

Integrante del grupo

-Beitia German

Presentacion de la organizacion

Properati es una empresa de gestion inmobiliaria destinada a favorecer el contacto entre vendedores y compradores (o entre arrendadores y arrendatarios) de distintos tipos de propiedades, trabajando desde plataforma web en Argentina, Colombia, Ecuador, Perú y Uruguay. También cuenta con versiones mobile, con funcionalidades especialmente diseñadas para estos dispositivos, como la búsqueda de propiedades cerca de la ubicación actual y una navegación simple. De esta manera, se pueden observar los precios publicados de los distintos tipos de viviendas y se fomenta así la interacción entre las partes.

Preguntas y objetivos de la investigación

Debido a que muchos de los anuncios publicados en esta pagina cuentan con datos como superficie, numero de habitaciones, numero de baños, etc; es interesante el planteo de modelos de Machine Learning que nos permitan predecir precios de distintos tipos de propiedades (según el tipo de operación del que se trate) utilizando estas variables. Esto puede ser útil como estimador de precio de operaciones nuevas. Es decir, cuando el propietario desee publicar un anuncio, tener un marco de referencia del precio de la operación, ya que para los tasadores la asignación de un precio resulta dificultoso y muchas veces, subjetivo.

Como los precios varían de una región a otra, o de un tipo de vivienda a otra, probablemente sea necesaria la implementación de modelos específicos según el interés de lo que se quiere predecir. Para esto será necesario realizar segmentaciones según distintas variables para obtener modelos precisos para una predicción en particular, y no un modelo general que funcione para predecir cualquier precio en cualquier circunstancia.

✓ Indicación de la fuente del dataset y los criterios de selección

Los datos fueron extraídos de <https://www.properati.com.ar/data/>, la base de datos con los distintos avisos publicados con los que cuenta la empresa. Estos datos se actualizan constantemente.

Los datos seleccionados para el presente trabajo corresponden solo al país Argentina, y se realizó una reducción de registros, ya que originalmente se contaba con más de 2 millones de datos, que se redujeron a aproximadamente 60 mil. Estos registros fueron seleccionados de manera aleatoria, de manera que se mantuviera la representación de las distintas variables del dataset original.

El archivo de excel resultante fue subido a la página GitHub, desde donde se accede mediante importación.

Importación de paquetes En esta sección importaremos todos los paquetes que utilizaremos, tanto para el análisis inicial de los datos como para la predicción y la generación de modelos de Machine Learning

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib
4 import seaborn as sns
5 import matplotlib.pyplot as plt
6 from matplotlib import cm
7 import warnings
8 import missingno as msno
9 import datetime as dt
10 from pandas.api.types import is_numeric_dtype
11 import statsmodels.api as sm
12 import pylab as py
13 import random
14 import time
15
16 from sklearn import tree
17 from sklearn.model_selection import StratifiedKFold, cross_val_score, train_test_split
18 from sklearn.ensemble import RandomForestRegressor
19 from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, mean_squared_error, r2_score
20 from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
21 from lightgbm.sklearn import LGBMRegressor
22 from sklearn.ensemble import GradientBoostingRegressor
23 from sklearn.ensemble import AdaBoostRegressor
24
25 import statsmodels.api as sm
26 import statsmodels.formula.api as smfs
```

```
27 from xgboost import XGBRegressor
28
29 warnings.filterwarnings('ignore')
30
31 !pip install joypy
32 from joypy import joyplot
33
34 !pip install sidetable
35 !pip install geoplot
36 import sidetable
37 import geopandas as gpd
38
39 !pip install colorama
40 from colorama import Fore
41 from scipy.stats import randint
42
43 !pip install git+https://github.com/scikit-optimize/scikit-optimize.git
44 from skopt import BayesSearchCV
```

 [Mostrar el resultado oculto](#)


Importacion de dataset

Importaremos el dataset a utilizar desde un repositorio Github, donde se encuentra almacenada como archivo.xlsx. Luego guardamos ese dataset como un DataFrame para comenzar a trabajar


```
1 Comienza a programar o generar con IA.

1 url = "https://github.com/gonzalotulin/Proyecto-DataScience/blob/4cb486dac077c49c88331bd4a20f4d18f114b993/propiedades4.xlsx"
2 viviendas = pd.read_excel(url)
3

1 #Analizamos cual es la forma de este conjunto de datos
2
3 print(f'El dataset viviendas posee {viviendas.shape[1]} columnas y {viviendas.shape[0]} filas')
4
```

 El dataset viviendas posee 23 columnas y 54083 filas

```
1 #Imprimimos el nombre de todas las columnas
2 print('Las columnas del dataset son: {}'.format(viviendas.columns))
3
4 #Para tener una primera imagen visual del dataset con el que trabajaremos, mostramos los primeros datos
5 display(viviendas.head())
```

 Las columnas del dataset son: Index(['start_date', 'end_date', 'created_on', 'Latitud', 'Longitud', 'l1', 'l2', 'l3', 'l4', 'l5', 'l6', 'rooms', 'bedrooms', 'bathrooms', 'surface_total', 'surface_covered', 'price', 'currency', 'price_period', 'title', 'description', 'property_type', 'operation_type'], dtype='object')

	start_date	end_date	created_on	Latitud	Longitud	l1	l2	l3	l4	l5	...	bathrooms	surface_total	su
0	2020-08-22	2020-09-04 00:00:00	2020-08-22	-37.996039	-57.542509	Argentina	Buenos Aires Costa Atlántica	Mar del Plata	NaN	NaN	...	NaN	687	
1	2020-08-22	2020-08-31 00:00:00	2020-08-22	-31.380200	-58.009200	Argentina	Entre Ríos	Concordia	NaN	NaN	...	1.0	80	
2	2020-08-22	2020-09-04 00:00:00	2020-08-22	-32.948900	-60.630500	Argentina	Santa Fe	Rosario	NaN	NaN	...	1.0	76	
3	2020-08-22	2020-09-04 00:00:00	2020-08-22	-32.884278	-60.710901	Argentina	Santa Fe	Rosario	NaN	NaN	...	1.0	39	
4	2020-08-22	2020-09-04 00:00:00	2020-08-22	-32.948800	-60.630200	Argentina	Santa Fe	Rosario	NaN	NaN	...	2.0	130	
5 rows × 23 columns														

Como podemos observar, el dataset contiene gran cantidad de variables (23) por lo que lo primero que hacemos es identificarlas y mostrar el tipo de dato de cada una de ellas. A su vez vamos a obtener informacion de interes como los valores faltantes, los valores nulos y si la

variable es o no numerica. Sumado a esto adicionamos al mismo dataframe algunas variables estadisticas de interes

✓ Descripción de variables

Object

end_date = fecha de finalizacion del anuncio

I1 - Nivel administrativo 1: país.

I2 - Nivel administrativo 2: usualmente provincia.

I3 - Nivel administrativo 3: usualmente ciudad.

I4 - Nivel administrativo 4: usualmente barrio.

currency: divisa en la cual se informa el precio

price_period: periodo que abarca el precio (para alquileres)

title : Título del anuncio.

description: Descripción del anuncio.

property_type: tipo de propiedad (Casa, Departamento, PH,etc).

operation_type: tipo de operacion (venta, alquiler, alquiler temporal)

datetime64[ns]

start_date: Fecha de alta del aviso.

created_on - Fecha de alta de la primera versión del aviso.

float64

Latitud: latitud de la vivienda

Longitud: longitud de la vivienda

rooms: Cantidad de ambientes (útil en Argentina).

bedrooms: Cantidad de dormitorios (útil en el resto de los países).

bathrooms: Cantidad de baños.

surface_total: Superficie total en m².


surface_covered: Superficie cubierta en m².

price: Precio publicado en el anuncio.

```

1
2 def univariado_info(df):
3     '''Calculo de informacion estadisticas y genericas de cada columna de un dataframe'''
4
5     #create a dataframe with especificas columnas
6
7     df_info = pd.DataFrame(columns=['Cantidad', 'Tipo', 'Missing', 'Unicos', 'Numeric'])
8     #loop de todas las variables del dataframe
9     for col in df:
10
11         #obtengo info de la columna
12         data_series = df[col]
13         #lleno dataframe con las columnas iniciales
14         df_info.loc[col] = [data_series.count(), data_series.dtype, data_series.isnull().sum(), data_series.nunique(), is_nur
15
16     #calculo el describe
17     df_describe = df.describe(include='all').T[['top', 'mean', 'std', 'min', '25%', '50%', '75%', 'max']]
18     #calculo sesgo y curtosis
19     df_stats = pd.DataFrame([df.skew(), df.kurtosis()], index=['sesgo', 'kurt']).T
20
21     return pd.concat([df_info, pd.concat([df_describe, df_stats], axis=1)], axis=1).fillna('-')
22
23
24 df_uni_stats = univariado_info(viviendas)
25 df_uni_stats

```



	Cantidad	Tipo	Missing	Unicos	Numeric	top	mean	std	min	:
start_date	54083	datetime64[ns]	0	81	False	2020-08-08 00:00:00	-	-	-	
end_date	54083	object	0	448	False	2021-06-05 00:00:00	-	-	-	
created_on	54083	datetime64[ns]	0	81	False	2020-08-08 00:00:00	-	-	-	
Latitud	49112	float64	4971	33426	True	-	-33.105307	22.095086	-54.808646	-34.6281
Longitud	49112	float64	4971	33914	True	-	-59.360217	2.826661	-95.712891	-58.7131
I1	54083	object	0	4	False	Argentina	-	-	-	
I2	54083	object	0	33	False	Capital Federal	-	-	-	
I3	52267	object	1816	434	False	Palermo	-	-	-	
I4	16074	object	38009	540	False	Nordelta	-	-	-	
I5	326	object	53757	19	False	BarrioPortezuelo	-	-	-	
I6	0	float64	54083	0	True	-	-	-	-	
rooms	43480	float64	10603	28	True	-	3.073965	1.826838		1.0
bedrooms	38167	float64	15916	35	True	-	2.257238	1.64599		1.0
bathrooms	49428	float64	4655	17	True	-	1.695314	1.096322		1.0
surface_total	54083	int64	0	1718	True	-	362.880591	3174.010813		10.0 4
surface_covered	54083	int64	0	1316	True	-	237.187545	2368.04462		-2.0 4
price	51634	float64	2449	3674	True	-	295271.420033	1455899.64009		0.0 4000
currency	51467	object	2616	4	False	USD	-	-	-	
price_period	28977	object	25106	1	False	Mensual	-	-	-	
title	54080	object	3	39586	False	Departamento - La Plata	-	-	-	

1 Y 2 AMBIENTES

✓ Analisis preliminar:

Entre las variables, tenemos como metadatos: 'title' y 'description'. Estas columnas contienen informacion respecto a cada vivienda que no pueden utilizarse en el analisis, ya que en principio cuentan con informacion que podemos recolectar de otras columnas (al menos que en algun caso en particular haya datos faltantes). Por lo que estas variables se eliminan

La variable target 'precio' tiene un sesgo muy alto, y el desvio estandar es mucho mayor a la media, por lo que probablemente hay anomalias o valores muy concentrados, será necesario hacer transformaciones para trabajar.

Columnas como I4,I5 y I6 tienen altísima cantidad de nulos que no puedo rellenar por falta de información, por lo que se eliminarán estas variables

Observamos que la variable price_period tiene un unico valor a lo largo de todos los registros. No es una variable, por lo que se elimina esta columna

Vemos que la variable end_date, que corresponde a una fecha, es de tipo object. Por lo que es necesario convertirla en datetime. Sin embargo, muchos de estas variables contienen fechas con año 9999 (indicando que el anuncio aun continua en curso). Como pandas no puede manejar estos valores, para que sea mas sencillo el posterior trabajo, vamos a convertir estas fechas en la fecha actual.

La variable bedroom no es util para propiedades en Argentina,(esto lo dice la descripcion del dataset) ya que incluye solo las habitaciones como tales. El dataset cuenta con la variable rooms, que es lo que nosotros conocemos como ambientes. Asi, nos quedaremos solo con la variable room(ambientes) y bathroom(baño), y eliminamos la variable bedrooms (habitacion de dormir).

```
1 #Eliminamos las columnas title, description , price_period, I4, I5, I6, y bedrooms por los motivos detallados en el analisis:
2 viviendas = viviendas.drop(columns = ['title', 'description', 'price_period','I4', 'I5', 'I6', 'bedrooms'])
3 viviendas.columns

Index(['start_date', 'end_date', 'created_on', 'Latitud', 'Longitud', 'I1',
      'I2', 'I3', 'rooms', 'bathrooms', 'surface_total', 'surface_covered',
      'price', 'currency', 'property_type', 'operation_type'],
      dtype='object')
```

```
1 #En el dataset hay fechas en la columna end_date que tienen el año 9999, como indicacion de que el anuncio no tiene fecha :
2 #Como pandas no puede convertir estas fechas a datetime, se indica el argumento errors ='coerce', que convierte estos valores a NaT
3
```

```

4 viviendas['end_date'] = pd.to_datetime(viviendas['end_date'], errors = 'coerce')
5
6 #Como no queremos tener las fechas de los anuncios no finalizados en nulo, los convierto a la fecha actual
7 hoy = pd.to_datetime('today')
8 viviendas['end_date'].fillna(hoy, inplace = True)
9
10 #Confirmamos que haya cambiado el tipo de dato
11 viviendas.info()
12

```

```

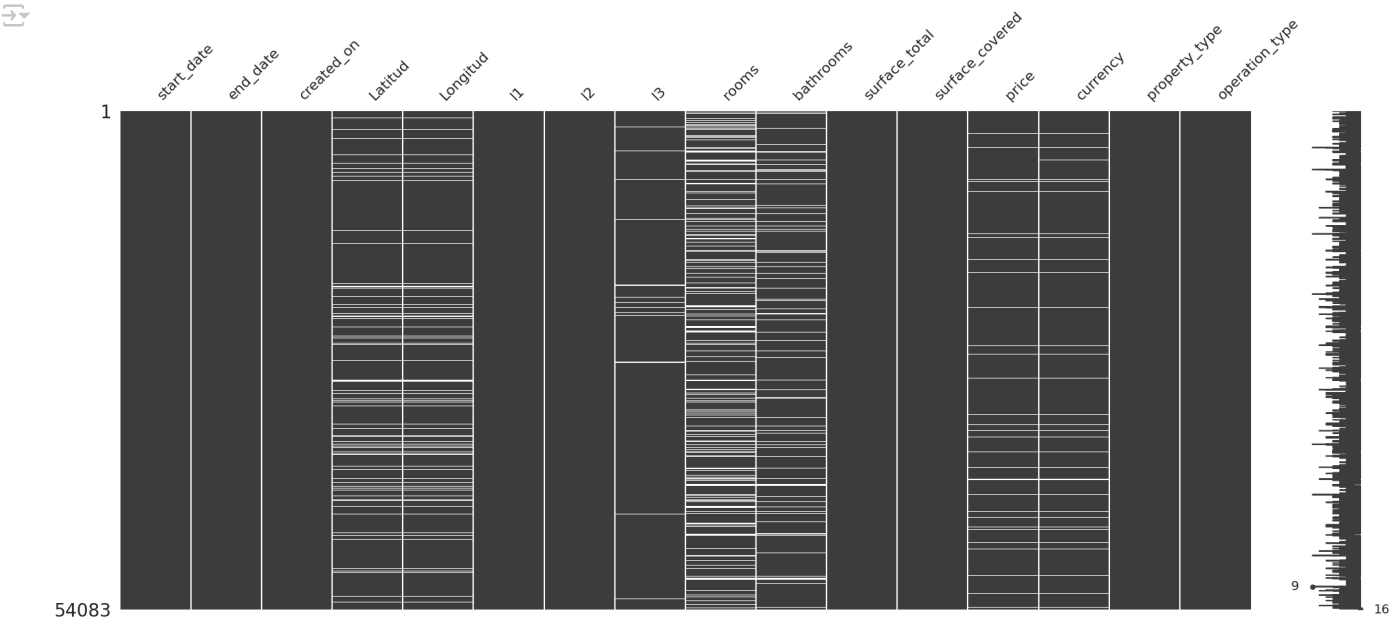
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54083 entries, 0 to 54082
Data columns (total 16 columns):
#   Column                Non-Null Count  Dtype
---  -
0   start_date            54083 non-null  datetime64[ns]
1   end_date              54083 non-null  datetime64[ns]
2   created_on            54083 non-null  datetime64[ns]
3   Latitud               49112 non-null  float64
4   Longitud              49112 non-null  float64
5   l1                    54083 non-null  object
6   l2                    54083 non-null  object
7   l3                    52267 non-null  object
8   rooms                 43480 non-null  float64
9   bathrooms             49428 non-null  float64
10  surface_total         54083 non-null  int64
11  surface_covered       54083 non-null  int64
12  price                 51634 non-null  float64
13  currency              51467 non-null  object
14  property_type         54082 non-null  object
15  operation_type        54082 non-null  object
dtypes: datetime64[ns](3), float64(5), int64(2), object(6)
memory usage: 6.6+ MB

```

✓ Analisis de datos nulos

Como ya determinamos anteriormente, existen datos nulos en nuestro dataset, para tener una visualización mas gráfica, recurrimos a una matriz utilizando la libreria missigno

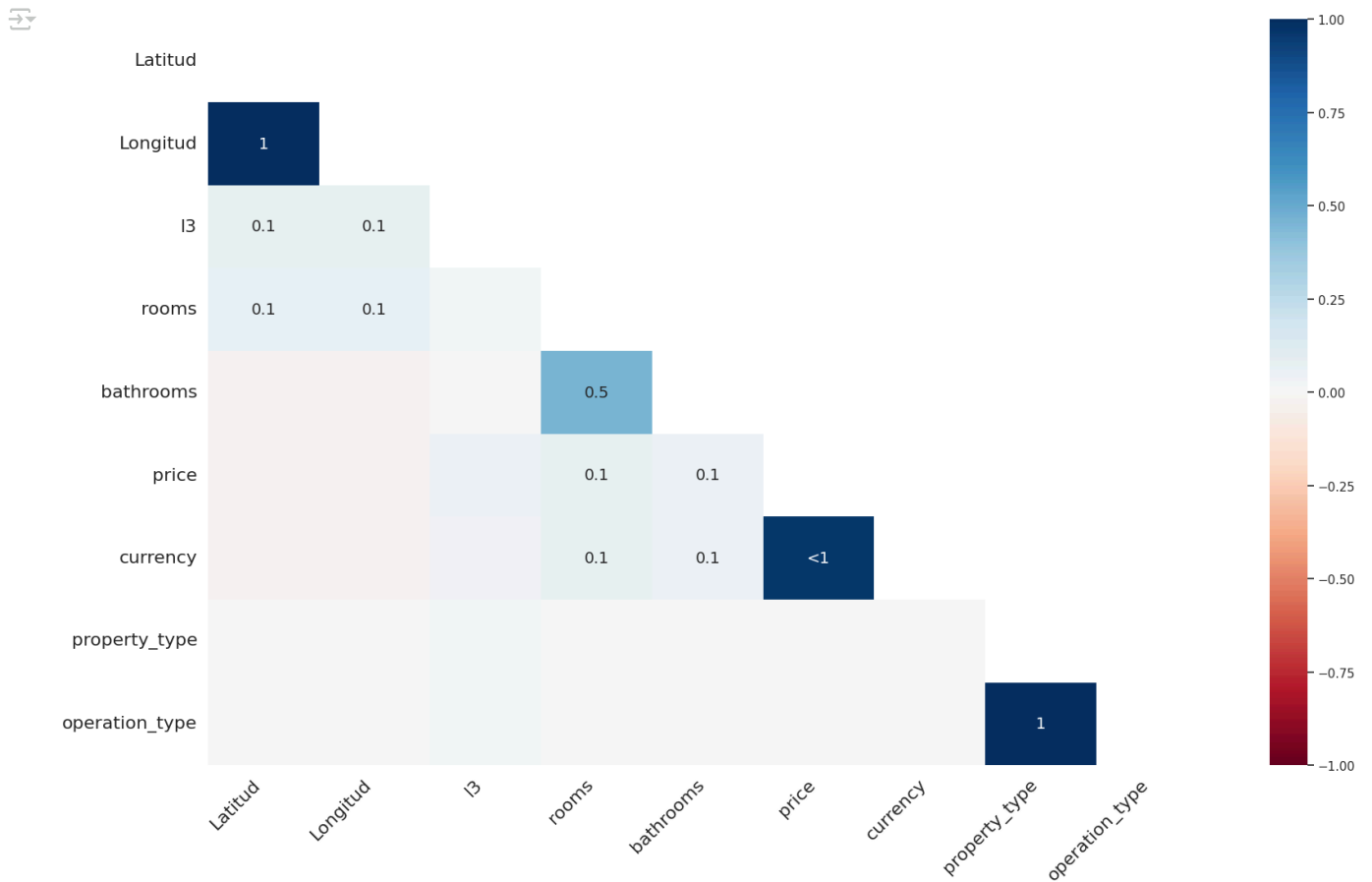
```
1 msno.matrix(viviendas);
```



Observamos que tenemos gran numero de nulos en variables como 'rooms' y 'bathrooms'que indican el numero de ambientes y el numero de baños respectivamente. Como estas variables son importantes para las predicciones, posteriormente deberán llenarse

Realizamos un grafico de heatmap para analizar relacion entre los valores nulos de las distintas variables

```
1 msno.heatmap(viviendas);
```



Cuando realizamos el heatmap de nulos, vemos que la correlacion entre latitud y longitud es igual a 1, lo que nos indica que en aquellos registros donde tenemos el dato de la latitud, disponemos del de la longitud. Lo mismo sucede con precio y tipo de moneda. A su vez, la relacion entre rooms y bathrooms de 0.5 nos indica que cada vez que tengamos un nulo de room, tenemos el 0.5 de probabilidad de tener un nulo en bathroom (o dicho de otra manera, tenemos el doble de nulos de room que bahtrooms en el dataset), algo que veíamos también en la matriz.

✓ Chequear duplicados en el dataset

Analizamos primero si hay algún registro que esté totalmente duplicado (es decir, todos los valores de todas las columnas sean iguales en dos o mas registros)

```
1 viviendas.duplicated().value_counts()
```

```
False    52936
True      1147
dtype: int64
```

Como vemos que hay valores duplicados en el dataset (probablemente ocasionados por la publicacion de una misma vivienda con distinta descripcion), los eliminamos y nos quedamos con el ultimo.

