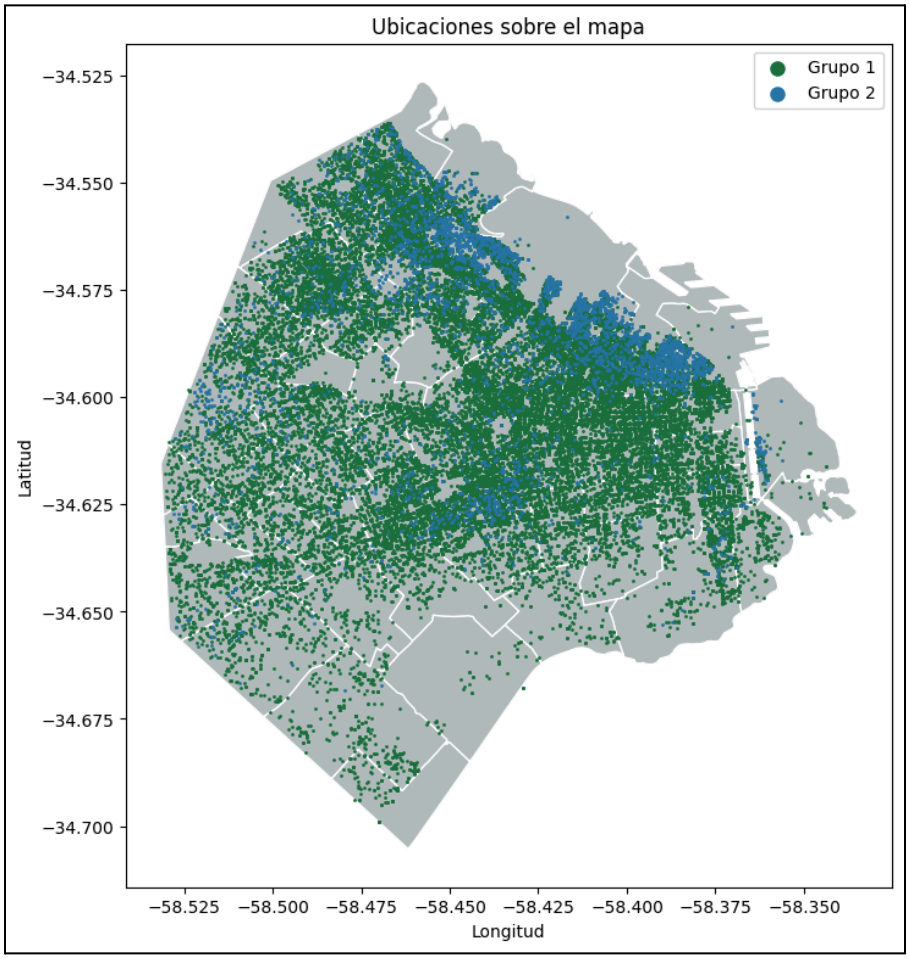
El gráfico permite visualizar la dispersión dimensional de las publicaciones gracias a los datos de latitud y longitud, así como también. Vemos un cúmulo importante de de datos con un amplio rango de precios en el sector de color más claro del gráfico. Mientras ciertas zonas mantienen un rango relativamente reducido en sus precios (eje vertical), este cúmulo más claro indica que esa ubicación (Palermo, seguido por Belgrano y Caballito) no tiene una fuerte tendencia del precio basado únicamente en su ubicación ya que el abanico de valores es más amplio.

Clustering

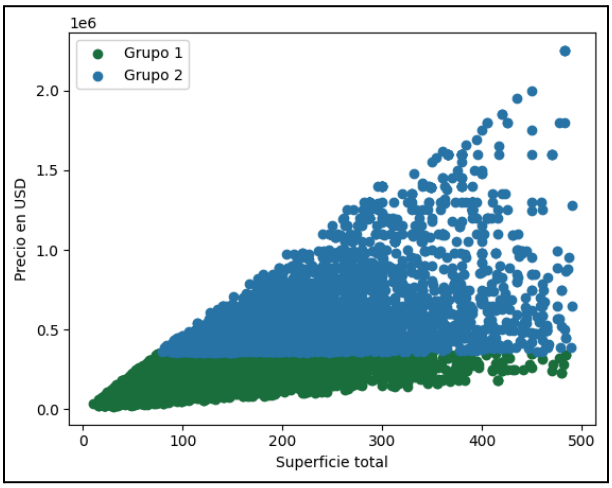
Buscamos con el SSE que cantidad de clusters sería apropiado para el dataframe dado, y observamos que el “quiebre” se da con clusters = 2.

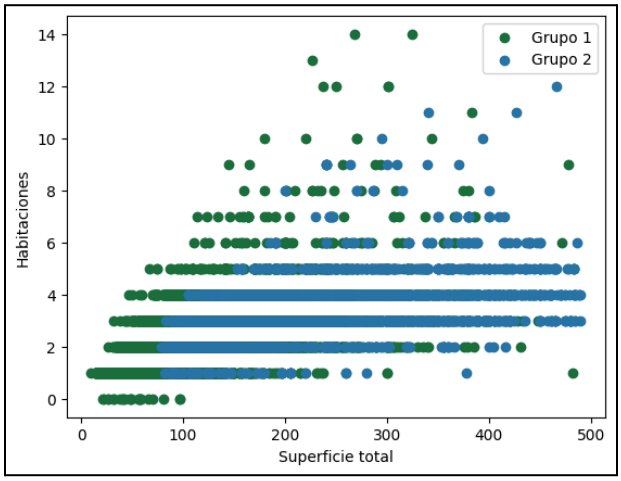


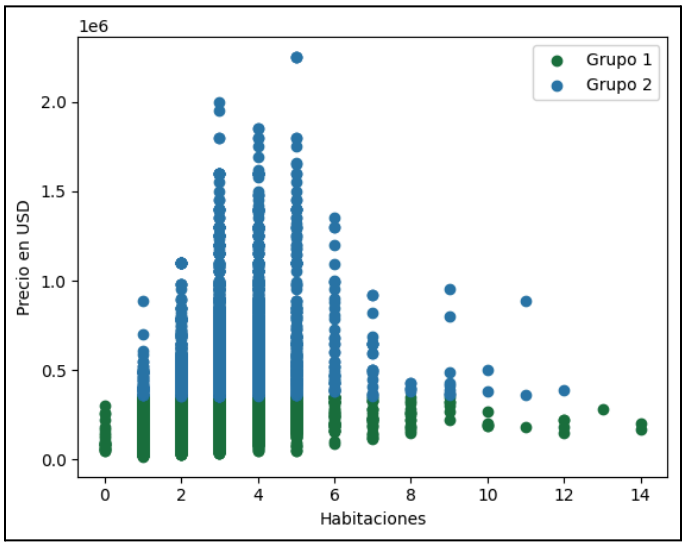
Luego utilizando el puntaje de silhouette contemplamos que efectivamente el que da mayor es con k=2. En el siguiente gráfico observamos la distribución de los grupos en el mapa de CABA.



En los siguientes gráficos mostramos la tendencia con la que se armaron los clusters







En ellos podemos concluir que los grupos fueron conformados según la variable *property_price*.

Clasificación

La variable *tipo_precio* fue construida con la alternativa de los cuartiles (sin diferenciación entre tipo de propiedad), y fue elegida así ya que probamos los modelos con las otras dos también, y respecto a la primera (separando en tercios) la diferencia era poca, pero era superior la elegida, y respecto a la de cuartiles pero separando por tipo de propiedad, daban mucho más bajo los scores.

Respecto a las similitudes con K-Means, observamos que los agrupamientos son similares, y se puede observar claramente en el mapa ploteado de CABA. Acerca de K-Means y en base a qué se habían formado los grupos, ya lo hemos comentado más arriba (en el CHP1), pero recordando que habían sido conformados en base al precio de la propiedad; por lo cual tiene todo el sentido que, siendo que *tipo_precio* depende del precio por metro cuadrado (lo cual tiene una relación directa con el precio de la propiedad), ambos mapas sean similares

a. Construcción del modelo

Se realizaron ejecuciones de los modelos Árbol de decisión, Random Forest y KNN, utilizando *RandomizedSearchCV* para optimizar la búsqueda de los hiperparámetros dividiendo el data frame en 5 folds. Esto se evaluó con las métricas de: Precision, Recall, Acuraccy y F1Score, las cuales se pueden observar en el **Cuadro de Resultados**