```
1 viviendas = viviendas.drop_duplicates (keep = 'last')
```

Como las variables *created_on* (fecha de alta de la primera versión del aviso) y *start_date* (fecha de alta del aviso) pueden ser similares en muchos registros, analizamos si hay valores duplicados a lo largo de estas dos columnas

```
1 viviendas.duplicated(subset = ['created_on', 'start_date']).value_counts()
2
```

```
True      52855
False       81
dtype: int64
```

Observamos que casi la totalidad de estas dos columnas corresponden a datos duplicados. Por lo tanto eliminamos la columna *created_on*

```
1 viviendas = viviendas.drop(columns = ['created_on'])
2
```

✓ Limpieza de datos

Como solo trabajaremos con propiedades de Argentina, eliminamos propiedades de cualquier otro pais.

```
1 print(viviendas['l1'].value_counts())
2 #Vamos a trabajar solo con viviendas en Argentina, por lo que eliminamos los registros que contengan otros paises
3 viviendas = viviendas[viviendas['l1']== 'Argentina']
```

```
Argentina      51377
Uruguay         1371
Estados Unidos   180
Brasil           8
Name: l1, dtype: int64
```

```
1 #Visualizamos los distintos tipos de moneda que tenemos
2 viviendas.groupby('operation_type')['currency'].value_counts()
```

```
operation_type  currency
Alquiler        ARS      10180
                 USD      1724
                 PEN         7
Alquiler temporal ARS      1341
                 USD       940
Venta            USD     33998
                 ARS       719
                 PEN         5
Name: currency, dtype: int64
```

Agrupando por tipo de operacion y viendo el tipo de moneda, vemos que la mayoría de los alquileres estan en pesos, mientras que la mayoría de las ventas estan en dolares.

Nos vamos a quedar solo con aquellas operaciones que estan en dolares o en pesos

```
1 print(viviendas['currency'].value_counts())
2
3 #Hay 12 registros que tienen tipos de monedas distintos a pesos y dolares(PEN). Eliminamos estos registros
4
5 viviendas = viviendas[(viviendas['currency'] == 'USD') | (viviendas['currency'] == 'ARS')|(viviendas['currency'].isna())]
6
```

```
USD      36663
ARS     12240
PEN        12
Name: currency, dtype: int64
```

En este caso, para hacer comparaciones y posteriores modelados, necesitamos que todos los precios esten expresados en la misma moneda. Para esto, vamos a expresar todos los precios en dolares. Utilizamos la API *exchangerate.host*, que permite realizar conversiones entre monedas al precio actual.


```

1 #API para realizar conversion de monedas
2 import requests
3
4 #Calculamos cual es el precio actual en pesos de un dolar
5 url = 'https://dolarapi.com/v1/dolares/blue'
6 response = requests.get(url, params = {'amount' : 1})
7 conversion = response.json()
8
9 precio_dolar = conversion['compra']
10 precio_dolar
11 #Se divide por 0.65 para agregarle el impuesto pais y el impuesto del 35%

```

↻ 1015

```

1 #Divido todos los valores que tengo en pesos por el valor del dolar que me devuelve la API
2 viviendas.loc[viviendas['currency']== 'ARS', 'price'] = viviendas.loc[viviendas['currency']== 'ARS', 'price']/precio_dolar
3 viviendas = viviendas.replace({'currency':'ARS'}, 'USD')

```

✓ Llenado de nulos

Variable target

Para llenar los precios nulos, vamos a agrupar el dataset segun la localizacion (l1, l2 y l3) y segun el tipo de vivienda y tipo de operacion. Llenaremos estos datos con la mediana de cada grupo, ya que se verá posteriormente que las distribuciones están muy sesgadas y tenemos bastantes registros con valores extremos, por lo que usar la media en este caso conducira a mucho error.

Sin embargo, esto no se podra hacer para aquellos registros en los que no disponemos ni del precio ni de la localizacion, por lo que eliminaremos estos registros primero

```

1
2 filtro = viviendas[(viviendas['l3'].isna()) & (viviendas['price'].isna())]
3 print(filtro.shape)
4 #Vemos que tenemos 145 registros donde los valores de l3 y de price son nulos
5
6 indices = []
7 ind = filtro.index
8 for i in ind:
9     indices.append(i)
10
11 viviendas = viviendas.drop(index = indices)
12

```

↻ (145, 15)

```

1 viviendas['price'] = viviendas['price'].fillna(viviendas.groupby(['property_type', 'operation_type', 'l2', 'l3'])['price'].transform('median'))
2 viviendas['currency'].fillna('USD', inplace = True)
3
4 viviendas = viviendas[viviendas['price'].notna()]

```

✓ Analisis univariado

Porcentaje de propiedades de cualquier tipo en cada provincia

■ Bloc con sangría

Para analizar cuantas propiedades totales hay en cada provincia (sin distinguir el tipo de propiedad), realizamos primero una tabla de frecuencia para evaluar el porcentaje

```

1 l2 = viviendas.stb.freq(['l2'], thresh = 95)
2 l2["l2"].replace({"others": "Otras provincias"}, inplace=True)
3 l2

```



		12	count	percent	cumulative_count	cumulative_percent
0	Capital Federal		22645	44.358472	22645	44.358472
1	Bs.As. G.B.A. Zona Norte		9798	19.192948	32443	63.551420
2	Santa Fe		4722	9.249755	37165	72.801175
3	Bs.As. G.B.A. Zona Sur		4470	8.756121	41635	81.557297
4	Bs.As. G.B.A. Zona Oeste		2987	5.851126	44622	87.408423
5	Córdoba		1870	3.663075	46492	91.071499
6	Buenos Aires Costa Atlántica		1304	2.554358	47796	93.625857
7	Otras provincias		3254	6.374143	51050	100.000000

Observamos que hay muchas provincias que representan un porcentaje total de viviendas menor a 1. Al graficar un pie-plot, me van a quedar muchas labels juntas y que no me dan real informacion. Por lo tanto, las provincias cuyo porcentaje de propiedades es mayor al 1% del total las agrupamos en la categoria 'Otras provincias'

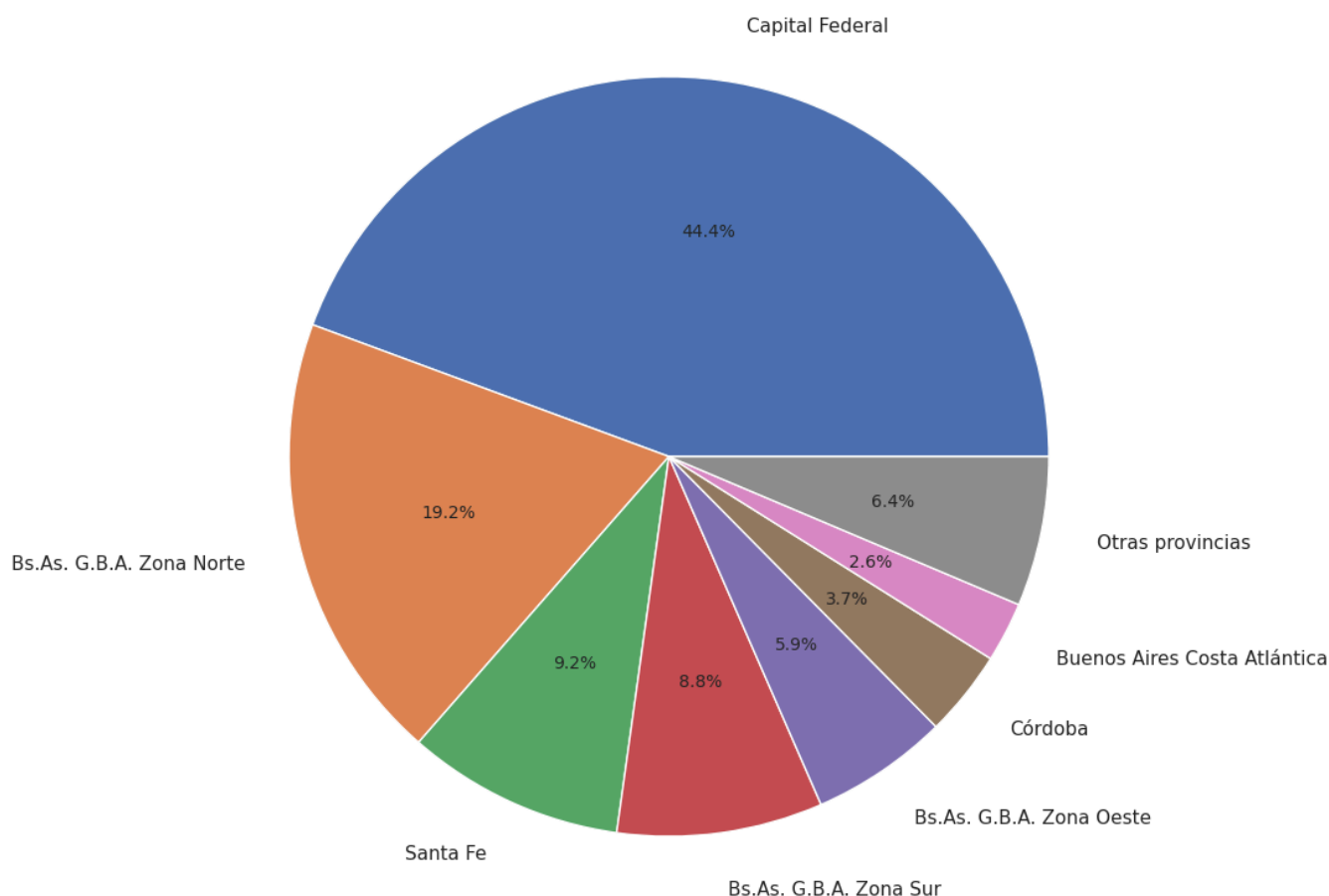
```

1
2 fig,ax =plt.subplots(figsize = (12,10))
3 ax.pie(l2['percent'], labels= l2['l2'], labeldistance=1.15, autopct='%1.1f%%', startangle=0)
4 ax.title.set_text('Porcentajes de viviendas segun provincia')

```



Porcentajes de viviendas segun provincia



Podemos concluir que el numero de propiedades por provincia no es balanceado, la mayor cantidad de propiedades se encuentran en la provincia de Buenos Aires, ya sea en Capital Federal o en GBA, seguido de Santa Fe y Córdoba. El resto de las provincias representa un porcentaje muy bajo de viviendas (la mayoría de ellas menor al 1% del total).

Creacion de mapas con geopandas

Para tener una distribucion mas visual de las propiedades, se genera un mapa de Argentina separado por provincias, y vemos como estan distribuidas las propiedades totales en el territorio

```
1 #Eliminamos los valores nulos de latitud y longitud y lo guardamos en un nuevo dataset
2 viviendasmap = viviendas.dropna(subset = ['Latitud', 'Longitud'])
3
4 #transformacion de latitud y longitud a float
5 viviendasmap['Latitud'] = viviendasmap['Latitud'].astype(float)
6
7 viviendasmap['Longitud'] = viviendasmap['Longitud'].astype(float)
8
9
```

```
1 import json
2 import os
3 from google.colab import drive
4 drive.mount('/content/drive')
5 os.chdir("/content/drive/MyDrive/Data science")
6 os.listdir()
7
8 with open('provincia.json') as json_file:
9     arg_prov= gpd.read_file(json_file)
10
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

↗ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
1 #Se convierten los datos de latitud y longitud del dataframe en puntos xy
2 gdf = gpd.GeoDataFrame(
3     viviendasmap, geometry=gpd.points_from_xy(viviendasmap.Longitud, viviendasmap.Latitud), crs='EPSG:4326')
4
5 print(gdf.head())
6
7 #Vemos que estos datos quedan contenidos en la columna 'geometry'
```

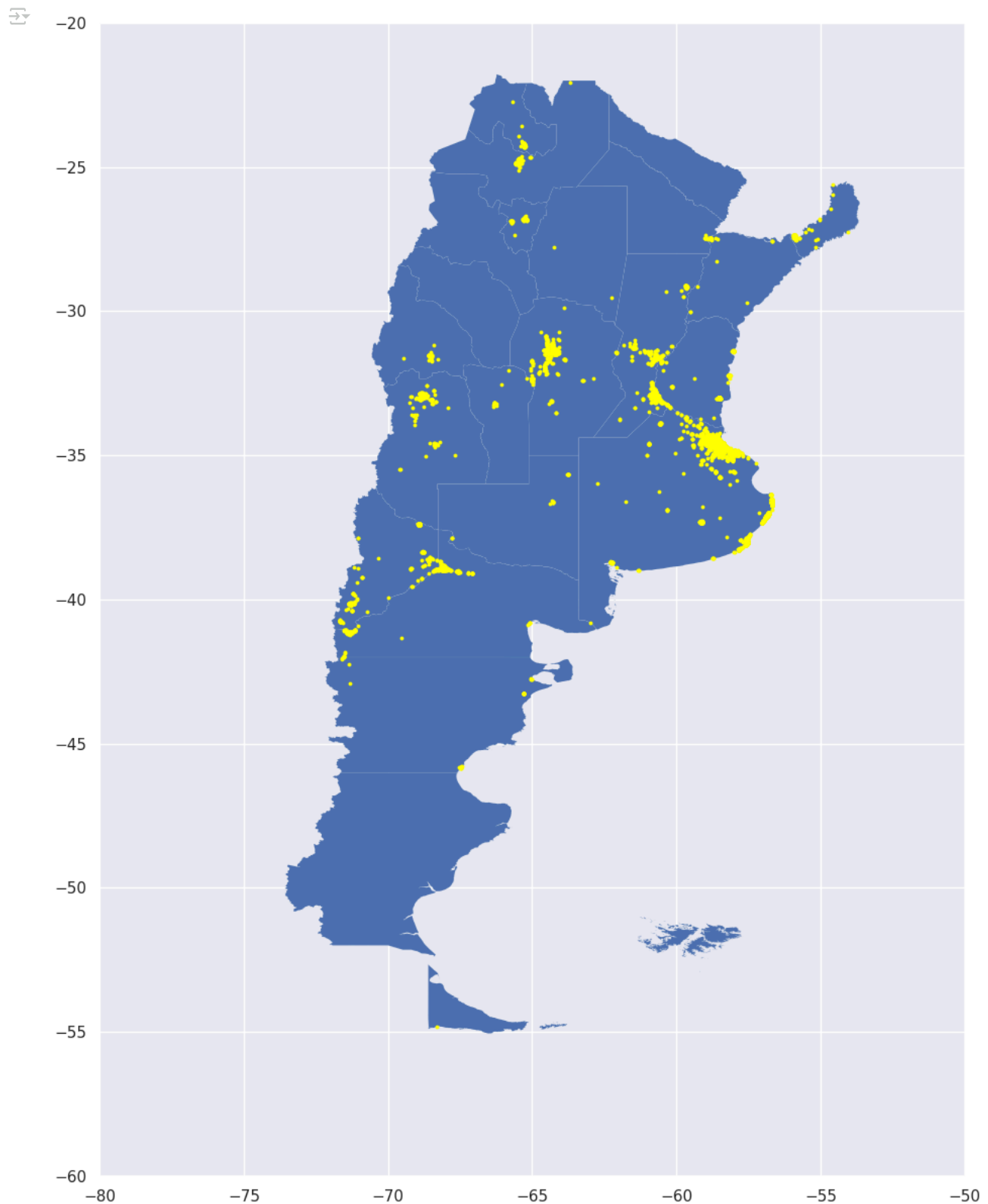
```
↗
  start_date  end_date  Latitud  Longitud  l1 \
0 2020-08-22 2020-09-04 -37.996039 -57.542509 Argentina
1 2020-08-22 2020-08-31 -31.380200 -58.009200 Argentina
2 2020-08-22 2020-09-04 -32.948900 -60.630500 Argentina
3 2020-08-22 2020-09-04 -32.884278 -60.710901 Argentina
4 2020-08-22 2020-09-04 -32.948800 -60.630200 Argentina

      l2      l3  rooms  bathrooms \
0 Buenos Aires Costa Atlántica Mar del Plata  8.0      NaN
1      Entre Ríos Concordia  2.0      1.0
2      Santa Fe Rosario  3.0      1.0
3      Santa Fe Rosario  2.0      1.0
4      Santa Fe Rosario  5.0      2.0

  surface_total  surface_covered  price  currency  property_type \
0          687          687  320000.000000  USD      Otro
1           80           80   16.256158  USD      Casa
2           76           66  74000.000000  USD  Departamento
3           39           33   13.793103  USD  Departamento
4          130          108  74000.000000  USD  Departamento

  operation_type  geometry
0      Venta POINT (-57.54251 -37.99604)
1    Alquiler POINT (-58.00920 -31.38020)
2      Venta POINT (-60.63050 -32.94890)
3    Alquiler POINT (-60.71090 -32.88428)
4      Venta POINT (-60.63020 -32.94880)
```

```
1 #Sobreponemos los puntos de las propiedades a la capa del mapa separado por provincias
2
3 mapa = arg_prov.plot(linewidth=0.03, figsize=(15,12))
4 gdf['geometry'].plot(ax=mapa, color='yellow', markersize=3.2, aspect=1)
5 plt.tight_layout()
6 plt.xlim([-80, -50])
7 plt.ylim([-60, -20]);
8
```



Vemos que, coincidiendo con el pieplot, se observa la mayor cantidad de propiedades en la provincia de Buenos Aires, pero no homogéneamente. También vemos que en Santa Fe, la siguiente provincia con mayor número de propiedades, los datos se concentran principalmente en el sur y centro, mientras que en Córdoba se concentran en el noroeste de la provincia.

Porcentaje de tipo de propiedad y de tipo de operacion

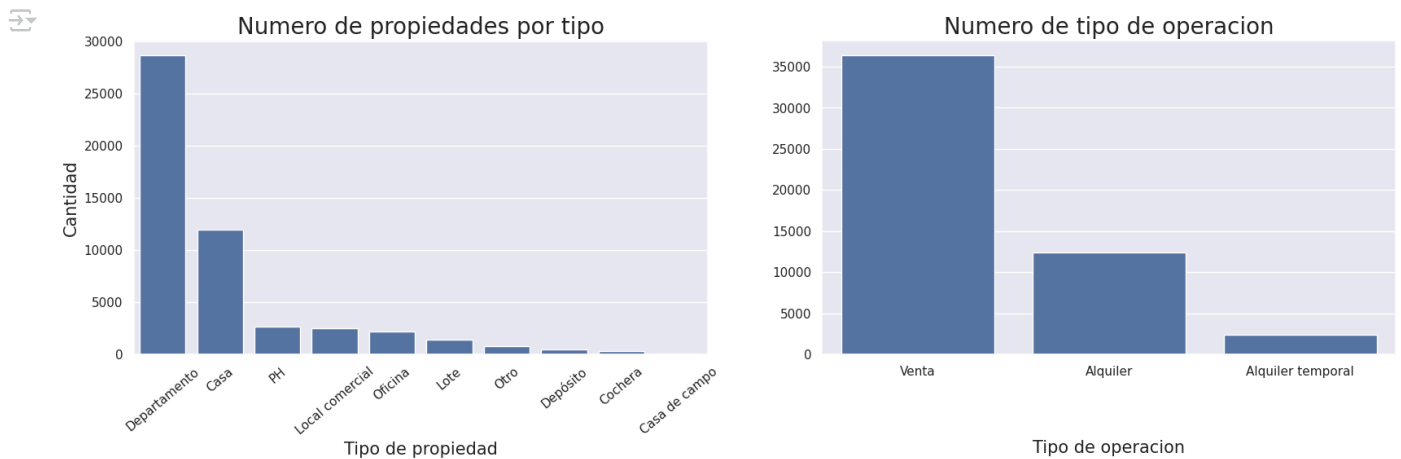
Nos interesa también conocer el número de tipo de propiedades, así como también el tipo de operación. Para ello realizamos un countplot que contenga estas variables por separado.