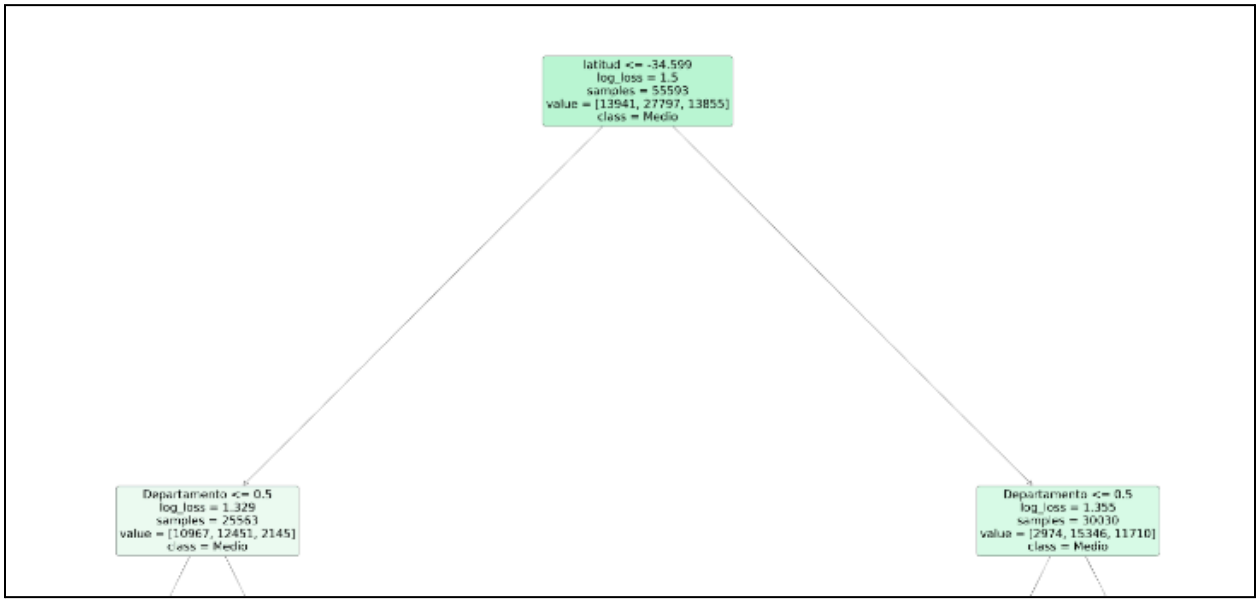
Árbol de Decisión

El modelo consiste de nodos los cuales deciden, valga la redundancia, entre varias opciones posibles. La idea es según la decisión de cada nodo, ir explorando a sus hijos y llegar a un nodo final (hoja) el cual indique a qué categoría pertenece el dato evaluado. Mientras más puro sea un nodo (menos entropía), es decir, mientras mejor decida, más importante debe ser en el rango del árbol (más cercano a la raíz).

Se buscaron optimizar los hiperparámetros de:

- **Criterion:** la función que se elige para medir la calidad de la división. Las opciones a elegir estaban la impureza de Gini, entropy y log_loss.
- **Min_samples_split:** determina el número mínimo de muestras necesarias para que un nodo se pueda dividir. Se usó un rango entre 10 y 15.
- **Ccp_alpha:** controla la poda del árbol. El rango usado va de 0 a 0.1.
- **Max_depth:** determina la profundidad máxima del árbol. El rango usado es entre 15 y 30.

Se usó RandomizedSearchCV con 5 folds. La métrica que se usó para buscar los hiperparámetros es accuracy



El árbol posee un max_deep de 27, por lo que se decidió mostrar una porción representativa. Se puede observar que en la clase Medio es la clase mayoritaria en los nodos, esto es algo esperado ya que es la clase que posee la mayor cantidad de muestras al elegir el método de clasificación. En el primer nodo, el primer atributo que se usa para dividir es la longitud, la cual es el atributo más importante. Para este nodo también podemos ver que hay unas 27297 muestras de la clase Medio de las 55593 del total. Para los siguientes nodos el atributo mayoritario es el de Departamento que se encuentra entre una de las más importantes.

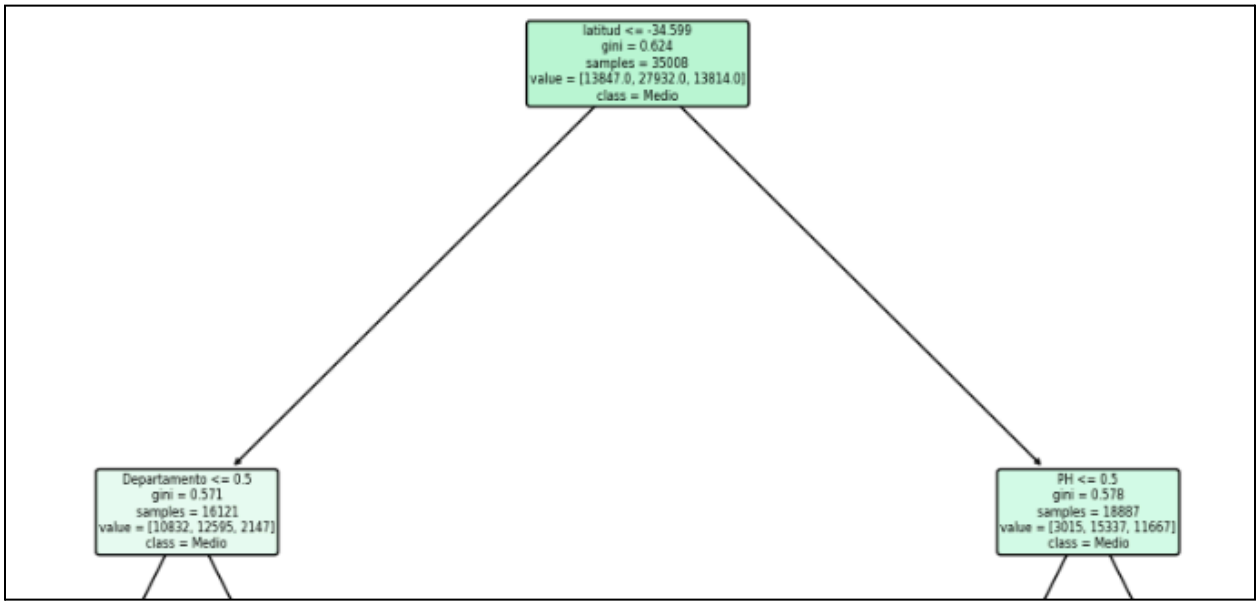
Random Forest

El modelo consiste en la combinación de distintos árboles de decisión, y utilizando la técnica de bagging, decide cuál es la categoría del registro a clasificar.

Se buscaron optimizar los parámetros de:

- **Criterion:** cumple la misma función que el hiperparámetro del árbol. Se probó con *gini* y *entropy*.
- **Min_samples_leaf:** mínimo de muestras para estar en una hoja. Se probó con 1, 5, 10.
- **Min_samples_split:** idem que criterion. Los valores a elegir fueron 2, 4, 10, 12, 16
- **N_estimators:** número de árboles del forest. Se probó con 10, 20, 50.
- **Max_depth:** idem el hiperparámetro del árbol pero se probó con el rango 15 a 50.

Se usó RandomizedSearchCV con 5 folds. La métrica que se usó para buscar hiperparámetros es accuracy.



Aquí mostramos el nodo raíz y sus dos hijos de uno de los árboles del mejor estimador del RF. El max_depth

Modelo a Elección (KNN)

Busca clasificar datos según la cercanía (similitud) que hay entre los mismos.

Se buscaron optimizar los parámetros de:

- **Weights:** determina la forma en la que influyen las distancias entre vecinos en el cálculo. Se probó con las opciones *uniform* y *distance*.