```
1 # Seteamos el estilo del plot
2 sns.set(style="darkgrid")
3 sns.set_theme(rc={"font.size":10,"axes.titlesize":20,"axes.labelsize":15})
4
```

```

5 # Creamos el lienzo
6 fig, axs = plt.subplots(ncols =2, figsize=(20,5))
7
8 # Creamos un countplot del tipo de propiedades
9 sns.countplot(x="property_type",
10               data=viviendas,
11               ax=axs[0],
12               order = viviendas['property_type'].value_counts().index,
13               ).set(title='Numero de propiedades por tipo',xlabel = 'Tipo de propiedad', ylabel = 'Cantidad');
14 axs[0].tick_params(axis='x', rotation=40)
15
16
17
18 # Creamos un countplot del tipo de operacion
19 sns.countplot(x="operation_type",
20               data=viviendas,
21               ax=axs[1],
22               order = viviendas['operation_type'].value_counts().index,
23               ).set(title='Numero de tipo de operacion',xlabel = 'Tipo de operacion', ylabel = '');
24 axs[1].set_xlabel(xlabel = 'Tipo de operacion', labelpad=55);
25

```



Observamos que la gran mayoría de propiedades disponibles son departamentos, seguido de casas. Tenemos también otro tipo de propiedades pero su número es mucho menor en comparación. A su vez, vemos que las operaciones de venta superan con creces a las de alquileres y a las de alquileres temporales.

```

1 #Calculamos algunas estadísticas para facilitar el análisis
2 q1 = viviendas.quantile(0.25)
3 q3= viviendas.quantile(0.75)
4 IQR = q3-q1
5 mediana = viviendas.median()
6
7 print('Q1:\n{}\n Q3\n{}\n IQR:\n{}\n Mediana:\n{}\n'.format(q1,q3,IQR, mediana))
8

```

```

Q1:
Latitud      -34.623598
Longitud     -58.720857
rooms        2.000000
bathrooms    1.000000
surface_total 49.000000
surface_covered 45.000000
price        1500.000000
Name: 0.25, dtype: float64
Q3:
Latitud      -34.461288
Longitud     -58.410654
rooms        4.000000
bathrooms    2.000000

```

```

surface_total      200.000000
surface_covered    154.000000
price              190000.000000
Name: 0.75, dtype: float64
IQR:
Latitud            0.162310
Longitud           0.310203
rooms              2.000000
bathrooms          1.000000
surface_total      151.000000
surface_covered    109.000000
price              188500.000000
dtype: float64
Mediana:
Latitud            -34.587973
Longitud           -58.469059
rooms              3.000000
bathrooms          1.000000
surface_total      83.000000
surface_covered    72.000000
price              90000.000000
dtype: float64

```

Observamos en el boxplot que para el caso del numero de habitaciones, poseemos una distribucion simetrica ya que la mediana se encuentra en el centro de la caja, siendo $q_1 = 2$ y $q_3 = 4$ habitaciones, siendo sus limites 1 y 7 aproximadamente. Tambien puede evidenciarse la presencia de varios valores atipicos, mas altos que el limite superior del boxplot.

En el caso del numero de baños, $q_1=1$ y $q_3=2$. La mediana en este caso coincide con q_1 , y la distribucion esta sesgada positivamente (hacia los valores mas chicos). Observamos a su vez varios outliers mas altos que el limite superior del boxplot

```

1 filtro lote = viviendas[viviendas['property_type'] == 'Lote'][['bathrooms', 'rooms']]
2 filtro cochera = viviendas[viviendas['property_type'] == 'Cochera'][['bathrooms', 'rooms']]
3
4 print('Baños lotes nulos')
5 print(filtro lote.isnull().sum()/(filtro lote.shape[0]))
6 print('\n Baños cocheras nulos')
7 print(filtro cochera.isnull().sum()/(filtro cochera.shape[0]))

```

```

↗ Baños lotes nulos
bathrooms    0.854793
rooms        0.891989
dtype: float64

Baños cocheras nulos
bathrooms    0.957655
rooms        0.951140
dtype: float64

```

Vemos que algunos tipos de propiedades como lotes y cocheras, tienen casi todos los datos de baños y ambientes como nulos, lo que tiene sentido.

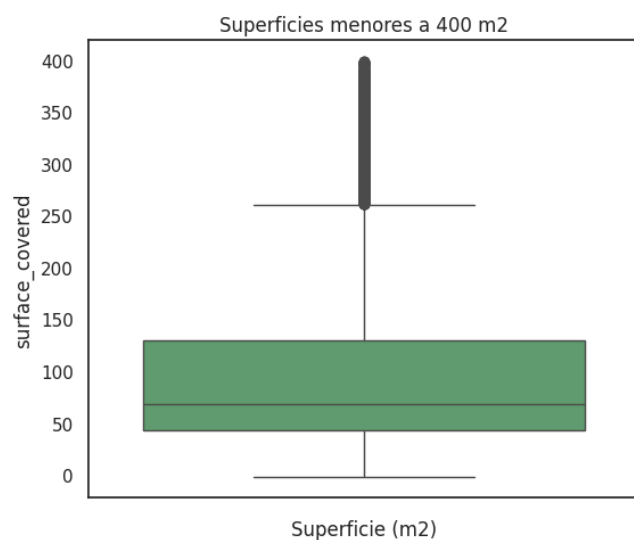
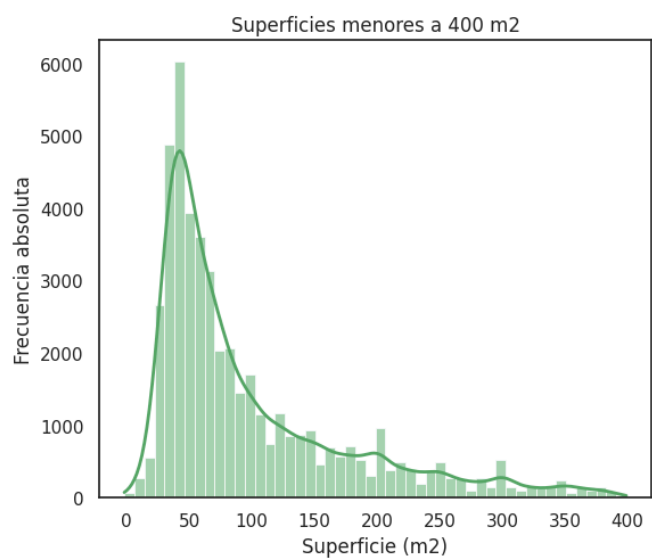
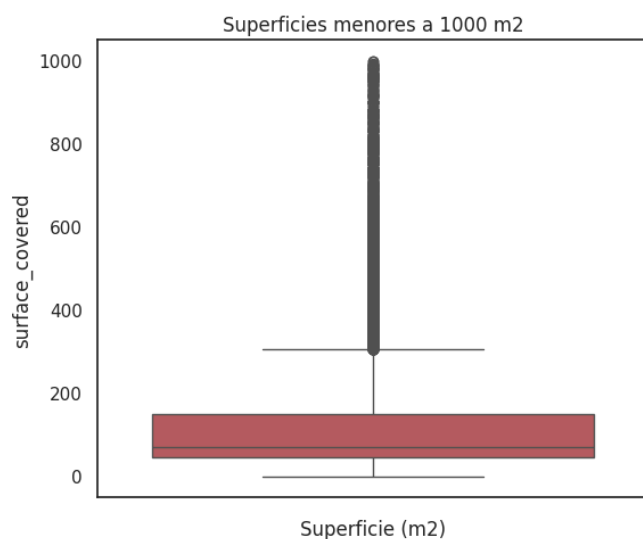
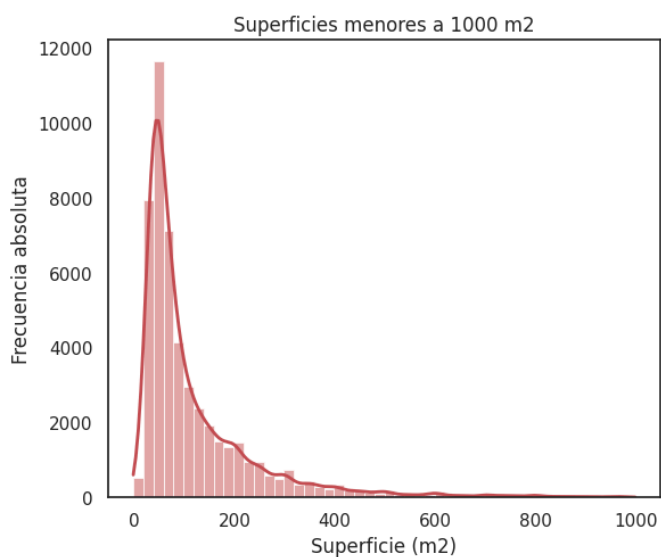
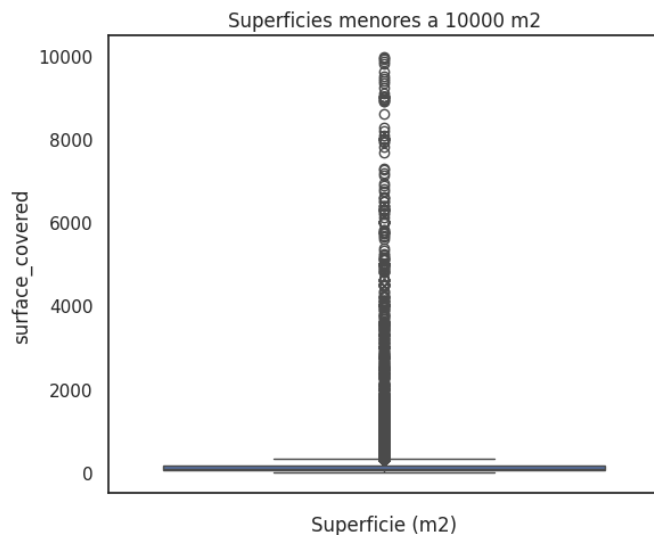
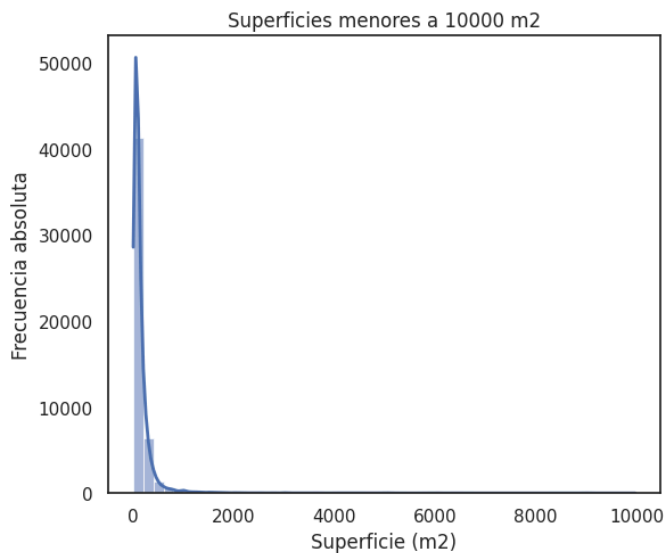
Numero de propiedades segun superficie

```

1 #Grafico histograma
2
3 sns.set(style="white", rc={"lines.linewidth": 2})
4 fig, ax = plt.subplots(3,2, figsize=(12,15))
5
6 sns.histplot(x='surface_covered',
7             data= viviendas[viviendas['surface_covered']<10000] ,
8             color='b',
9             ax=ax[0][0],
10            bins = 50,
11            kde=True).set(title='Superficies menores a 10000 m2', xlabel= 'Superficie (m2)', ylabel= 'Frecuencia absoluta')
12
13 sns.histplot(x='surface_covered',
14            data= viviendas[viviendas['surface_covered']<1000] ,
15            color='r',
16            ax=ax[1][0],
17            bins = 50,
18            kde=True).set(title='Superficies menores a 1000 m2', xlabel= 'Superficie (m2)', ylabel= 'Frecuencia absoluta')
19
20 sns.histplot(x='surface_covered',
21            data= viviendas[viviendas['surface_covered']<400] ,
22            color='g',
23            ax=ax[2][0],
24            bins = 50,

```

```
25
26         kde=True).set(title='Superficies menores a 400 m2', xlabel= 'Superficie (m2)', ylabel= 'Frecuencia absoluta')
27
28 plt.subplots_adjust(hspace = 0.3)
29
30 #Grafico boxplot
31
32 filtro_sup1 = viviendas[viviendas['surface_covered']<10000]['surface_covered']
33 filtro_sup2 = viviendas[viviendas['surface_covered']<1000]['surface_covered']
34 filtro_sup3 = viviendas[viviendas['surface_covered']<400]['surface_covered']
35
36 sns.boxplot(filtro_sup1 , ax = ax[0][1], color = 'b').set(title='Superficies menores a 10000 m2', xlabel='Superficie (m2)')
37 sns.boxplot(filtro_sup2 , ax = ax[1][1], color = 'r').set(title='Superficies menores a 1000 m2', xlabel='Superficie (m2)')
38 sns.boxplot(filtro_sup3 , ax = ax[2][1],color = 'g').set(title='Superficies menores a 400 m2', xlabel='Superficie (m2)')
39
40 fig.tight_layout()
41
```



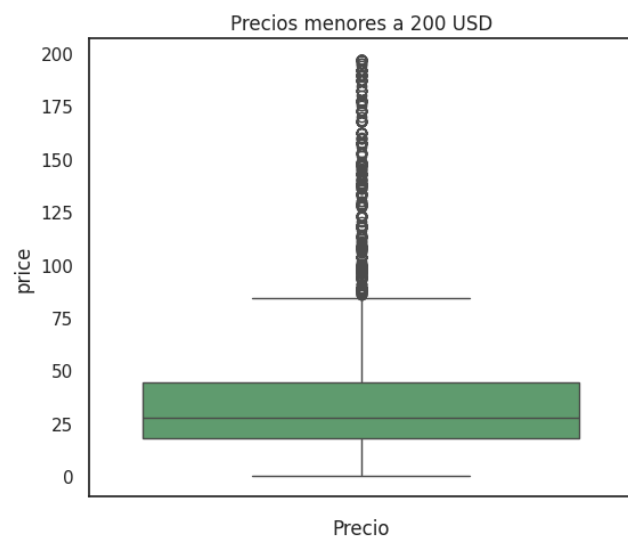
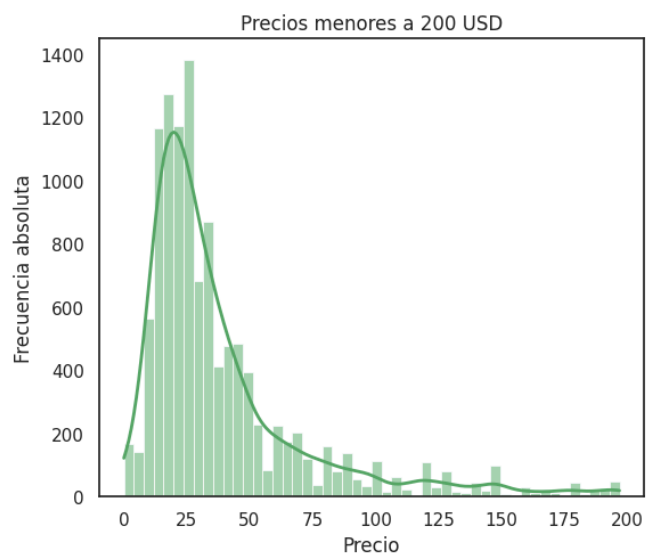
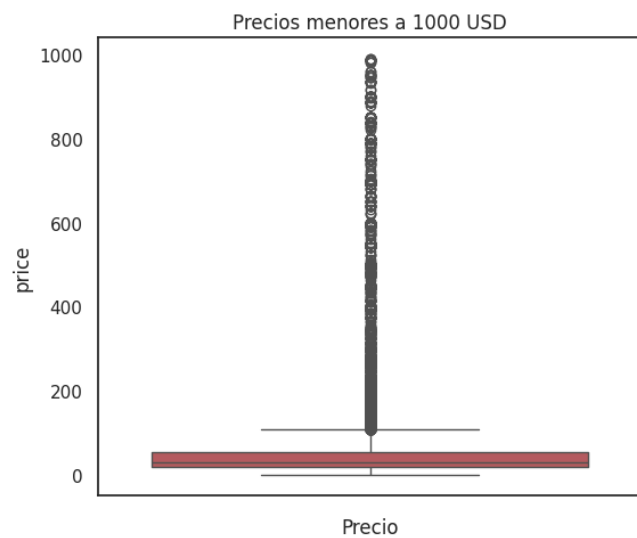
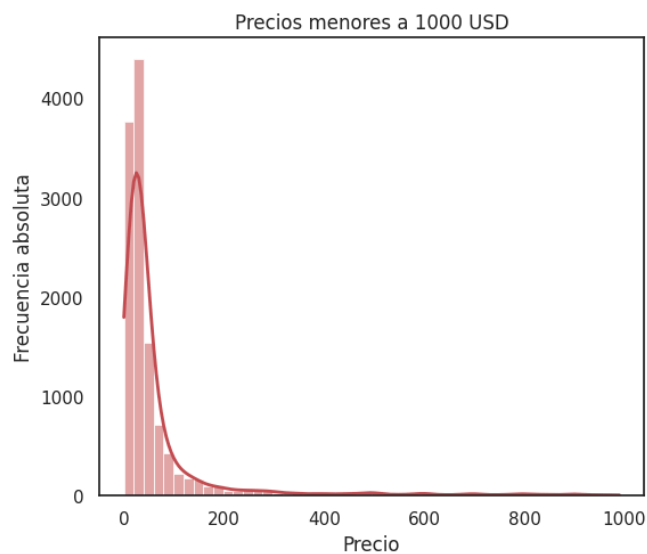
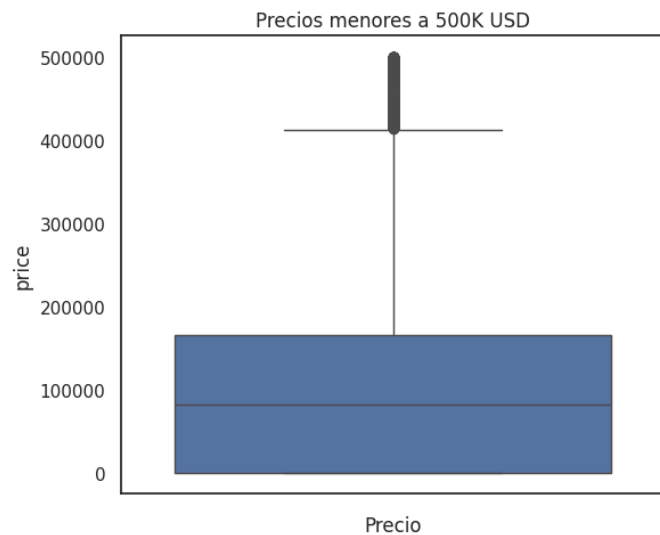
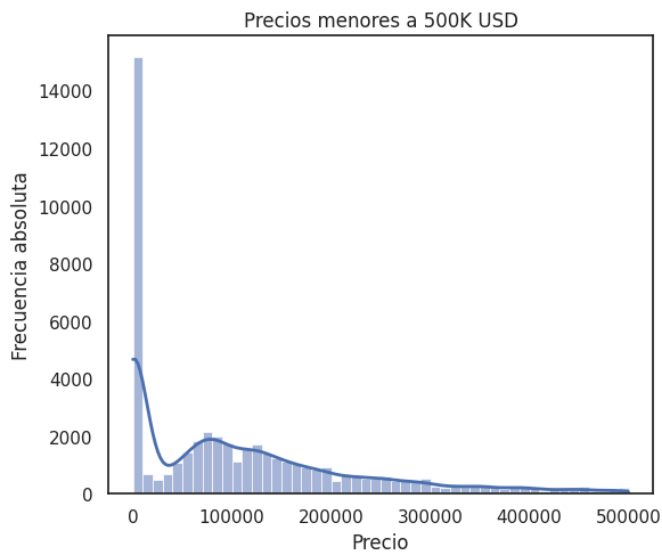
Observamos que la superficie edificada presenta una distribucion completamente desplazada hacia la izquierda, con muchisimos outliers. Cuando filtramos y graficamos valores mas chicos, observamos como la distribucion se va acercando a una distribucion normal. Esto sucede porque la mayoría de los datos estan concentrados en valores chicos, pero igualmente hay una elevada cantidad de datos con valores muy altos, que en los boxplot se visualizan como outliers

[texto del vínculo](#)Numero de propiedades segun precio