- **N_neighbors:** cantidad de vecinos por cada “centroide”. Disminuir mucho el valor puede llevar al overfitting, mientras que un aumento desmedido tiene el efecto contrario. Se probó con el rango de (1, 10).
- **Algorithm:** determina qué algoritmo se utiliza para identificar los nodos vecinos. Se probaron las opciones ["auto", "ball_tree", "kd_tree", "brute"].
- **P:** parámetro de distancia, siendo 1 equivale a usar la distancia de Manhattan, y 2 a usar la distancia Euclídea. Siendo estos valores las opciones a probar.

Se usó RandomizedSearchCV con 10 folds. La métrica que se usó para buscar hiperparámetros es accuracy.

Cuadro de Resultados

Medidas de rendimiento en el conjunto de Test:

Modelo	F1-Test	Precision Test	Recall Test	Accuracy Test
Arbol de Decision	0.7	0.7	0.7	0.7
Random Forest	0.76	0.76	0.76	0.75
KNN	0.69	0.69	0.70	0.69

Medidas de rendimiento en el conjunto de Train:

Modelo	F1-Test	Precision Test	Recall Test	Accuracy Test
Arbol de Decision	0.88	0.89	0.88	0.88
Random Forest	0.98	0.98	0.98	0.98
KNN	0.98	0.98	0.98	0.98

Nota: las medidas indicadas en los cuadros previos son los weighted avg de cada métrica.

Los hiperparámetros utilizados en cada modelo de clasificación son:

- Arbol de decision: {'min_samples_split': 10, 'max_depth': 26, 'criterion': 'entropy', 'ccp_alpha': 0.0}

- Random Forest: {'n_estimators': 50, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_depth': 40, 'criterion': 'gini'}

- KNN: {'weights': 'distance', 'p': 1, 'n_neighbors': 8, 'algorithm': 'auto'}

Regresión

a. Construcción del modelo

Se realizaron ejecuciones de los modelos KNN, XGBoost y LightGBM utilizando *RandomizedSearchCV* para optimizar la búsqueda de parámetros dividiendo el dataframe de entrenamiento en 5 folds.

Los data frames puestos a prueba y evaluados fueron previamente normalizados mediante *StandardScaler*. Esta normalización mejoró hasta en un 8% los resultados de la validación cruzada.

Esta evaluación se repitió con 3 métodos de scoring diferentes: MSE, RMSE y R2. Los mejores resultados de cada prueba por la validación cruzada utilizando los mejores parámetros se muestran en el apartado **Cuadro de resultados**.

KNN

El algoritmo, aplicado a un problema de regresión, busca predecir el valor de la variable objetivo en base a las propiedades de sus vecinos más cercanos en un espacio de características.

Algunos de los parámetros con los que se evaluó el modelo son los siguientes:

- **N_neighbors:** Cantidad de vecinos por cada “centroide”. Disminuir mucho el valor puede llevar al overfitting, mientras que un aumento desmedido tiene el efecto contrario.
- **Weights:** Determina la forma en la que influyen las distancias entre vecinos en el cálculo.
- **Algorithm:** Determina qué algoritmo se utiliza para identificar los nodos vecinos.
- **Metric:** Determina cómo se calcula la distancia entre vecinos. Algunos ejemplos son la distancia euclídea o la Manhattan.

XGBoost

El algoritmo utiliza una jerarquía de árboles de decisión donde cada árbol hijo aprende del error de su árbol padre para reforzar ese punto. Es un modelo muy eficiente para el trabajo con grandes cargas de datos.

Algunos de los parámetros con los que se evaluó el modelo son los siguientes:

- **N_estimators:** Número de árboles utilizados. Similar comportamiento al n_neighbors de KNN.
- **Max_depth:** Profundidad máxima de cada árbol.

- **Learning_rate:** Indica la taza con la que el modelo aprende los datos. Disminuirla implica un rendimiento más lento pero disminuye el riesgo de overfitting.
- **Reg_alpha:** Regula la complejidad del modelo. Aumentar el reg_alpha implica una menor influencia de las variables menos importantes, lo cual puede llevar a un underfitting.

LightGBM

Similar a XGBoost, este modelo se distingue por la forma en la que distribuye los datos de mayor peso en los árboles hijos. El modelo busca distribuir el peso de todas las variables evitando la concentración de XGBoost en las más relevantes.