```

1 #Grafico histograma
2
3 sns.set(style="white", rc={"lines.linewidth": 2})
4 fig, ax = plt.subplots(3,2, figsize=(12,15))
5
6 sns.histplot(x='price',
7             data= viviendas[viviendas['price']<500000] ,
8             color='b',
9             ax=ax[0][0],
10            bins = 50,
11            kde=True).set(title='Precios menores a 500K USD', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
12
13 sns.histplot(x='price',
14            data= viviendas[viviendas['price']<1000] ,
15            color='r',
16            ax=ax[1][0],
17            bins = 50,
18            kde=True).set(title='Precios menores a 1000 USD', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
19
20 sns.histplot(x='price',
21            data= viviendas[viviendas['price']<200] ,
22            color='g',
23            ax=ax[2][0],
24            bins = 50,
25
26            kde=True).set(title='Precios menores a 200 USD', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
27
28 plt.subplots_adjust(hspace = 0.3)
29
30 #Grafico boxplot
31
32 filtro_precio1 = viviendas[viviendas['price']<500000]['price']
33 filtro_precio2 = viviendas[viviendas['price']<1000]['price']
34 filtro_precio3 = viviendas[viviendas['price']<200]['price']
35
36 sns.boxplot(filtro_precio1 , ax = ax[0][1], color = 'b').set(title='Precios menores a 500K USD', xlabel='Precio') #orient='
37 sns.boxplot(filtro_precio2 , ax = ax[1][1], color = 'r').set(title='Precios menores a 1000 USD', xlabel='Precio') #orient="
38 sns.boxplot(filtro_precio3 , ax = ax[2][1],color = 'g').set(title='Precios menores a 200 USD', xlabel='Precio') #orient="v"
39
40 fig.tight_layout()
41

```



Cuando realizamos el histograma, vemos que hay muchísimos valores pequeños, el gráfico está sesgado positivamente. Por lo tanto, vemos que tenemos una frecuencia altísima entre los precios más bajos. Para poder ver mejor la distribución, se filtra la variable precios para valores menores a 1000 y se vuelve a graficar el histograma debajo del anterior.

Vemos a su vez con los boxplot que cuando graficamos los precios menores a 500K USD o menores a 1000 USD, se observan una gran cantidad de outliers, mientras que los valores menores a 200USD no presentan outliers y tienen una distribución cercana a la normal.

Conclusion univariado

Luego de todo el análisis univariado, se puede concluir que al graficar todos los datos sin agrupar, la mayor cantidad de los registros está concentrada en valores pequeños de las variables. Sin embargo, hay algunos valores elevados, detectados como outliers pero según los modelos de predicción que utilizemos pueden ser de utilidad (los departamentos grandes en venta seguramente tengan alto precio que es detectado como outlier ya que tenemos muy baja cantidad en comparación con otros datos con valores más pequeños). Por lo tanto, conservaremos estos valores hasta el momento de realizar los modelos de Machine Learning.

✓ Analisis bivariado

Procedemos ahora a realizar un análisis bivariado de las variables. Comenzamos realizando una matriz de correlación, para visualizar cómo se comporta una variable respecto a la otra. Esta correlación utiliza Pearson como método estándar.

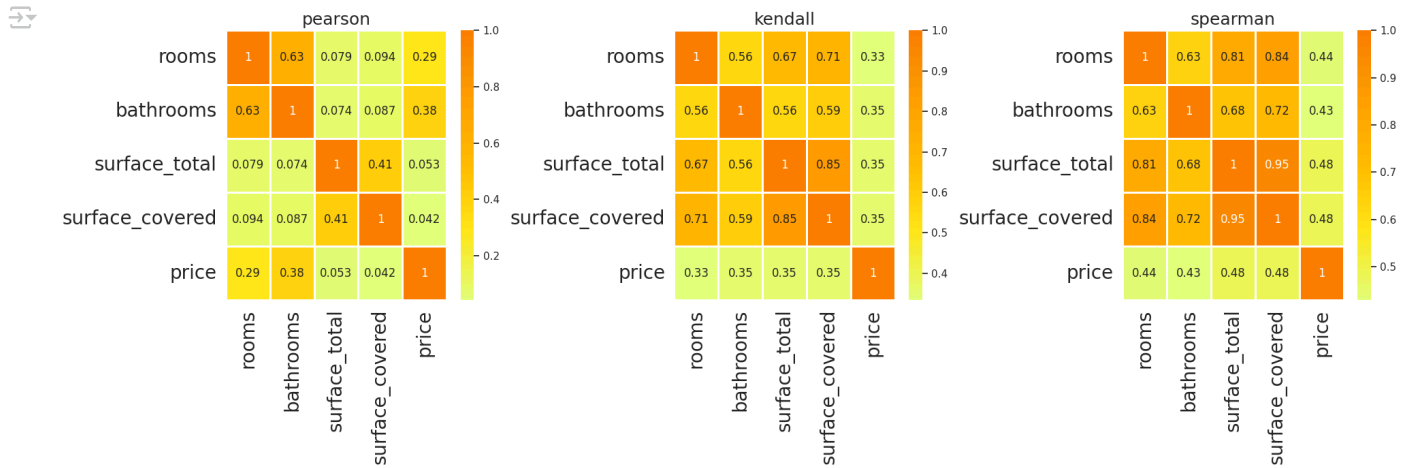
```
1 matrix_correlation = viviendas.corr()
2 matrix_correlation.head()
```



	Latitud	Longitud	rooms	bathrooms	surface_total	surface_covered	price
Latitud	1.000000	-7.005862e-02	0.057578	-5.774342e-04	0.016972	0.017872	-0.019531
Longitud	-0.070059	1.000000e+00	-0.111220	-4.096904e-07	-0.060430	-0.033679	0.050141
rooms	0.057578	-1.112203e-01	1.000000	6.329788e-01	0.084395	0.097876	0.300579
bathrooms	-0.000577	-4.096904e-07	0.632979	1.000000e+00	0.103439	0.126610	0.366020
surface_total	0.016972	-6.042952e-02	0.084395	1.034391e-01	1.000000	0.669761	0.093999

Traducimos esta tabla en un gráfico de correlación, utilizando 3 métodos de correlación: Pearson, Kendall y Spearman.

```
1 #generacion de los graficos de correlacion entre los 3 metodos,
2
3
4 plt.figure(figsize=(20,7))
5 for j,i in enumerate(['pearson','kendall','spearman']):
6     plt.subplot(1,3,j+1)
7     correlation = viviendas.dropna().drop(['Latitud', 'Longitud'], axis = 1).corr(method=i)
8     sns.heatmap(correlation, linewidth = 2, annot=True, cmap="Wistia")
9     plt.title(i, fontsize=18)
10    plt.yticks(fontsize=20)
11    plt.xticks(fontsize=20)
12    plt.tight_layout();
```



Como observamos anteriormente que la distribución de las variables no es normal, y contamos con un dataset grande, realizamos el análisis teniendo en cuenta el método de Spearman. Vemos una correlación positiva entre rooms y bathrooms (que indica que el número de ambientes es proporcional al número de baños en la vivienda), así como también entre superficie cubierta y la superficie total (la porción edificada de un terreno es proporcional a la superficie total del terreno). Lo mismo sucede con la superficie y el número de baños o de ambientes (a mayor superficie disponible, mayor cantidad de baños y ambientes).

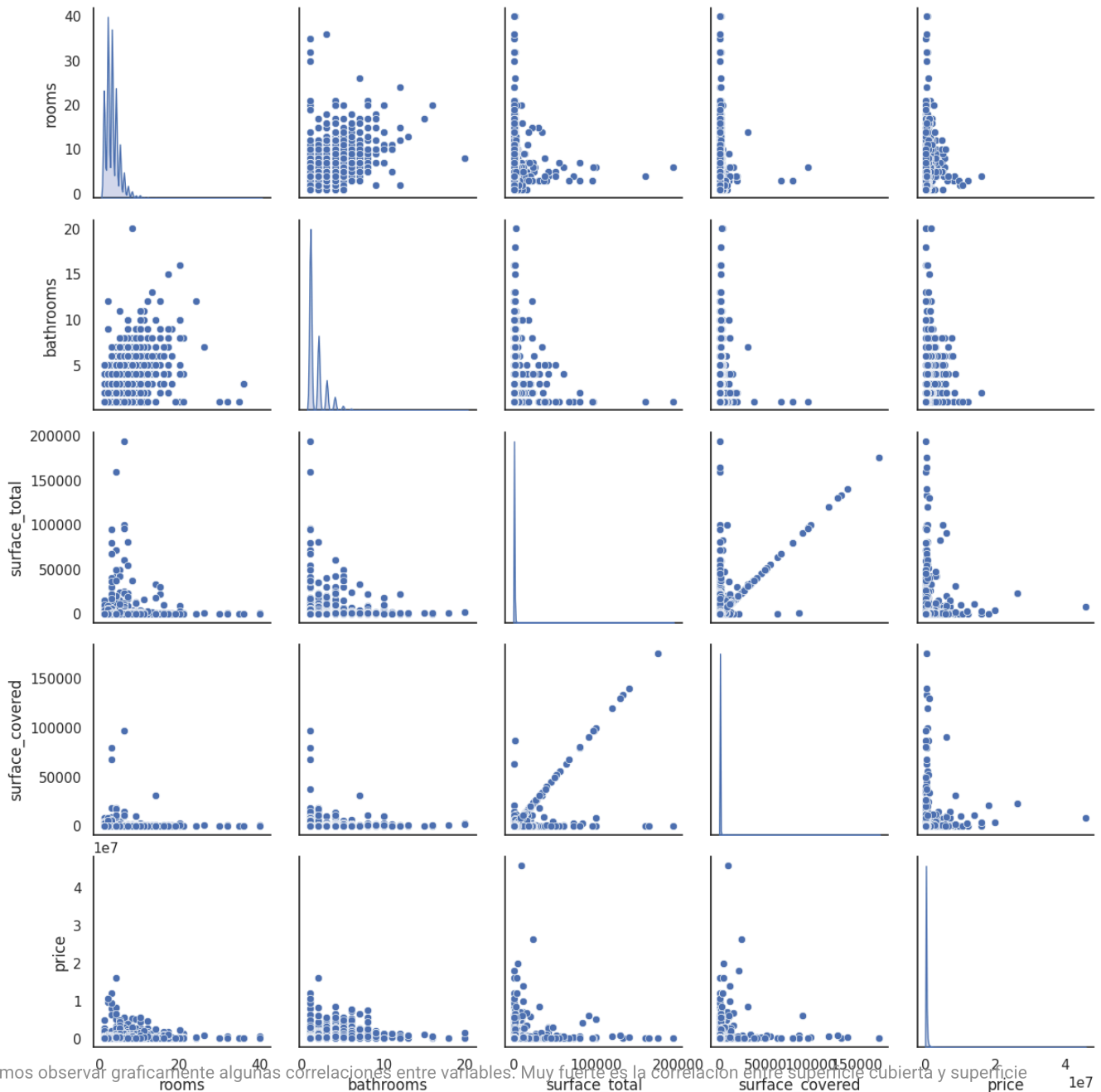
Observamos también que la correlación entre el precio y el resto de las variables es baja. Esto puede deberse a que el precio no se explique correctamente por ninguna de estas variables por separado, o que el precio que observamos aquí es el total (es decir, no está separado por provincias, por tipo de vivienda ni por tipo de operación), por lo que la correlación no está representando exactamente las distintas variables. Sería correcto realizar distintos agrupamientos antes de ver cómo se determina la variable precio según las otras variables (en un análisis multivariado).

Para tener una primera visualización gráfica, realizamos un pairplot con las mismas variables numéricas mostradas en la matriz de correlación.

```
1 plt.figure(figsize=(14, 14))
2
3 g = sns.pairplot(viviendas[['rooms', 'bathrooms', 'surface_total', 'surface_covered', 'price']], diag_kind='kde');
4 g.fig.suptitle("Grafico de puntos en variables numericas", fontsize=20, y=1.02);
```

<Figure size 1400x1400 with 0 Axes>

Gráfico de puntos en variables numericas

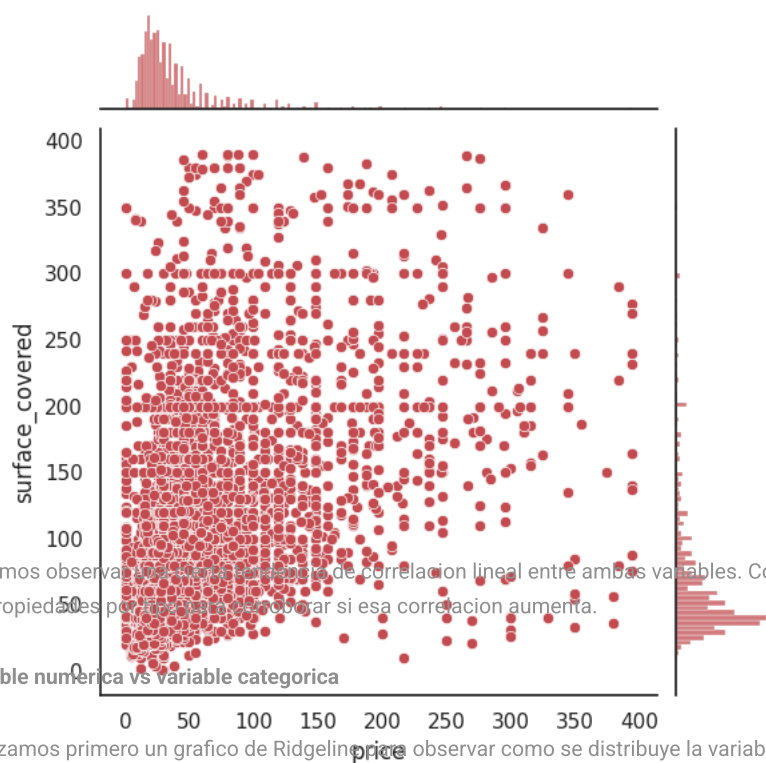
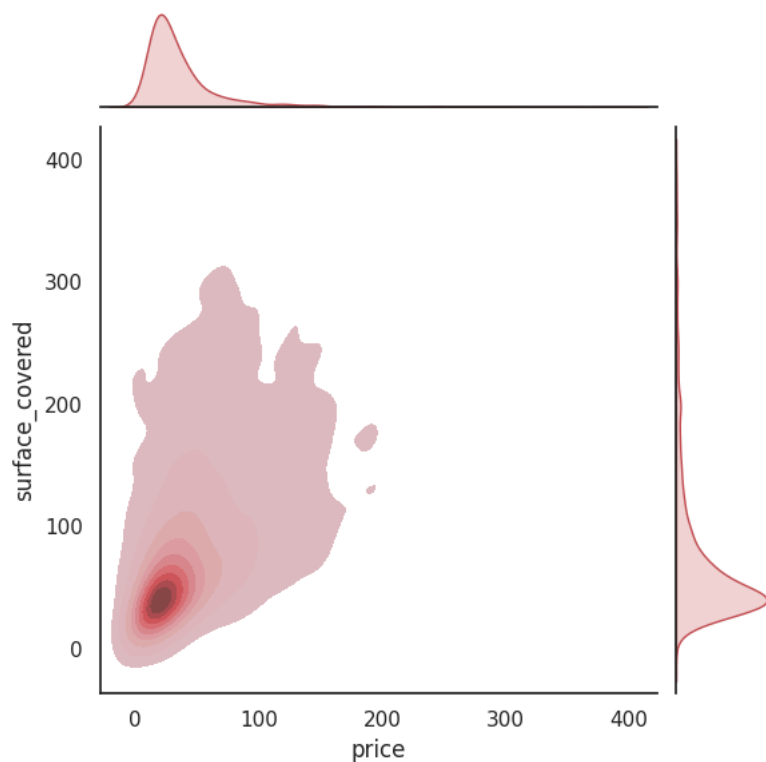


Podemos observar gráficamente algunas correlaciones entre variables. Muy fuerte es la correlación entre superficie cubierta y superficie total, cosa que tiene sentido, ya que a mayor superficie de terreno mayor será la superficie donde se edifique. También hay una correlación positiva entre el número de ambientes y el número de baños.

Variable numerica vs variable numerica

Como se mencionó, todas las variables están muy desplazadas hacia los valores pequeños, donde están concentrados los datos. Sin embargo, existe alto número de valores altos que impide realizar gráficos confiables entre variables numéricas. Debido a esta distribución, para realizar este tipo de gráficos (al menos utilizando solo dos variables) es necesario filtrar las variables y quedarnos solo con los valores más pequeños. Guiándonos por los boxplot y los diagramas de distribución realizados en el análisis univariado, decidimos utilizar solo los precios menores a 400USD y las superficies edificadas menores a 400 m².

```
1 #Aplicamos el filtro
2 viviendas_sup_prec = viviendas[(viviendas['surface_covered']<400) & (viviendas['price']<400) & (viviendas['surface_total']
3
4 sns.jointplot(x = 'price', y = 'surface_covered', data=viviendas_sup_prec, color = 'r', kind = 'kde', fill= True)
5 sns.jointplot(x = 'price', y = 'surface_covered', data=viviendas_sup_prec, ax=axis[0], color = 'r');
6
7 sns.jointplot(x = 'price', y = 'surface_total', data=viviendas_sup_prec, ax=axis[0], color = 'g', kind = 'kde', fill= True);
8 sns.jointplot(x = 'price', y = 'surface_total', data=viviendas_sup_prec, ax=axis[0], color = 'g');
9
10
```



Podemos observar una débil tendencia de correlación lineal entre ambas variables. Como se mencionó anteriormente, es necesario separar las propiedades por tipo de operación para observar si esa correlación aumenta.

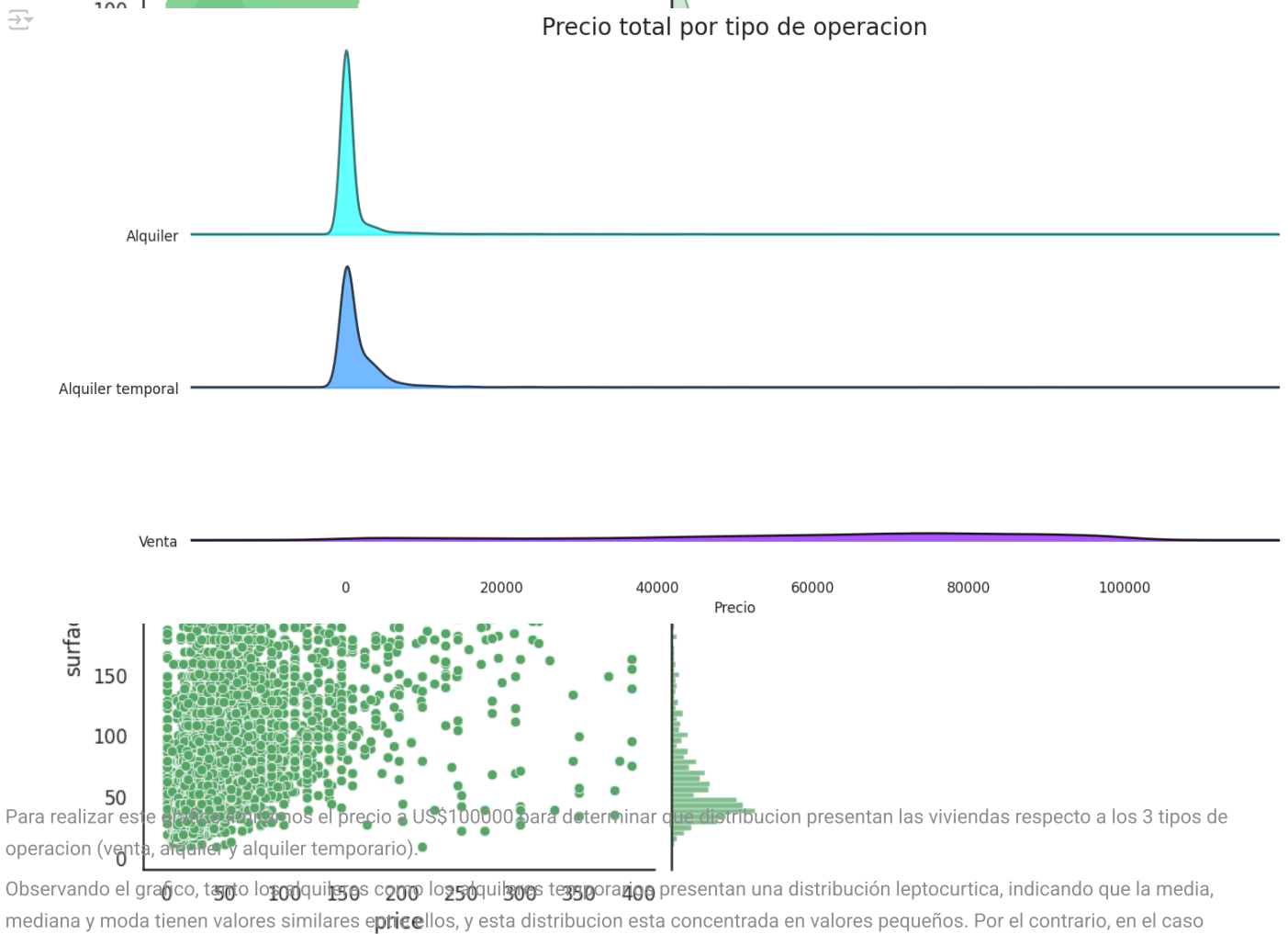
Variable numérica vs variable categórica

Realizamos primero un gráfico de Ridgeline para observar como se distribuye la variable precio según el tipo de operación (alquiler, alquiler temporal y venta)



```
1 #Precio vs tipo de operacion
2
3 joyplot( viviendas[viviendas['price']<100000], by = 'operation_type', column = 'price', fade = True, figsize=(15,7), color=
4 plt.title('Precio total por tipo de operacion', fontsize=20)
5 plt.xlabel("Precio")
6 plt.ylabel("Tipo de operacion")
7 plt.show();
8
```





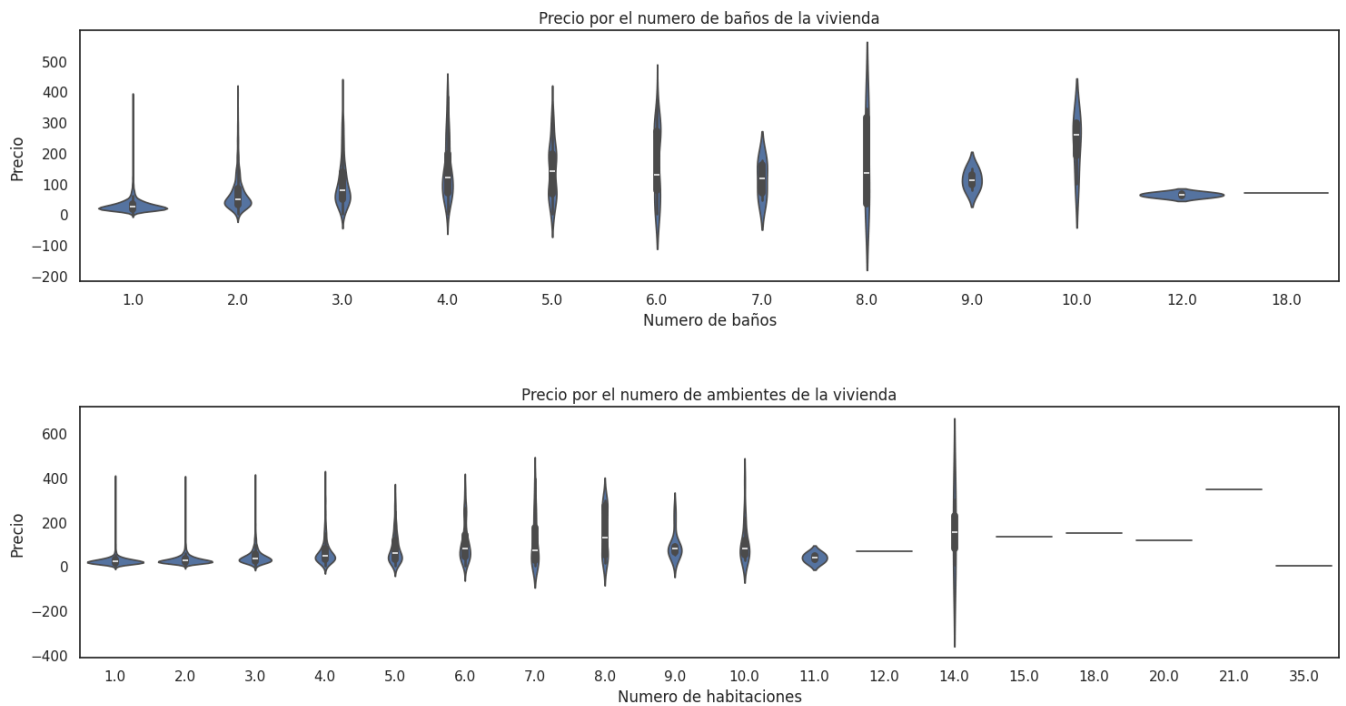
Observando el gráfico, tanto los alquileres como los alquileres temporales presentan una distribución leptocurtica, indicando que la media, mediana y moda tienen valores similares e pequeños, y esta distribución esta concentrada en valores pequeños. Por el contrario, en el caso de venta vemos que el precio esta distribuido a lo largo de muchos valores distintos, mayormente valores altos.

Graficamos luego el precio total según el numero de baños y de ambientes de la propiedad.

```

1 # viviendas_sup_prec = viviendas[(viviendas['surface_covered']<400) & (viviendas['price']<400) & (viviendas['surface_total
2 # filtro_precio3 = viviendas[viviendas['price']<400]
3 fig,ax = plt.subplots(nrows = 2, figsize=(15,8))
4
5 #Grafico numero de baños vs precio
6 sns.violinplot(x = 'bathrooms',
7               y='price',
8               data= viviendas[viviendas['price']<400],
9               ax=ax[0] ).set(title = 'Precio por el numero de baños de la vivienda', xlabel = 'Numero de baños', ylabel= 'P
10
11 #Grafico numero de ambientes vs precio
12 sns.violinplot(x = 'rooms',
13               y='price',
14               data= viviendas[viviendas['price']<400],
15               ax=ax[1] ).set(title = 'Precio por el numero de ambientes de la vivienda', xlabel = 'Numero de habitaciones'.
16 plt.tight_layout()
17 plt.subplots_adjust(hspace = 0.5)
18 ;

```



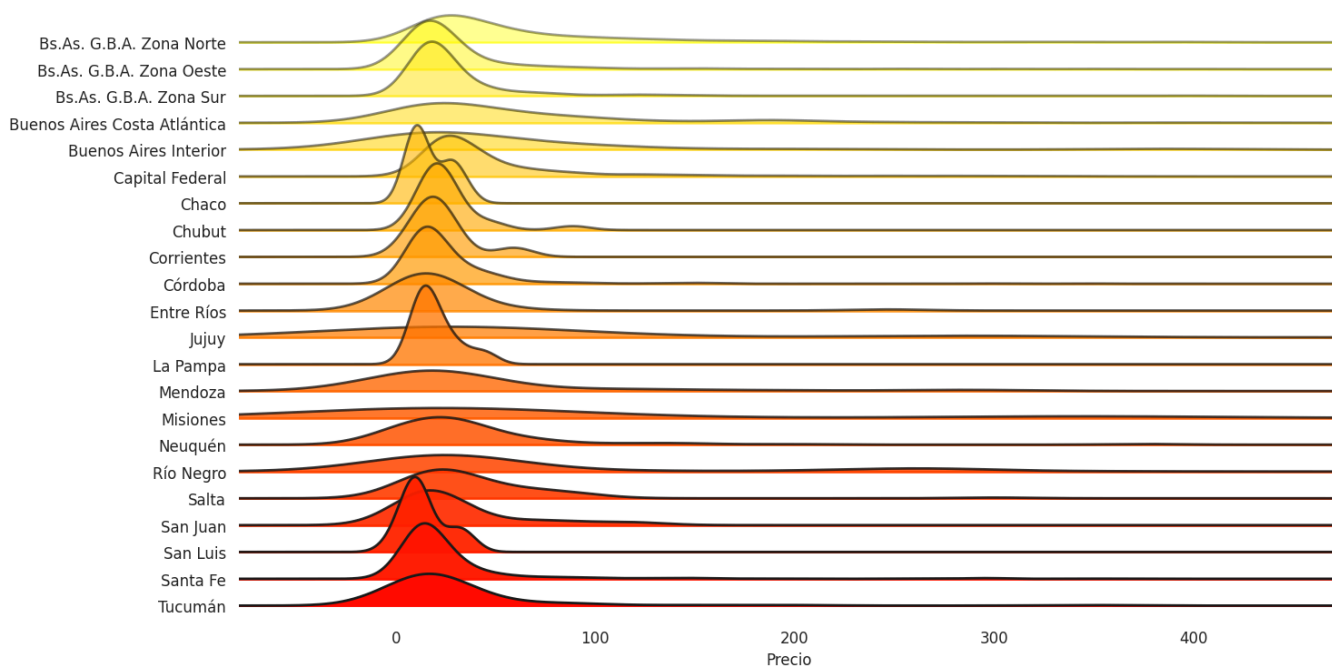
Cuando graficamos el precio versus el numero de baños o de ambientes, vemos que la mayor cantidad de datos se concentran alrededor de la mediana. Vemos tambien que los valores mas altos en el eje x no presentan una distribucion de precios en el eje y. Esto significa que son valores unicos, aislados, a los cuales corresponde un solo precio, o un numero muy acotado de los mismos.

Observamos tambien un ligero aumento del precio al aumentar el numero de habitaciones/baños, pero esta correlacion tampoco es tan notoria. En sintonía con lo visto arriba, es necesario agrupar y trabajar con las variables antes de observar como se comporta la variable precio.

```
1 #Precio promedio vs ubicacion por provincia
2
3 joyplot(viviendas[viviendas['price']<400], by = 'l2', column = 'price', fade = True, figsize=(15,8), colormap=cm.autumn_r)
4 plt.title('Precio total por provincia', fontsize=20)
5 plt.xlabel("Precio")
6 plt.ylabel("Provincia")
7 plt.show();
```




Precio total por provincia

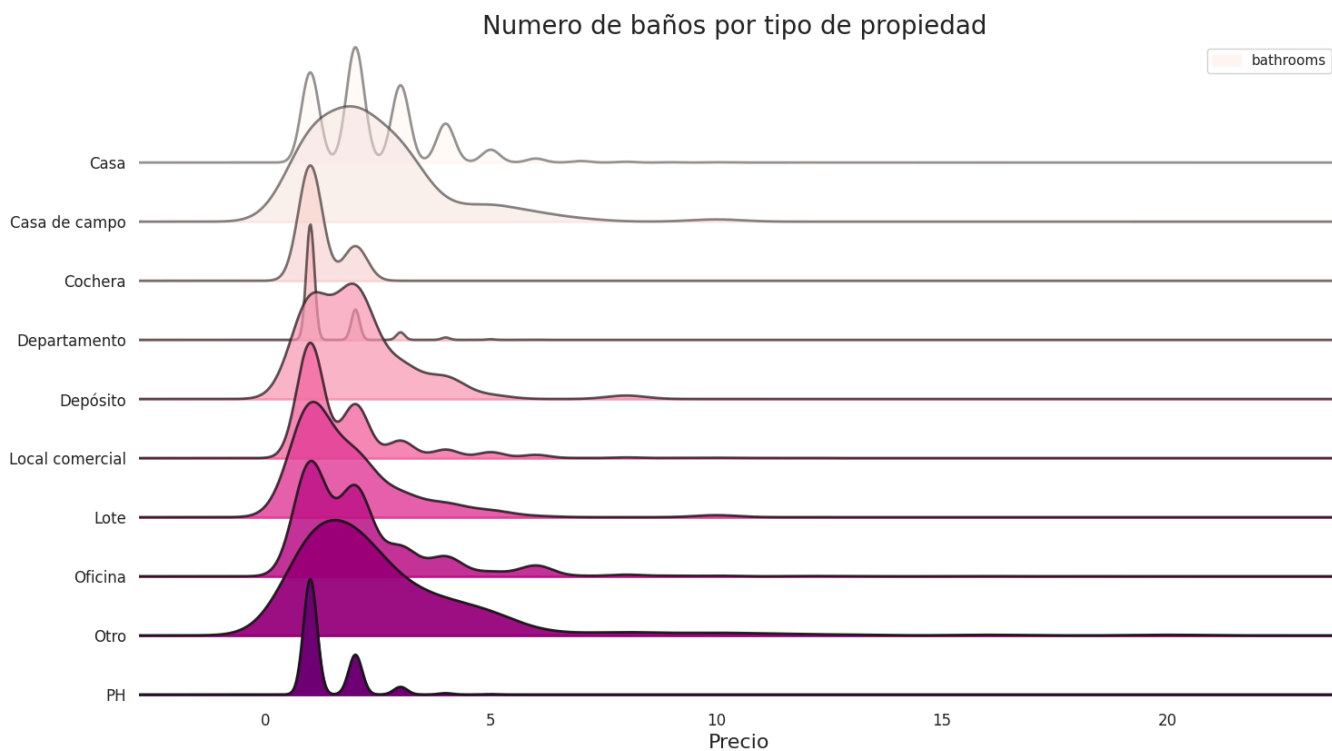


Así agrupados no se evidencia una real diferencia entre los precios totales de las viviendas en las distintas provincias, distribuyéndose los valores mayormente entre 0 y 200 USD. Como vimos antes, estos valores muy probablemente correspondan a alquileres, por lo que se debiera separar alquileres de ventas y volver a evaluar la distribución de precios.

Evaluamos ahora como se distribuyen los baños y los ambientes según el tipo de propiedad en consideración

```

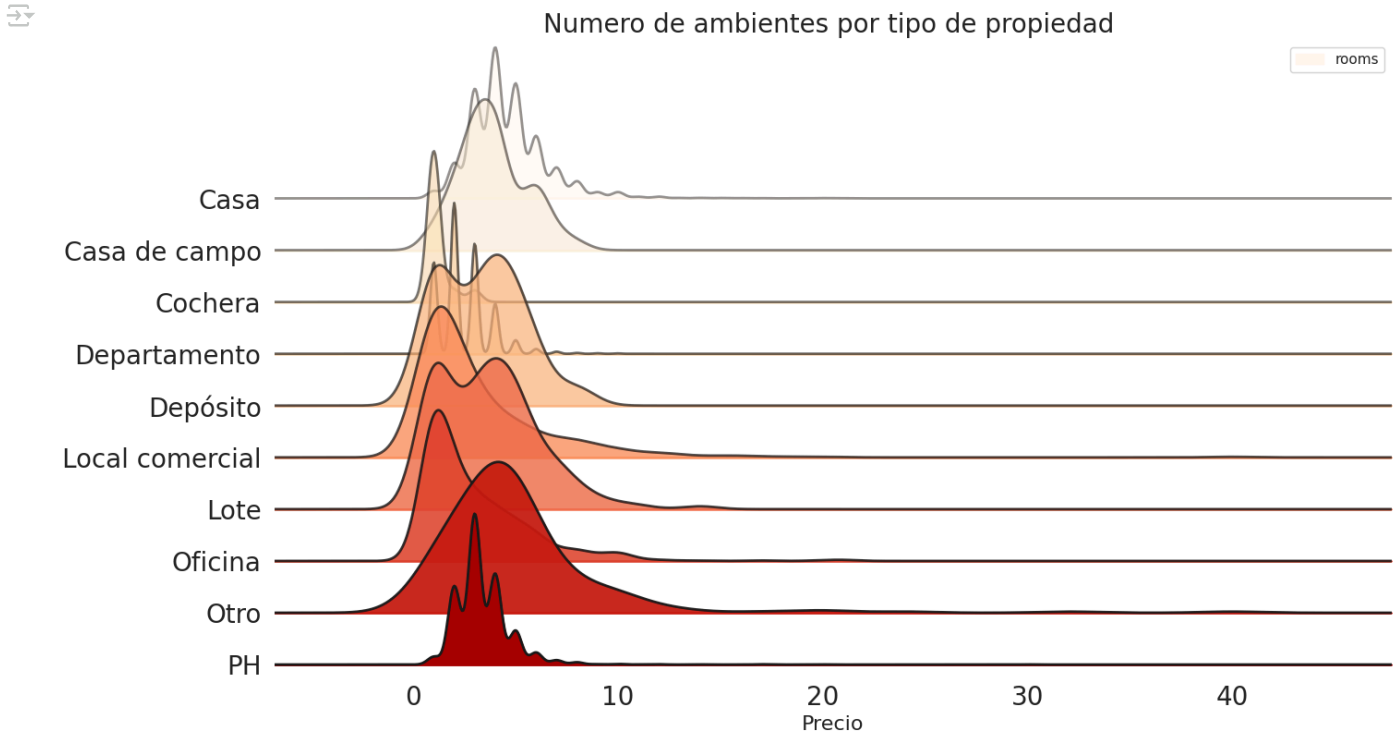
1 joyplot(viviendas, by = 'property_type', column = 'bathrooms', figsize=(15,8), fade = True, ylim='own', colormap= cm.RdPu,
2 legend=True, alpha=0.4)
3
4 plt.title('Numero de baños por tipo de propiedad ', fontsize=20)
5 plt.rc("font", size=20)
6 plt.xlabel('Precio', fontsize=16, alpha=1);
7
8
```



```

1 joyplot(viviendas, by = 'property_type', column = 'rooms', figsize=(15,8), fade = True, ylim='own',colormap= cm.OrRd,
2 legend=True, alpha=0.4)
3
4 plt.title('Numero de ambientes por tipo de propiedad ', fontsize=20)
5 plt.rc("font", size=20)
6 plt.xlabel('Precio', fontsize=16, alpha=1);
7

```



Vemos que ambas distribuciones están sesgadas a la izquierda (esto es porque cuentan con varios outliers extremos), como vimos anteriormente en el boxplot. En el caso de baños se distribuyen mayormente entre 1 y 3 mientras que en el caso de ambientes se distribuyen en su mayoría entre 1 y 5. Varios tipos de propiedad presentan distribuciones con varios picos, indicando que hay por ejemplo departamentos con distintos números de ambientes y de baños, como era de esperar.

Variable categorica vs variable categorica

Queremos ver ahora cómo varía el tipo de propiedad según el tipo de operación, es decir, cuántas de las propiedades en alquiler, venta y alquiler temporal son casas, departamentos, locales comerciales, etc. Para eso hacemos un barplot utilizando ambas variables

```
1 viviendas.stb.freq(['property_type', 'operation_type'], style=True)
2 #se puede observar de la tabla que mas de la mitad del dataset contiene datos de venta de departamento y casas
```

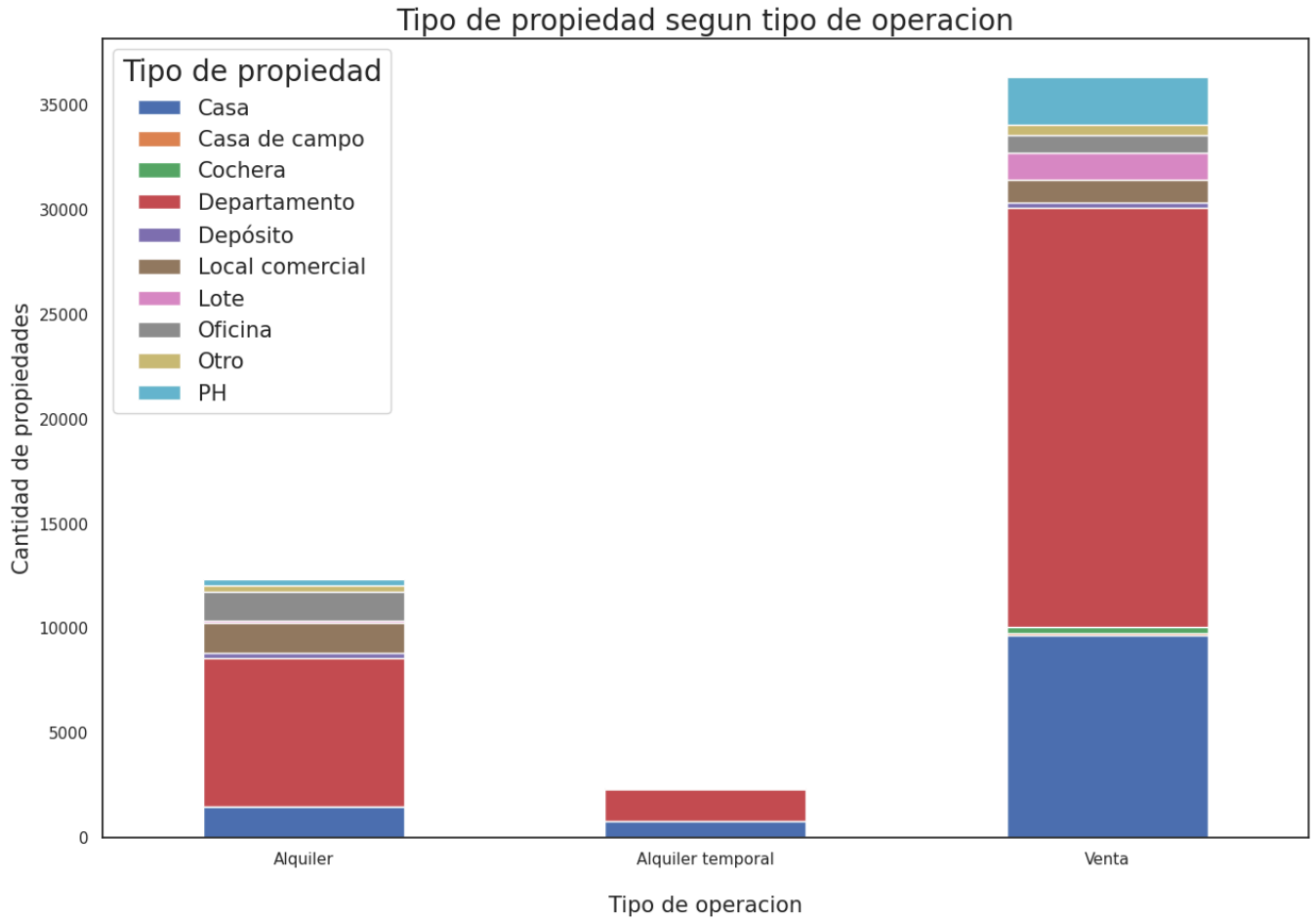


	property_type	operation_type	count	percent	cumulative_count	cumulative_percent
0	Departamento	Venta	20,040	39.26%	20,040	39.26%
1	Casa	Venta	9,685	18.97%	29,725	58.23%
2	Departamento	Alquiler	7,073	13.86%	36,798	72.08%
3	PH	Venta	2,311	4.53%	39,109	76.61%
4	Departamento	Alquiler temporal	1,497	2.93%	40,606	79.54%
5	Casa	Alquiler	1,464	2.87%	42,070	82.41%
6	Local comercial	Alquiler	1,428	2.80%	43,498	85.21%
7	Oficina	Alquiler	1,366	2.68%	44,864	87.88%
8	Lote	Venta	1,304	2.55%	46,168	90.44%
9	Local comercial	Venta	1,101	2.16%	47,269	92.60%
10	Oficina	Venta	832	1.63%	48,101	94.23%
11	Casa	Alquiler temporal	791	1.55%	48,892	95.77%
12	Otro	Venta	475	0.93%	49,367	96.71%
13	PH	Alquiler	333	0.65%	49,700	97.36%
14	Otro	Alquiler	297	0.58%	49,997	97.94%
15	Cochera	Venta	274	0.54%	50,271	98.48%
16	Depósito	Alquiler	252	0.49%	50,523	98.97%
17	Depósito	Venta	219	0.43%	50,742	99.40%
18	Lote	Alquiler	92	0.18%	50,834	99.58%
19	Casa de campo	Venta	85	0.17%	50,919	99.75%
20	Cochera	Alquiler	33	0.06%	50,952	99.81%
21	Otro	Alquiler temporal	29	0.06%	50,981	99.87%
22	PH	Alquiler temporal	23	0.05%	51,004	99.91%
23	Casa de campo	Alquiler temporal	21	0.04%	51,025	99.95%
24	Oficina	Alquiler temporal	10	0.02%	51,035	99.97%
25	Casa de campo	Alquiler	9	0.02%	51,044	99.99%
26	Local comercial	Alquiler temporal	3	0.01%	51,047	100.00%
27	Lote	Alquiler temporal	2	0.00%	51,049	100.00%

```

1 cross = pd.crosstab(viviendas['operation_type'].sort_values() , viviendas['property_type'])
2 cross.plot(kind="bar", stacked=True, rot=0, figsize=(15,10));
3
4 plt.xlabel ('Tipo de operacion', labelpad=20,  fontsize=15)
5 plt.ylabel('Cantidad de propiedades', fontsize=15)
6 plt.title('Tipo de propiedad segun tipo de operacion', fontsize=20);
7 plt.legend(title='Tipo de propiedad', fontsize=15, title_fontsize=20)
8 ;

```



En este gráfico se puede observar que en Venta disponemos de la mayor cantidad de datos, luego en alquiler y unos pocos datos en alquiler temporal. Independientemente de que tipo de operación se trate, es notable la mayor participación de departamento y casas.

✓ Analisis multivariado

Para tener una visualización inicial de algunas variables, vamos a realizar gráficos de distribución del precio de una propiedad, pero agrupados por tipo de operación y tipo de propiedad. Para ello es necesario realizar una tabla de pivot para separar algunas columnas.