Algunos de los parámetros con los que se evaluó el modelo son los siguientes:

- **border_count:** Determina el número de cortes máximo por cada característica del dataset. Aumentar este valor aumenta el tiempo de cómputo pero también aumenta la precisión.
- **begging_temperature:** .Regula la tasa de variabilidad en la selección de los datos que entrena a cada árbol del modelo. Es una propiedad de LightGBM e implementaciones del mismo, como CatBoost.
- **random_strenght:** Controla el peso de la aleatoriedad en el entrenamiento de los árboles. Aumentar el valor ayuda a ganar variabilidad y lograr un mejor rendimiento general.

b. Cuadro de Resultados

Modelo	MSE	RMSE	R2
KNN	-5761099671	-74518.06	0.8382
XGBoost	-6153214271	-77324.06	0.8319
LightGBM	-6082288721	-76924.30	0.8336

Para los métodos de scoring MSE y RMSE el valor más bajo es el mejor, mientras que en R2 los es el más próximo a 1.

Luego se evalúa cada uno de los modelos utilizando los mejores parámetros obtenidos en el paso anterior. Se evalúan los resultados con el conjunto de prueba y de entrenamiento para descartar casos de overfitting.
La métrica para puntuar este paso es R2.

Resultados en train y test

Modelo	Train	Test	Observaciones
KNN	0.998	0.885	Overfit en entrenamiento
XGBoost	0.903	0.823	Menor puntaje en test
LightGBM	0.888	0.869	Valores similares en entrenamiento y test

Esta prueba muestra una muy buena respuesta de los modelos al conjunto de test, incluso cuando parece existir un overfit con el de entrenamiento, como es el caso de KNN.
Por la cantidad e intensidad de las pruebas hechas, y por el funcionamiento de KNN, los tiempos de espera en su ejecución son demasiado largos. Por ello en la notebook dejamos comentada su ejecución, pero con los resultados en pantalla.
Dado que el modelo LightGBM obtiene un puntaje de test bastante similar al más alto (KNN), pero alejándose del overfit a los datos de prueba, optamos por avanzar con el mismo como modelo definitivo.
Además, vemos que la diferencia de score entre train y test de este modelo es muy pequeña, lo cual es un indicio de que el modelo es poco susceptible a los cambios en el dataframe a procesar.

Habiendo elegido el modelo, vamos a hacer una última optimización volviendo a evaluarlo pero con mayor cantidad de parámetros y más valores por cada uno.

Intento	Train	Test
Antes	0.888	0.869
Despues	0.908	0.866

Tras esto se puede apreciar un aumento en el score con el set de entrenamiento, y una ligera disminución en el score del set de evaluación. Claramente esto indica una tendencia al overfitting, por lo que vamos a conservar los parámetros de la evaluación original.

Mejores parámetros utilizados en LightGBM:

- random_strengt: 0.5
- learning_rate: 0.1
- l2_leaf_reg: 7
- depth: 6
- border_count: 32
- bagging_temperature: 1.5

Conclusiones Finales

Experiencia

Este trabajo práctico fue nuestro primer contacto con la ciencia de datos. Durante el desarrollo del mismo atravesamos distintas etapas que nos sirvieron tanto a nivel general para aprender a manipular datos y convertirlos en información aplicable en una problemática, como también a comprender el problema específico trabajado y mejorar nuestros modelos.

Entendimos el trabajo en la ciencia de datos como una experiencia de mejora progresiva, en la que cada nuevo aprendizaje puede aplicarse a lo previamente hecho para mejorarlo.

Comenzamos conociendo el dataset sin entender del todo la forma de interpretarlo. Identificamos las features del mismo y sus tipos. En ese punto comenzamos a elaborar hipótesis sobre qué campos podrían ser los más relevantes y cuales parecían carecer de relevancia. Con el paso del tiempo fuimos descartando las primeras hipótesis y proponiendo nuevas, que otra vez eran reemplazadas por nuevas acorde a la información que íbamos obteniendo.

Como grupo supimos distribuir las tareas y complementarnos de forma tal que los 3 tengamos un total conocimiento y entendimiento de lo hecho, además de poder ayudarnos mutuamente y tener distintos puntos de vista.