```

1 #Primero duplico el dataframe y le asigno como columna el indice, ya que es necesario luego para referenciar cada registro
2 #desde la tabla pivot
3
4 viviendas2 = viviendas
5 viviendas2 = viviendas.reset_index(level=0)
6
7 #Vuelvo a limitar el rango de precios que quiero graficar
8 precio = viviendas2[viviendas2['price']<100000]
9
10 #Creo una tabla pivot donde me separe la columna operation_type en las 3 variables, cada una con el precio correspondiente
11 pivot_tipoProp = precio.pivot_table(index= ['index', 'property_type'], columns='operation_type', values='price').reset_index()
12 pivot_tipoProp

```



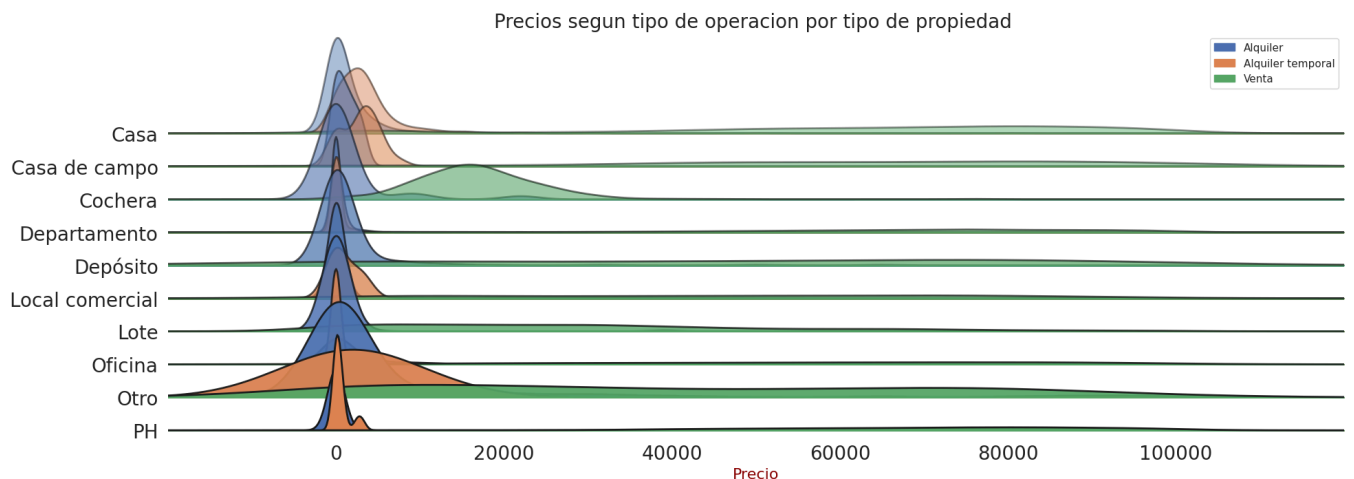
operation_type	index	property_type	Alquiler	Alquiler temporal	Venta
0	1	Casa	16.256158	NaN	NaN
1	2	Departamento	NaN	NaN	74000.0
2	3	Departamento	13.793103	NaN	NaN
3	4	Departamento	NaN	NaN	74000.0
4	6	Departamento	NaN	NaN	74000.0
...
27254	54074	Departamento	NaN	NaN	96000.0
27255	54077	Departamento	NaN	NaN	44010.0
27256	54080	Departamento	NaN	NaN	40410.0
27257	54081	Departamento	NaN	NaN	80000.0
27258	54082	Departamento	NaN	NaN	61880.0

27259 rows × 5 columns

```

1 #Realizamos luego un grafico de joyplot que me informe el precio de cada tipo de operacion segun el tipo de propiedad consi:
2 joyplot(pivot_tipoProp, by = 'property_type', column = ['Alquiler', 'Alquiler temporal', 'Venta'], fade = True, ylim='own',
3 legend=True, alpha=0.4)
4
5 plt.title('Precios segun tipo de operacion por tipo de propiedad ', fontsize=20)
6 plt.rc("font", size=20)
7 plt.xlabel('Precio', fontsize=16, color='darkred', alpha=1);
8

```



Observamos que para cualquier tipo de propiedad, los alquileres y alquileres temporales están distribuidos en los valores más bajos, sin demasiada variación entre tipo de propiedad. Por otro lado, las ventas, como es de esperar, se distribuyen en precios más altos, pero también esta distribución es muy amplia, abarcando un amplio rango de precios con similar frecuencia.

Debido al resultado anterior es necesario comenzar a considerar exactamente aquellos modelos que queremos predecir en el futuro. Ya que los precios (variable target) varían según el tipo de propiedad, tipo de operación y probablemente localización, es necesario realizar agrupamientos para poder determinar esta variable de manera más precisa (anteriormente calculábamos el precio promedio total y veíamos que no presentaba una alta correlación con las otras variables).

Así, vamos a utilizar solamente algunos tipos de operación y de propiedad, los que están más representados y los que sean de mayor interés, ya que utilizarlos todos demandaría una alta cantidad de análisis gráfico y de modelos, y como muchos de ellos no tienen gran cantidad de datos (como la gran mayoría de alquileres temporales o alquileres de casa de campo por ejemplo), comenzaremos solo trabajando con departamentos y casas, que son los más representados en nuestro dataset, y solo utilizaremos operaciones de alquiler y de venta.

Agrupación por tipo de operación y tipo de propiedad

```

1 #Realizamos filtro para quedarnos solo con las operaciones de alquiler y venta y con el tipo de propiedad Casa y departam
2 #Se generan así 4 datasets distintos para luego trabajar más cómodamente

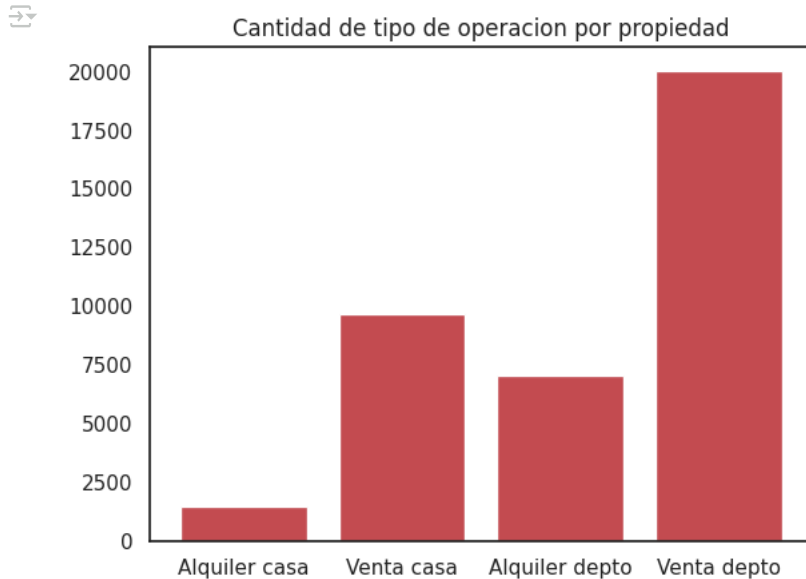
```

```

3
4 alquiler_casa = viviendas[(viviendas['operation_type'] == 'Alquiler') & (viviendas['property_type'] == 'Casa')]
5
6 alquiler_dpto = viviendas[(viviendas['operation_type'] == 'Alquiler') & (viviendas['property_type'] == 'Departamento')]
7
8 venta_casa = viviendas[(viviendas['operation_type'] == 'Venta') & (viviendas['property_type'] == 'Casa')]
9
10 venta_dpto = viviendas[(viviendas['operation_type'] == 'Venta') & (viviendas['property_type'] == 'Departamento')]
11

1 #Construyo un grafico de barras solo para tener una idea de la cantidad de valores que tengo en cada dataset
2 labels = ['Alquiler casa', 'Venta casa', 'Alquiler depto', 'Venta depto']
3 valores = [alquiler_casa.shape[0], venta_casa.shape[0], alquiler_dpto.shape[0], venta_dpto.shape[0]]
4
5 fig, ax = plt.subplots()
6 ax.bar(labels,valores, color = 'r')
7 plt.title('Cantidad de tipo de operacion por propiedad')
8 plt.show()

```

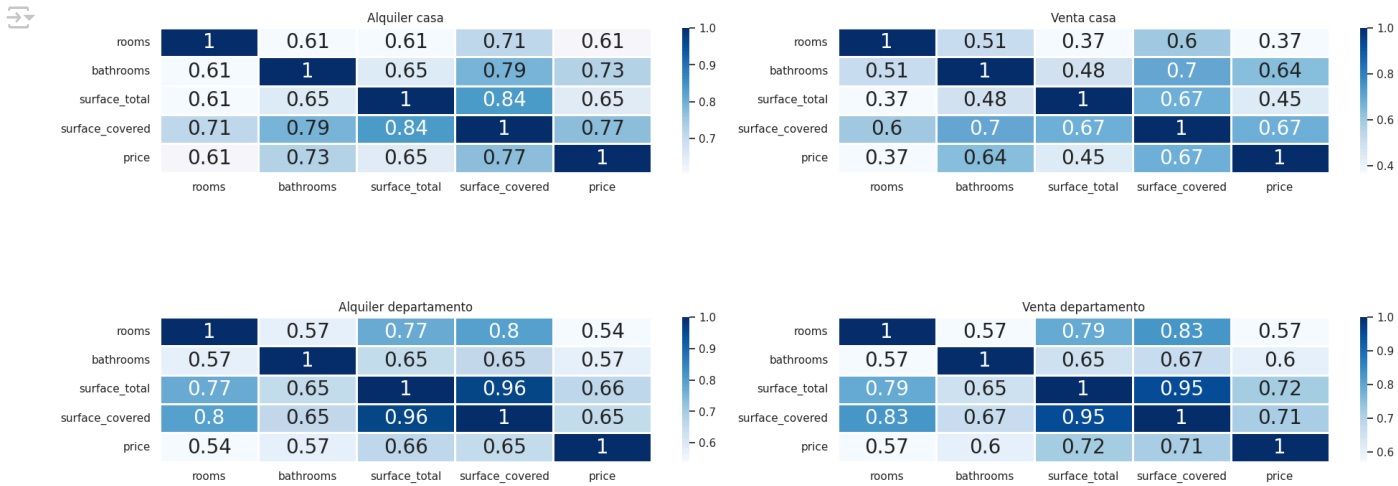


Se ve por ejemplo que en alquiler casa tengo bastante menos valores que en el resto y pocos valores en general.

```

1 listacorr = [alquiler_casa, venta_casa, alquiler_dpto, venta_dpto]
2 titulos = ['Alquiler casa', 'Venta casa', 'Alquiler departamento', 'Venta departamento']
3 ejes = []
4
5 plt.figure(figsize=(20,7))
6 for j,i in enumerate(listacorr):
7     ax = plt.subplot(2,2,j+1)
8     correlation = i.dropna().drop(['Latitud', 'Longitud'], axis = 1).corr(method='spearman')
9     sns.heatmap(correlation, linewidth = 2, annot=True, cmap="Blues")
10    plt.tight_layout()
11    ejes.append(ax)
12    plt.subplots_adjust(hspace = 1)
13
14 for j,i in enumerate(titulos):
15     ejes[j].set(title= i)
16

```



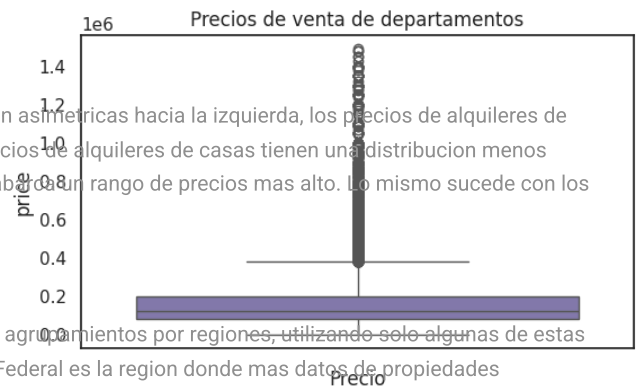
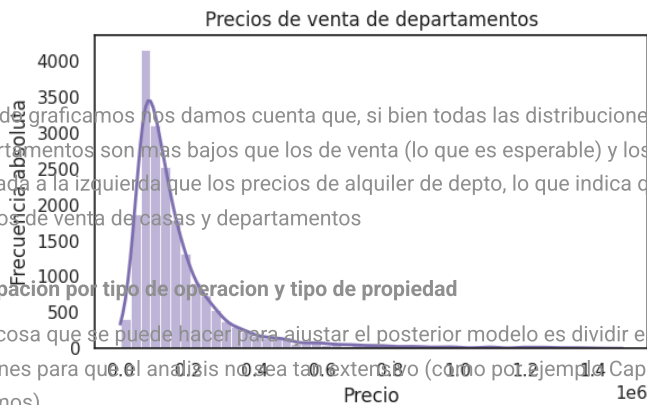
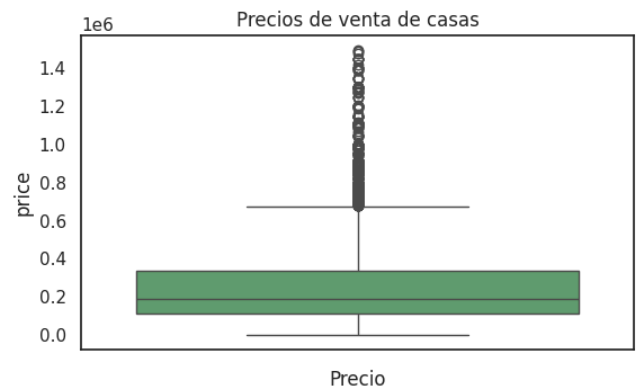
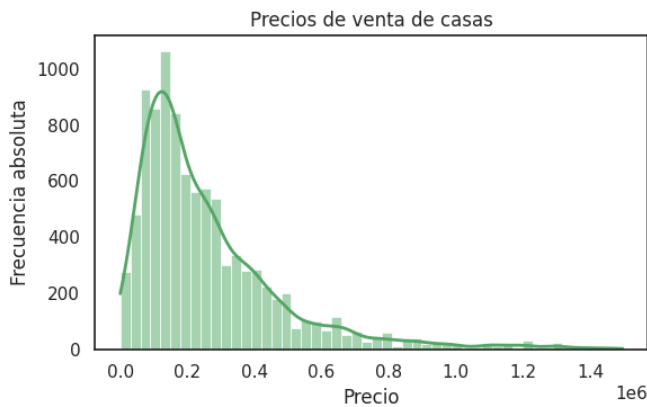
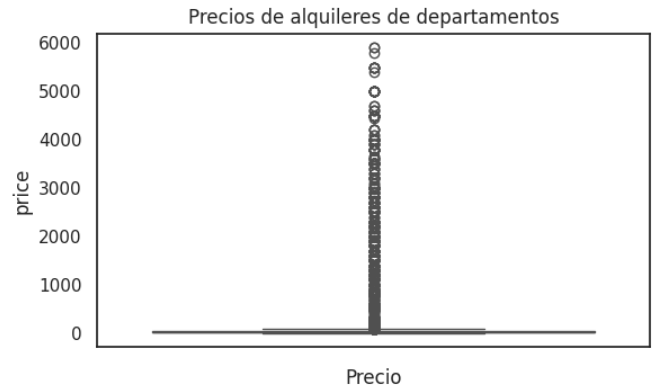
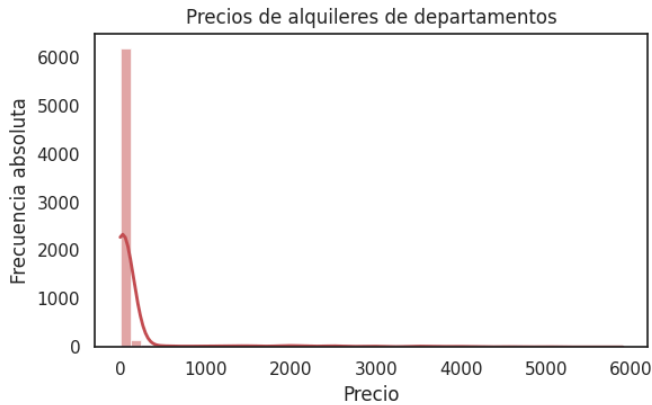
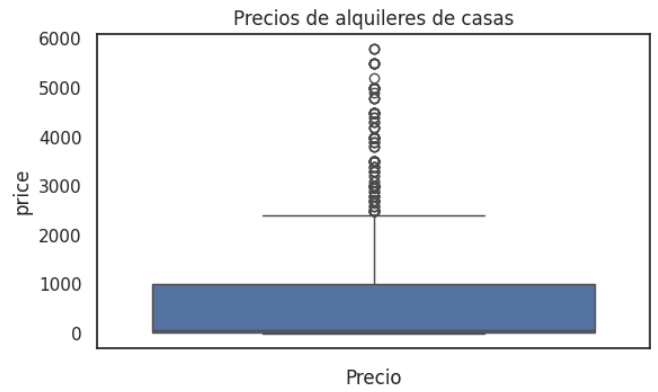
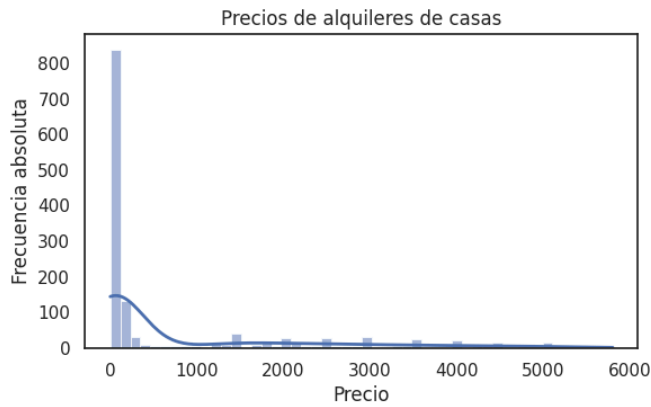
Observamos que al segmentar por propiedad y tipo de operacion, la correlación entre precios y el resto de las variables aumenta en comparación a lo que veíamos cuando los precios se consideraban todos en su conjunto, aunque observamos que la correlación entre variables depende del tipo de operacion y de propiedad con la que se trabaje.

Realizamos un histograma para ver como se distribuye el precio según un tipo de propiedad y un tipo de operación determinada


```

1 #Grafico histograma
2
3 sns.set(style="white", rc={"lines.linewidth": 2})
4 fig, ax = plt.subplots(4,2, figsize=(12,15))
5
6 sns.histplot(x='price',
7             data= alquiler_casa[alquiler_casa['price']<6000],
8             color='b',
9             ax=ax[0][0],
10            bins = 50,
11            kde=True).set(title='Precios de alquileres de casas', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
12
13 sns.histplot(x='price',
14            data= alquiler_dpto[alquiler_dpto['price']<6000] ,
15            color='r',
16            ax=ax[1][0],
17            bins = 50,
18            kde=True).set(title='Precios de alquileres de departamentos', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
19
20 sns.histplot(x='price',
21            data= venta_casa[venta_casa['price']<150000],
22            color='g',
23            ax=ax[2][0],
24            bins = 50,
25
26            kde=True).set(title='Precios de venta de casas', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
27
28 sns.histplot(x='price',
29            data= venta_dpto[venta_dpto['price']<150000] ,
30            color='m',
31            ax=ax[3][0],
32            bins = 50,
33
34            kde=True).set(title='Precios de venta de departamentos', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
35
36 plt.subplots_adjust(hspace = 0.3)
37
38 #Grafico boxplot
39
40 sns.boxplot(alquiler_casa[alquiler_casa['price']<6000]['price'] , ax = ax[0][1], color = 'b').set(title='Precios de alquileres de casas')
41 sns.boxplot(alquiler_dpto[alquiler_dpto['price']<6000]['price'] , ax = ax[1][1], color = 'r').set(title='Precios de alquileres de departamentos')
42 sns.boxplot(venta_casa[venta_casa['price']<150000]['price'] , ax = ax[2][1],color = 'g').set(title='Precios de venta de casas')
43 sns.boxplot(venta_dpto[venta_dpto['price']<150000]['price'] , ax = ax[3][1],color = 'm').set(title='Precios de venta de departamentos')
44
45 fig.tight_layout()
46

```



Cuando graficamos nos damos cuenta que, si bien todas las distribuciones son asimétricas hacia la izquierda, los precios de alquileres de departamentos son mas bajos que los de venta (lo que es esperable) y los precios de alquileres de casas tienen una distribución menos sesgada a la izquierda que los precios de alquiler de depto, lo que indica que abarca un rango de precios mas alto. Lo mismo sucede con los precios de venta de casas y departamentos

Agrupación por tipo de operación y tipo de propiedad

Otra cosa que se puede hacer para ajustar el posterior modelo es dividir estos agrupamientos por regiones, utilizando solo algunas regiones para que el análisis no sea tan extenso (como por ejemplo la Capital Federal es la region donde mas datos de propiedades tenemos).

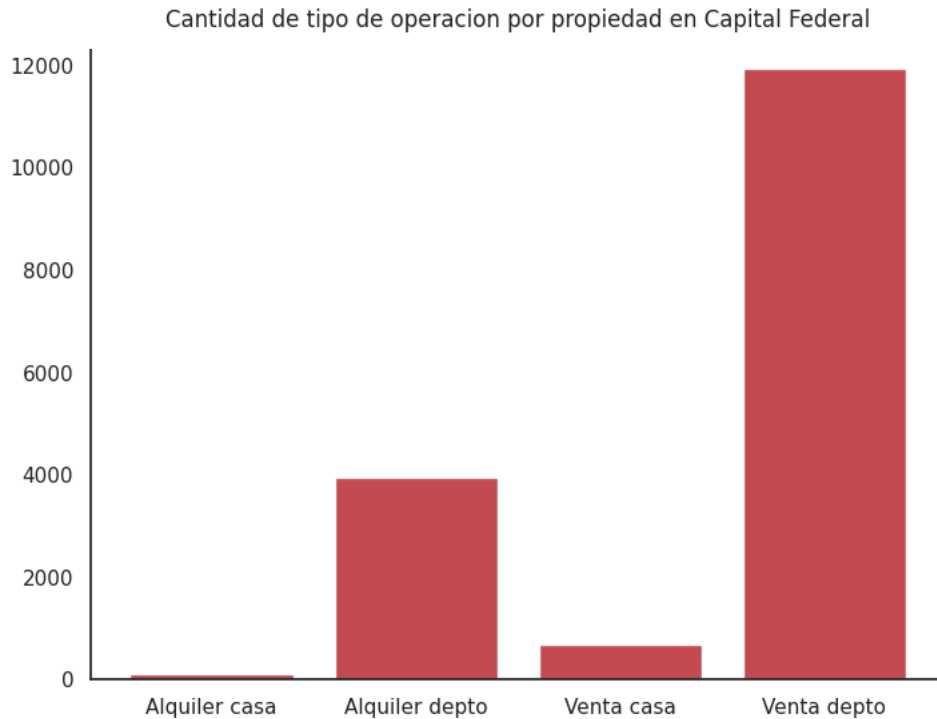
```
1 #Realizamos filtro para quedarnos solo con las operaciones de alquiler y venta y con el tipo de propiedad Casa y departamento
2 #Se generan así 4 datasets distintos para luego trabajar mas comodamente
3
4 alquiler_casa_capital = viviendas[(viviendas['operation_type'] == 'Alquiler') & (viviendas['property_type'] == 'Casa') & (viviendas['region'] == 'Capital Federal')]
5
6 alquiler_dpto_capital = viviendas[(viviendas['operation_type'] == 'Alquiler') & (viviendas['property_type'] == 'Departamento') & (viviendas['region'] == 'Capital Federal')]
7
8 venta_casa_capital = viviendas[(viviendas['operation_type'] == 'Venta') & (viviendas['property_type'] == 'Casa') & (viviendas['region'] == 'Capital Federal')]
9
10 venta_dpto_capital = viviendas[(viviendas['operation_type'] == 'Venta') & (viviendas['property_type'] == 'Departamento') & (viviendas['region'] == 'Capital Federal')]
```

```
1 #Construyo un grafico de barras para tener una idea de la cantidad de valores que tengo en cada dataset
2 labels = ['Alquiler casa', 'Alquiler depto', 'Venta casa', 'Venta depto']
```

```

3 valores = [alquiler_casa_capital.shape[0], alquiler_dpto_capital.shape[0], venta_casa_capital.shape[0], venta_dpto_capital
4
5 fig = plt.figure()
6 ax = fig.add_axes([0,0,1,1])
7 ax.bar(labels,valores, color = 'r')
8 plt.title('Cantidad de tipo de operacion por propiedad en Capital Federal')
9 plt.show()
10

```

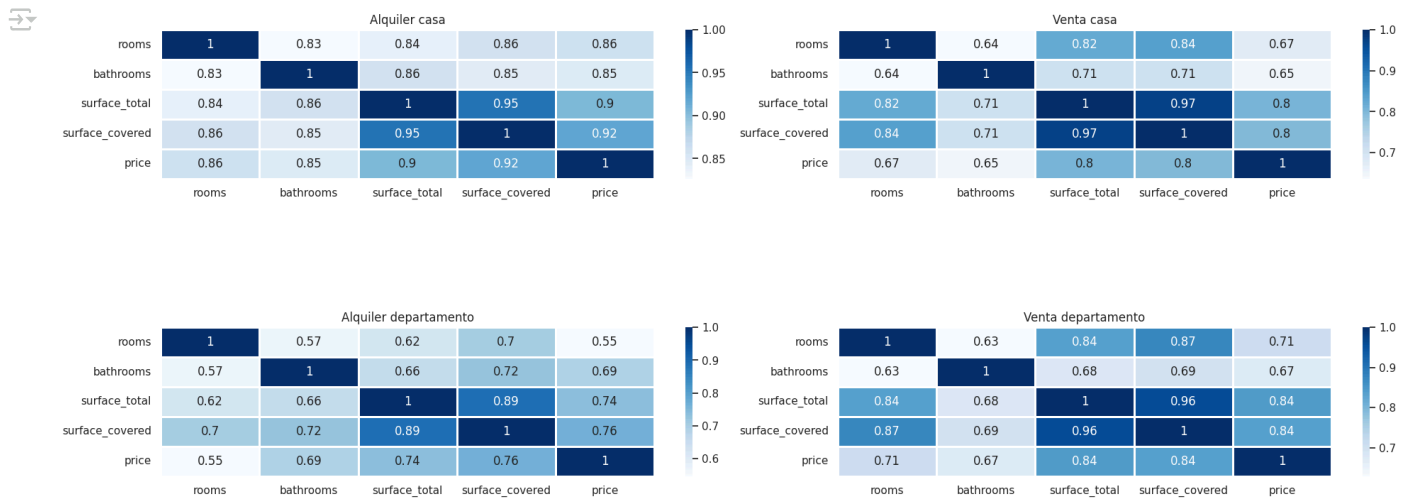


Vemos que en Capital Federal, hay muy pocas operaciones con casas, por lo que los modelos generados seguramente no serán muy robustos (no tenemos tantos datos para entrenar el modelo). En el caso de departamentos tenemos más datos, tantos de alquiler como de venta.

```

1 listacorr_capital = [alquiler_casa_capital, alquiler_dpto_capital, venta_casa_capital, venta_dpto_capital]
2 titulos = ['Alquiler casa', 'Venta casa', 'Alquiler departamento', 'Venta departamento']
3 ejes = []
4
5 plt.figure(figsize=(20,7))
6 for j,i in enumerate(listacorr_capital):
7     ax = plt.subplot(2,2,j+1)
8     correlation = i.dropna().drop(['Latitud', 'Longitud'], axis = 1).corr(method='spearman')
9     sns.heatmap(correlation, linewidth = 2, annot=True, cmap="Blues")
10    plt.tight_layout()
11    ejes.append(ax)
12    plt.subplots_adjust(hspace = 1)
13
14 for j,i in enumerate(titulos):
15     ejes[j].set(title= i)

```

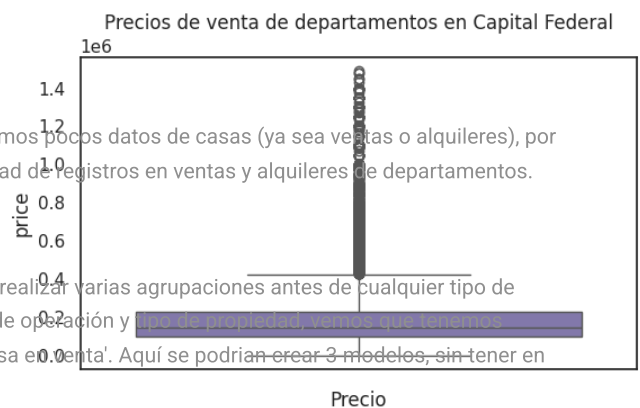
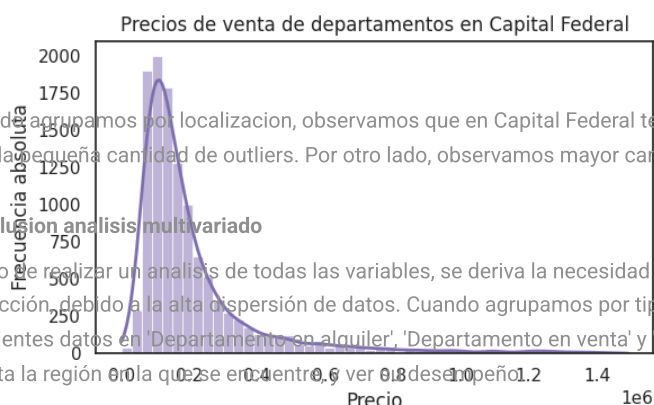
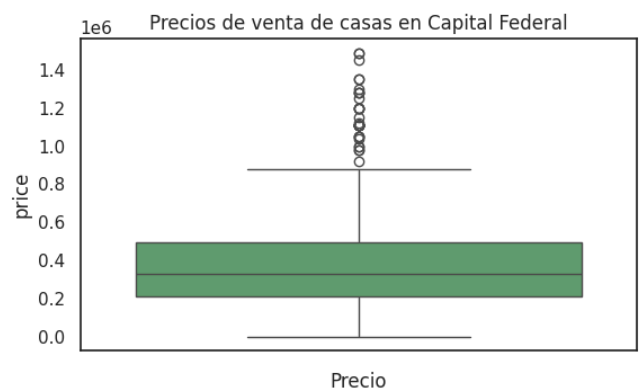
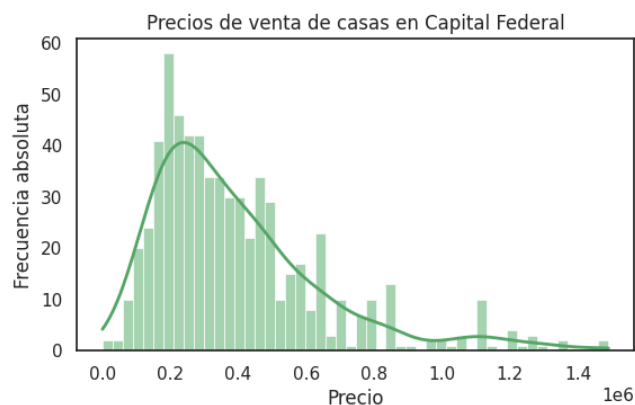
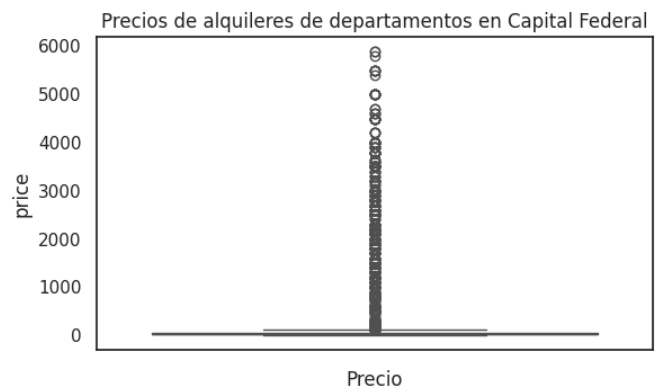
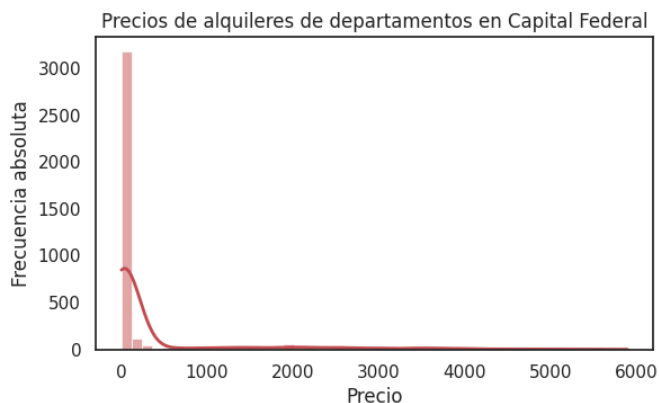
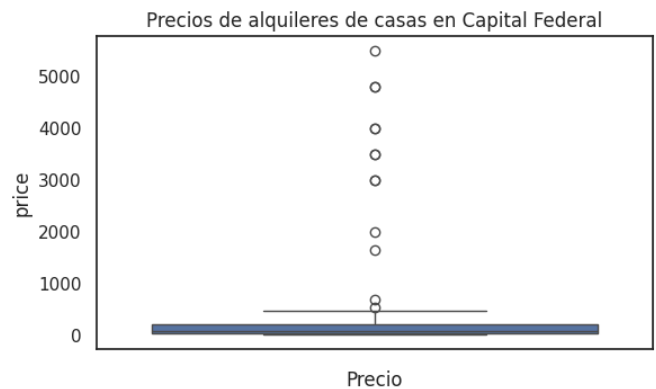
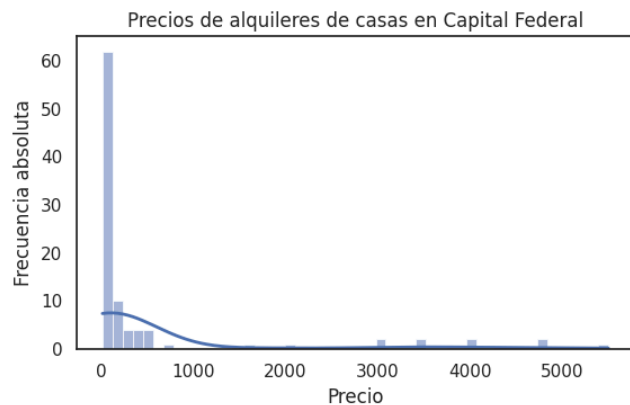


Al realizar esta segmentación por provincia (en este caso Capital Federal), observamos que la relación entre la variable precio y el resto de las variables aumenta con respecto a la agrupación por tipo de propiedad y tipo de operación. Sin embargo, depende de la agrupación. En el caso de alquiler de casa la correlación es alta, pero esto es probablemente porque hay poca cantidad de datos. En el caso de ventas (tanto de casas como de deptos) la relación es alta con las variables de superficies, y un poco menos con las variables de ambientes y baños

```

1 #Grafico histograma
2
3 sns.set(style="white", rc={"lines.linewidth": 2})
4 fig, ax = plt.subplots(4,2, figsize=(12,15))
5
6 sns.histplot(x='price',
7             data= alquiler_casa_capital[alquiler_casa_capital['price']<6000],
8             color='b',
9             ax=ax[0][0],
10            bins = 50,
11            kde=True).set(title='Precios de alquileres de casas en Capital Federal', xlabel= 'Precio', ylabel= 'Frecuencia')
12
13 sns.histplot(x='price',
14             data= alquiler_dpto_capital[alquiler_dpto_capital['price']<6000] ,
15             color='r',
16             ax=ax[1][0],
17            bins = 50,
18            kde=True).set(title='Precios de alquileres de departamentos en Capital Federal', xlabel= 'Precio', ylabel= 'Frecuencia')
19
20 sns.histplot(x='price',
21             data= venta_casa_capital[venta_casa_capital['price']<1500000],
22             color='g',
23             ax=ax[2][0],
24            bins = 50,
25
26            kde=True).set(title='Precios de venta de casas en Capital Federal', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
27
28 sns.histplot(x='price',
29             data= venta_dpto_capital[venta_dpto_capital['price']<1500000] ,
30             color='m',
31             ax=ax[3][0],
32            bins = 50,
33
34            kde=True).set(title='Precios de venta de departamentos en Capital Federal', xlabel= 'Precio', ylabel= 'Frecuencia absoluta')
35
36 plt.subplots_adjust(hspace = 0.3)
37
38 #Grafico boxplot
39
40 sns.boxplot(alquiler_casa_capital[alquiler_casa_capital['price']<6000]['price'] , ax = ax[0][1], color = 'b').set(title='Precios de alquileres de casas en Capital Federal')
41 sns.boxplot(alquiler_dpto_capital[alquiler_dpto_capital['price']<6000]['price'] , ax = ax[1][1], color = 'r').set(title='Precios de alquileres de departamentos en Capital Federal')
42 sns.boxplot(venta_casa_capital[venta_casa_capital['price']<1500000]['price'] , ax = ax[2][1], color = 'g').set(title='Precios de venta de casas en Capital Federal')
43 sns.boxplot(venta_dpto_capital[venta_dpto_capital['price']<1500000]['price'] , ax = ax[3][1], color = 'm').set(title='Precios de venta de departamentos en Capital Federal')
44
45 fig.tight_layout()
46

```



Cuando agrupamos por localización, observamos que en Capital Federal tenemos pocos datos de casas (ya sea ventas o alquileres), por esto la pequeña cantidad de outliers. Por otro lado, observamos mayor cantidad de registros en ventas y alquileres de departamentos.

Conclusión análisis multivariado

Luego de realizar un análisis de todas las variables, se deriva la necesidad de realizar varias agrupaciones antes de cualquier tipo de predicción, debido a la alta dispersión de datos. Cuando agrupamos por tipo de operación y tipo de propiedad, vemos que tenemos suficientes datos en 'Departamento en alquiler', 'Departamento en venta' y 'Casa en venta'. Aquí se podrían crear 3 modelos, sin tener en cuenta la región en la que se encuentra, o ver si se diferencia por región.

Otros modelos mas precisos se podrían generar agrupando a su vez por región (provincia). En este caso vemos que las correlaciones aumentan, aunque obviamente dispondremos de menos datos. 'Departamento en venta en Capital Federal' y 'Departamento en alquiler en Capital Federal' presentan una buena correlación entre la variable target (precio) y el resto de las variables. Se elige Capital Federal por ser la provincia donde mayor cantidad de datos tenemos (cerca de un 45%, como pudimos ver en el PiePlot), aunque también podrían ensayarse modelos de diferentes regiones. Para realizar las predicciones, se utilizan las variables 'rooms', 'bathrooms', 'surface_covered' y 'surface_total', que muestran una buena correlación con la variable precio.

Llenado de nulos en variables bathrooms y rooms

Llegados a este punto, donde ya se decidió que primeros modelos se realizarán, podemos completar los valores nulos de las variables baños y ambientes. Esto es así ya que no se llenarán todos los nulos, si no solo los que nos interesan para los modelos. Para un mismo tipo de vivienda con una superficie similar, es probable que tengan el mismo número de ambientes y de baños. Por lo tanto, para llenar los nulos de

estas dos variables, se va a agrupar segun el tipo de propiedad y su superficie cubierta (no la superficie total), y se calculará la mediana de cada grupo.

Primero llenamos los nulos en los departamentos. Segun los analisis anteriores, la mayoría de las superficies de los departamentos están concentrados en los valores de superficie menores a 250 m2. Es necesario realizar un filtro ya que si no estaríamos considerando una alta cantidad de outliers que pueden generar muchos errores en el momento de llenar estos valores

```
1 viviendas_dpto = viviendas[(viviendas['property_type']=='Departamento')]
2 viviendas_dptosup = viviendas[(viviendas['property_type']=='Departamento') & (viviendas['surface_covered']<250)]
3
4 (viviendas_dptosup['property_type'].count())/(viviendas_dpto['property_type'].count())*100
```

↗ 98.10905277874869

Vemos que los departamentos menores a 250 m2 son el 98%, por lo que realizamos este filtro previo a la imputación (limitando los outliers pero sin perder un numero significativo de datos), generamos un rango de superficies y agrupamos por este rango para luego llenar los valores nulos con la mediana de cada grupo

```
1 #Llenado nulos en departamentos
2 viviendas_dptosup['rangosup'] = pd.cut(viviendas_dptosup['surface_covered'], 40) #Genero 40 rangos de superf entre 0 y 250
3
4 viviendas_dptosup['bathrooms'] = viviendas_dptosup['bathrooms'].fillna(viviendas_dptosup.groupby(['property_type', 'rangosup']).
5 viviendas_dptosup['rooms'] = viviendas_dptosup['rooms'].fillna(viviendas_dptosup.groupby(['property_type', 'rangosup'])['rooms
6
7 viviendas.update(viviendas_dptosup)
```

Para realizar la imputación en el caso de las casas, si usamos como límite superficies menores a 250 m2 (como en el caso de departamentos) solo estaríamos comprendiendo el 80% de las casas. Esto es lógico ya que por lo general las casas poseen más superficie. Por lo tanto, es necesario aumentar el limite de superficie cubierta, y para eso tomamos aquellas casas menores de 600 m2, que corresponde a un 98% de las casas