Esta iteración continua se aplicó también sobre las imputaciones y la eliminación de registros. De acuerdo a pruebas con modelos y a gráficos que fuimos realizando fuimos puliendo el tratamiento de outliers y valores nulos. Pocas cosas del trabajo práctico no fueron sujetas a cambio una vez pensadas.

En la recta final entrenamos variedad de modelos utilizando distintas combinaciones de hiper parámetros y amoldando el dataset de distintas formas (agrupando los registros por tipo de propiedad, normalizando columnas, entre otras transformaciones) para evaluar rendimientos. En esta etapa nos ayudamos construyendo funciones que generalizan la creación, entrenamiento y evaluación de modelos para poder estandarizar las salidas y simplificar el entendimiento de la notebook.

Para finalizar nos concentramos en aquellos modelos que mejores resultados nos ofrecieron y nos centramos en pulirlos para obtener resultados con los que nos sentimos a gusto. Debido a limitaciones en nuestros tiempos nos quedaron cosas por probar, que si bien estamos conformes con lo obtenido, nos incitan a seguir profundizando en la materia, y en el área de la ciencia de datos de por sí.

Conclusiones de la experimentación

El desarrollo del trabajo fue de mejora continua. Es decir, mediante fuimos experimentando y haciendo pruebas fuimos ajustando datos, parámetros, modelos, y demás.

Entre las pruebas realizadas, algunas significaron una mejora para el trabajo por lo que terminaron siendo definitivas. Entre ellas podemos mencionar las siguientes:

- Elección de modelos: Tras evaluar el dataset en distintos modelos y utilizando variedad de hiper parámetros, terminamos seleccionando un mejor modelo para la clasificación y otro mejor modelo para la regresión.
- Elección de hiper parámetros: Estos modelos una vez seleccionados fueron sometidos a pruebas con mayor cantidad de hiper parámetros donde ajustamos hasta detectar que se estaba llegando a un overfitting.
- Evaluar superficies por tipo de propiedad: Otra de las pruebas consistió en evaluar correlaciones y outliers agrupando por tipo de propiedad en lugar de utilizar el general de los datos. La idea surgió de una de nuestras hipótesis donde suponíamos que esta relación en departamentos debería ser casi exacta ya que no suelen tener espacios abiertos, a diferencia de las casas o PH's. Estas hipótesis surgen de un mínimo conocimiento del negocio evaluado.
- Preparación del target: Otro punto de prueba consistió en la preparación del target para la clasificación. Probamos distintas formas de establecer los límites de las clases objetivo, agrupando o no por tipo de propiedad y estableciendo distintos umbrales. Todas las formas fueron evaluadas con todos los modelos de clasificación que probamos, y seleccionamos la que mejores resultados nos dió en general.

Respecto a otras opciones que hubiéramos tomado a lo largo del trabajo, nos hubiera gustado entre otras cosas:

- Imputar latitudes y longitudes: con el método Cold Deck podríamos haber imputado el centro de cada barrio, con el dataframe otorgado por el gobierno de CABA, indicado en la columna *place_l3*, e incluso en el caso de las viviendas en Palermo mayormente, tambien podriamos haber imputado los centros de los *place_l4*.

- Imputar superficies por regresión: esto lo podríamos haber hecho con el dataframe entero, discriminando por place_l2, discriminando por *place_l2* y cantidad de habitaciones, y demás combinaciones posibles para ser lo más precisos posibles.
- Imputar barrios con la columna, eliminada inicialmente, *property_title*: en ella en algunos casos el texto indicaba algo del estilo “VENDO CASA EN PARQUE CHACABUCO” o leyendas similares, y confiando ciegamente en estas afirmaciones hechas por el vendedor a la hora de publicar la vivienda, podríamos haber imputado los barrios de cada vivienda que dispusiera información similar.
- Agregar features nuevas: como distancia a trenes, bocas de subte, distancia a espacios verdes y otros detalles que evidentemente en la realidad elevan el precio de una vivienda.