```
1 viviendas_casa = viviendas[(viviendas['property_type']=='Casa')]
2 viviendas_casasup = viviendas[(viviendas['property_type']=='Casa') & (viviendas['surface_covered']<600)]
3
4 (viviendas_casasup['property_type'].count())/(viviendas_casa['property_type'].count())*100
```

↗ 98.03182579564489

```
1 #Llenado nulos en casas
2 viviendas_casasup['rangosup'] = pd.cut(viviendas_casasup['surface_covered'], 50) #Genero 50 rangos de superf entre 0 y 600
3
4 viviendas_casasup['bathrooms'] = viviendas_casasup['bathrooms'].fillna(viviendas_casasup.groupby(['property_type', 'rangosup']).
5 viviendas_casasup['rooms'] = viviendas_casasup['rooms'].fillna(viviendas_casasup.groupby(['property_type', 'rangosup'])['bathr
6
7 viviendas.update(viviendas_casasup)
```

Con estos datos completos, podemos comenzar a generar modelos de regresión para predecir la variable 'precio'.

✓ Modelos de regresion

Filtros y transformacion de las variables

Elegimos realizar predicciones para los departamentos en venta de Capital Federal, ya que disponemos de bastantes datos. El primer modelo que haremos es el mas sencillo de todos, una regresion lineal simple, ya que como vimos hay alta correlacion principalmente entre la variable target precio y la variable superficie cubierta (y superficie total), por lo que si bien es simple podria ser un buen inicio.

Veremos como es la distribucion de la variable target en estos casos

```
1 venta_dpto_capital = viviendas[(viviendas['operation_type'] == 'Venta') & (viviendas['property_type'] == 'Departamento')]
2
3 print(venta_dpto_capital['price'].skew())
```

↗ 6.6494491150969255

Como era de esperar debido a las distribuciones observadas anteriormente, estas variables estan muy alejadas de una distribucion normal, ya que presentan una simetria muy positiva. Antes de realizar una transformacion logaritmica, veremos de filtrar el rango de precios para ver

si podemos utilizar los valores donde estan mas concentrados los datos. Para esto utilizaremos los valores limites del boxplot obtenidos en el analisis multivariado al graficar el precio luego de agrupar por estas categorias

```
1 filtro_venta_dpto_capital = venta_dpto_capital[(venta_dpto_capital['price']<450000) & (venta_dpto_capital['price']>0) & (v

1 print(filtro_venta_dpto_capital['price'].skew())

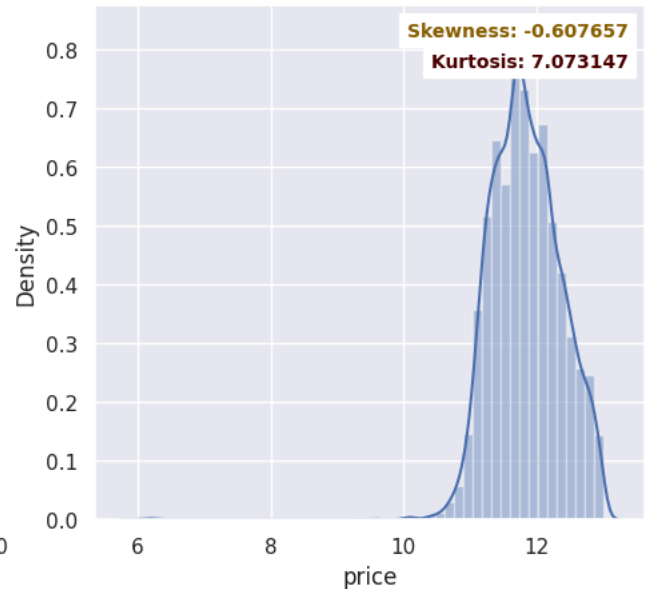
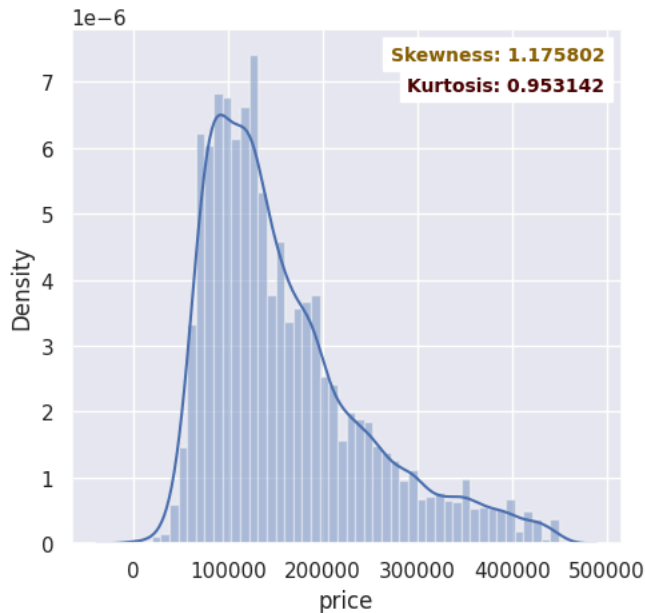
1.175802351533701
```

Si bien los valores de asimetria disminuyen bastante, siguen siendo altos (es recomendable que sea mas cercano a cero), por lo que vamos a hacer una transformacion a logaritmo. Para ello uso una funcion determinada:

```
1 def plot_compare(variable , data ):
2     '''
3     Generacion de una comparativa de una variables , para saber si se realiza una trasnformacion logaritmica o no
4
5     Parameters:
6         variable (Pandas.Series): Serie de la variable a estudiar
7         data (DataFrame): Dataframe donde realizar el estudio
8
9     Returns:
10         y (numpy.ndarray): es un array que determina el resultado de la funcion pasando por todos los valores
11     '''
12
13     #check skew variable
14     data_serie = data[variable]
15     sesgo = data_serie.skew()
16     print(f'Para la variable {variable} tiene un sesgo de {round(sesgo,4)}')
17
18
19     #genero dos variables para luego comparar
20
21     log_serie = np.log(data_serie) #Transformacion logaritmica
22
23     #genero un grafico donde pueda comparar su distribucion
24     sns.set(rc={'figure.figsize':(11.7,5)})
25     fig, axs = plt.subplots(ncols=2)
26     sns.distplot(data_serie, ax=axs[0]);
27     sns.distplot(log_serie, ax=axs[1])
28     for ax, d in zip(axs, [data_serie, log_serie]):
29         ax.text(x=0.97, y=0.97, transform=ax.transAxes, s="Skewness: %f" % d.skew(),\
30                fontweight='demibold', fontsize=10, verticalalignment='top', horizontalalignment='right',\
31                backgroundcolor='white', color='xkcd:poo brown')
32         ax.text(x=0.97, y=0.91, transform=ax.transAxes, s="Kurtosis: %f" % d.kurt(),\
33                fontweight='demibold', fontsize=10, verticalalignment='top', horizontalalignment='right',\
34                backgroundcolor='white', color='xkcd:dried blood')
35
36     return log_serie

1 log_precio_ventadpto = plot_compare('price' , filtro_venta_dpto_capital )
```

Para la variable price tiene un sesgo de 1.1758

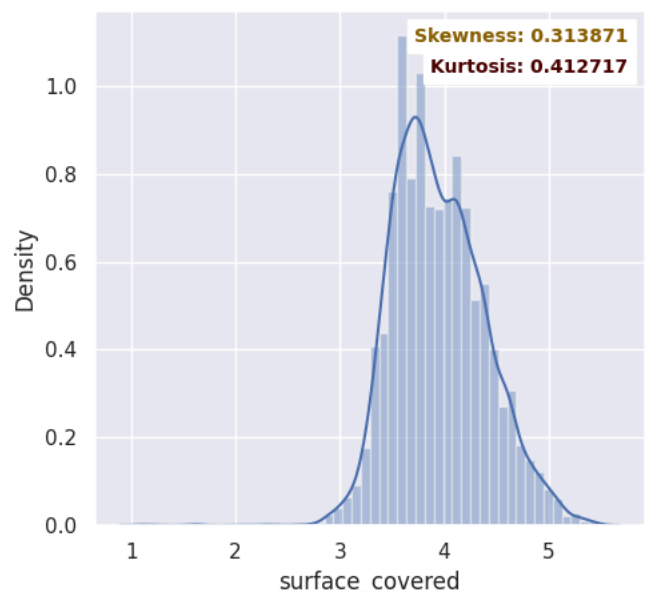
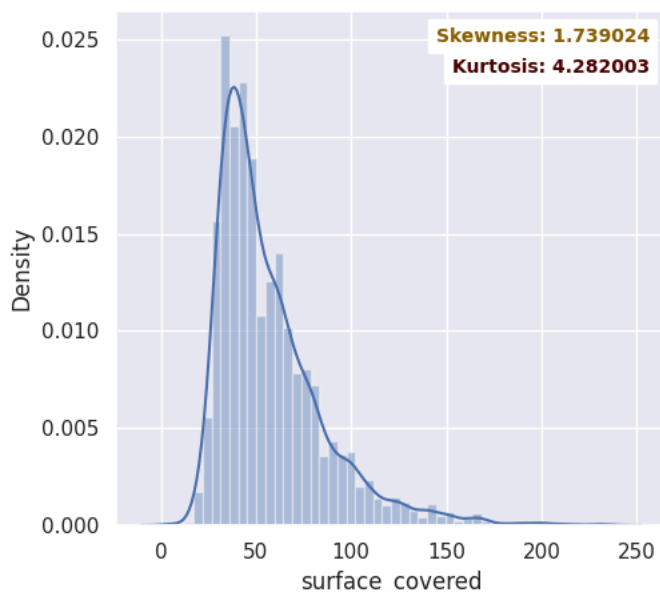


Veo que si bien la asimetría disminuye considerablemente, la distribución posee alto nivel de kurtosis, probablemente debido al alto valor de outlier que todavía poseen, incluso después del filtro inicial (aunque decidimos no eliminarlos porque representan un volumen de datos importante)

En este modelo simple se eligió como variable X a la superficie cubierta, ya que en el análisis multivariado se observó que poseía alta correlación con el precio. A esta variable también se le aplicará la transformación logarítmica

```
1 log_sup_venta_dpto = plot_compare('surface_covered' , filtro_venta_dpto_capital )
```

Para la variable surface_covered tiene un sesgo de 1.739



Vemos que es necesario aplicar la transformación de logaritmo a todas las variables que introduzcamos en el modelo, lo que se hará previo a entrenarlo

✓ Regresión lineal simple

```
1 # Modelo de venta de departamentos en capital usando la superficie cubierta
2 # como variable independiente
3
4 x = np.log(filtro_venta_dpto_capital['surface_covered'])
5 y = np.log(filtro_venta_dpto_capital['price'])
6
7 X_train = sm.add_constant(x, prepend=True)
```



```

8
9 modelo = sm.OLS(endog=y, exog=X_train)
10
11 modelo = modelo.fit()
12
13 print(modelo.summary())

```



```

=====
                        OLS Regression Results
=====
Dep. Variable:          price      R-squared:                0.546
Model:                  OLS      Adj. R-squared:             0.546
Method:                 Least Squares  F-statistic:           1.286e+04
Date:                  Thu, 14 Mar 2024  Prob (F-statistic):       0.00
Time:                  14:10:52    Log-Likelihood:        -4000.5
No. Observations:      10683      AIC:                   8005.
Df Residuals:          10681      BIC:                   8020.
Df Model:               1
Covariance Type:       nonrobust
=====
                        coef      std err          t      P>|t|      [0.025      0.975]
-----
const                8.3797       0.031    272.478     0.000     8.319     8.440
surface_covered       0.8781       0.008   113.422     0.000     0.863     0.893
=====
Omnibus:              9831.779    Durbin-Watson:           1.580
Prob(Omnibus):         0.000    Jarque-Bera (JB):        2121313.306
Skew:                  -3.797    Prob(JB):                0.00
Kurtosis:              71.615    Cond. No.                38.1
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

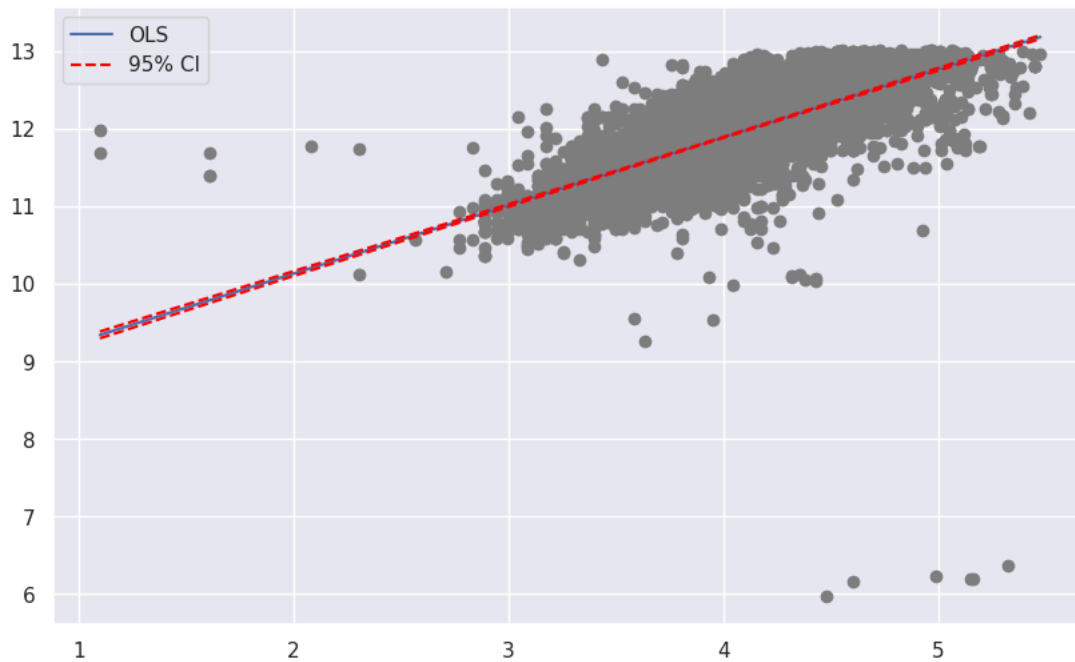
El p-value de la variable dependiente es menor a 0.05 Veo que el R-squared es igual a 0.58, lo que me dice que el modelo predice el 58% de todos mis datos, por lo que no es un modelo tan bueno.

Graficamos luego el modelo

```

1 predicciones = modelo.get_prediction(exog = X_train).summary_frame(alpha=0.05) #Summary_frame es el intervalo de confianza
2 predicciones['x'] = X_train['surface_covered'].values
3 predicciones['y'] = y
4 predicciones = predicciones.sort_values('x')
5
6
7 # Gráfico del modelo
8 # =====
9 fig, ax = plt.subplots(figsize=(10, 6))
10
11 ax.scatter(predicciones['x'], predicciones['y'], marker='o', color = "gray")
12 ax.plot(predicciones['x'], predicciones["mean"], linestyle='-', label="OLS")
13 ax.plot(predicciones['x'], predicciones["mean_ci_lower"], linestyle='--', color='red', label="95% CI")
14 ax.plot(predicciones['x'], predicciones["mean_ci_upper"], linestyle='--', color='red')
15 ax.fill_between(predicciones['x'], predicciones["mean_ci_lower"], predicciones["mean_ci_upper"], alpha=0.1)
16 ax.legend();

```



✓ Regresion lineal multiple

Probamos ahora un modelo de regresion lineal multiple, usando cuatro variables como dependientes: el numero de baños, el numero de habitaciones, la superficie cubierta y la superficie total. Como estas variables deben estar transformadas en logaritmo para normalizarse, creamos un nuevo Data Frame, donde almacenamos las variables transformadas

```
1 lst = ['surface_covered', 'surface_total', 'bathrooms', 'rooms', 'price'] #Creo una lista con las columnas del nuevo dataframe
2 filtroLog_venta_dpto_capital = filtro_venta_dpto_capital.loc[:, lst] #Elimino el resto de las columnas
3 filtroLog_venta_dpto_capital = np.log(filtroLog_venta_dpto_capital) #Transformo a logaritmo
4
```

```
1 #Generamos una funcion que contenga la generacion del modelo para venta de depto en Capital
2
3 def regMultipleVenta(columns):
4
5     X = filtroLog_venta_dpto_capital[columns] # variables independientes
6     y = filtroLog_venta_dpto_capital['price'] # variable target, transformada tambien a logaritmo
7     X = sm.add_constant(X) ## se agrega una constante necesaria para poder entrenar en regresion
8
9     model = sm.OLS(y, X).fit()
10
11     print(model.summary())
12     return model
```

```
1 #Llamo a la funcion para que corra el modelo
2 model = regMultipleVenta(['surface_covered', 'surface_total', 'bathrooms', 'rooms'])
3 model
4
```



OLS Regression Results

```
=====
Dep. Variable:          price    R-squared:                0.569
Model:                  OLS      Adj. R-squared:           0.569
Method:                 Least Squares    F-statistic:         3522.
Date:                   Thu, 14 Mar 2024    Prob (F-statistic):    0.00
Time:                   14:10:53    Log-Likelihood:       -3729.0
No. Observations:       10683    AIC:                  7468.
Df Residuals:           10678    BIC:                  7504.
Df Model:                4
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	8.6101	0.050	172.970	0.000	8.513	8.708
surface_covered	0.5917	0.018	32.388	0.000	0.556	0.627
surface_total	0.2158	0.013	17.202	0.000	0.191	0.240
bathrooms	0.1999	0.013	15.497	0.000	0.175	0.225
rooms	-0.0180	0.012	-1.552	0.121	-0.041	0.005

```
=====
```

Omnibus:	10685.160	Durbin-Watson:	1.560
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2838419.205
Skew:	-4.341	Prob(JB):	0.00
Kurtosis:	82.381	Cond. No.	92.4

=====

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x79c53d135930>

Vemos que al realizar una regresion multiple, obtenemos un r^2 de 0.604, mas alta que la obtenida en regresion simple, pero no es un modelo muy bueno

✓ *Arbol de decision*

Probamos ahora con un modelo de arbol de decision, para ver si podemos obtener un mejor modelo que el generado mediante regresion. Lo primero que hacemos es generar un dataframe que contenga solo las variables que nos interesan para las predicciones

```
1 #En este caso uso las variables sin transformar.
2 modelo_venta_dpto_capital = filtro_venta_dpto_capital[['rooms', 'bathrooms', 'surface_total', 'surface_covered', 'price']]

1 # Modelo de venta de departamentos en capital usando un arbol de decision:
2 #spliteo de datos en target y features for test and train
3
4 #seleccion de x values sin el target
5 X = modelo_venta_dpto_capital.drop("price", axis=1)
6
7 #seleccion del target
8 y = modelo_venta_dpto_capital["price"]
9
10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

1 model = tree.DecisionTreeRegressor( random_state = 42) #Creo un modelo de arbol de decision sin ningun hiperparametro espe
2 model.fit(X_train, y_train) #Se entrena el modelo
3 preds_val = model.predict(X_test)
4 r2 = r2_score(y_test,preds_val)
5 mae = mean_absolute_error(y_test, preds_val)
6
7 print('r2:{ } \n mae: { }' .format(r2, mae))
8
```

```
➡ r2:0.5501198273996928
   mae: 37423.519557971566
```

Vemos que el modelo sin setear ningun hiperparametro arroja un r^2 de 0.58 y un MAE (error absoluto medio) de 36141. El MAE me indica que este modelo me da un error de 36141 dolares entre el valor predicho y el valor real. Para ver si se puede mejorar esto, se van a generar varios modelos, variando la profundidad maxima del arbol, y evaluando si esto mejora la precisión del modelo.

```
1 #Genero una funcion que tome como argumento, ademas de los valores x e y de train y test, la profundidad del arbol
2 def r2(max_depth, X_train, X_test, y_train, y_test):
3     model = tree.DecisionTreeRegressor(max_depth=max_depth, random_state=0)
4     model.fit(X_train, y_train)
5     preds_val = model.predict(X_test)
6     r2 = r2_score(y_test,preds_val)
7     return(r2)
8

1 #Genero un loop que itere entre max_Depth 2 y 10, y paso la funcion r2, que me va entrenando
2 #modelos y me arroja los r2 de cada uno
3
4 for max_depth in range(2,11):
5     resr2 = r2(max_depth, X_train, X_test, y_train, y_test)
6     print("Maxima profundidad del arbol: %d \t\t R2: %.2f" %(max_depth, resr2))
```

```
➡ Maxima profundidad del arbol: 2           R2: 0.59
   Maxima profundidad del arbol: 3           R2: 0.62
   Maxima profundidad del arbol: 4           R2: 0.64
   Maxima profundidad del arbol: 5           R2: 0.65
   Maxima profundidad del arbol: 6           R2: 0.64
   Maxima profundidad del arbol: 7           R2: 0.64
   Maxima profundidad del arbol: 8           R2: 0.63
   Maxima profundidad del arbol: 9           R2: 0.62
   Maxima profundidad del arbol: 10          R2: 0.61
```