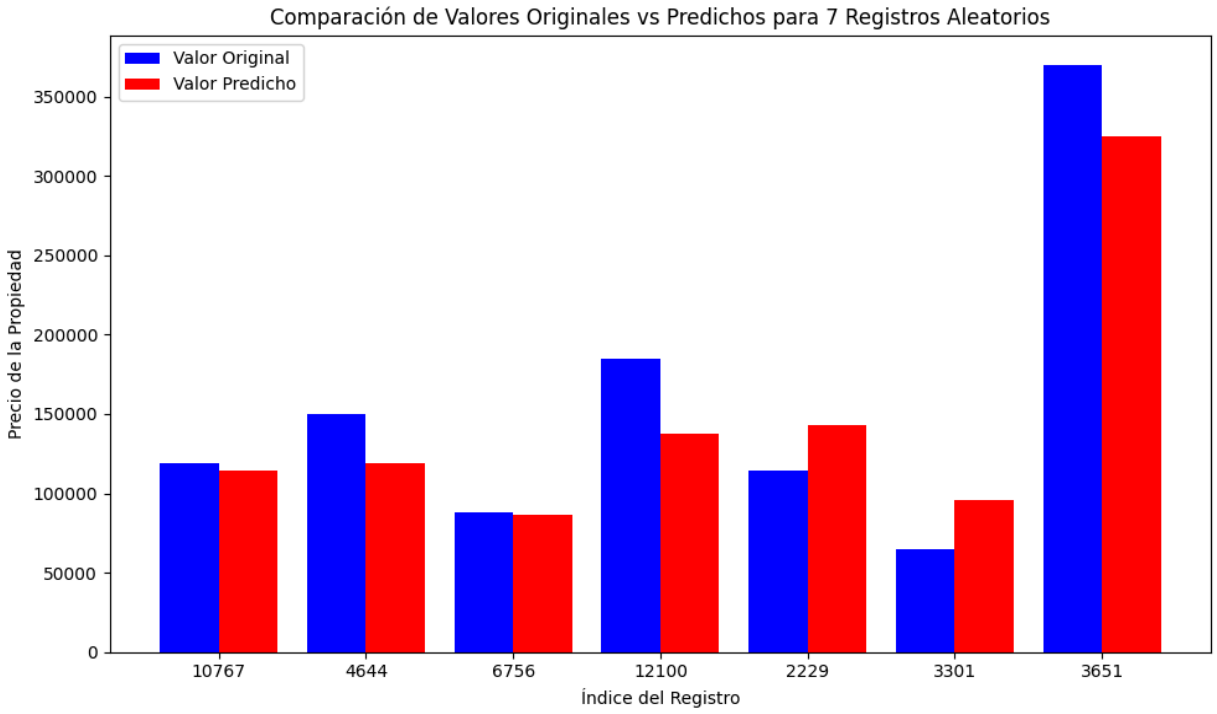
Conclusiones modelos

Mejor modelo clasificación: Random Forest

El modelo con el que mejor resultado hemos obtenido ha sido el Random Forest tras comparar con los otros modelos especificados en la consigna, y con otros modelos probados que no terminamos implementando (para el caso del modelo a elección). Es de esperarse en parte ya que termina decidiendo entre varios árboles y por ello suele llegar a mejores resultados, utilizando el ya mencionado bagging, que como se ha visto en la materia, busca ajustar los modelos cada vez más, poniendo cierto corte para no llegar al overfitting.

Mejor modelo regresión: LightGBM

En el caso de la regresión el mejor modelo fue LightGBM. Se evaluó a la par con KNN y XGBoost, y por lo visto en el apartado *resultados en train y test* nos decidimos por el primero.
Para concluir tomamos el modelo con sus mejores parámetros obtenidos, lo entrenamos con los datos de entrenamiento normalizados y comparamos los resultados de 7 registros aleatorios del set de test con los valores predichos por el modelo.



Tiempo dedicado

Integrante	Tarea	Prom. Hs Semana
Taiel Molina	Clasificación (Random Forest) Regresión (KNN) Armado de Reporte	7
Ignacio Fernandez	Clasificación (KNN) Regresión (XGBoost y LightGBM) Armado de Reporte	7
Sebastián Vera	Clasificación (Árbol de Decisión) Clasificación (KNN) Armado de Reporte	7