

Análisis inmobiliario Buenos Aires

Indice

- 1 Descripción
- 2 Importación de librerías y paquetes
- 3 Carga de datos
- 4 Inspección inicial
- 5 Limpieza de datos
 - 5.1 Duplicados
 - 5.2 Ubicación geográfica
 - 5.2.1 País
 - 5.2.2 Ciudad
 - 5.2.3 Barrio
 - 5.2.4 Coordenadas geográficas
 - 5.3 Características del inmueble
 - 5.3.1 Tipo de propiedad
 - 5.3.2 Tipo de operación
 - 5.3.3 Superficie cubierta
 - 5.3.4 Rooms
 - 5.3.5 Bathrooms
 - 5.3.6 Bedrooms
 - 5.3.7 Superficie Total
 - 5.3.8 Currency
 - 5.3.9 Precio
 - 5.3.10 Limpieza de columnas
- 6 Visualizaciones
 - 6.1 Mapa de precios
 - 6.2 Ubicación de los inmuebles
 - 6.3 Cantidad de inmuebles en venta
 - 6.4 Cantidad de inmuebles ponderado por superficie
- 7 Regresión lineal
 - 7.1 Procesamiento de datos
 - 7.1.1 Variables categóricas
 - 7.1.2 Estandarización de variables numéricas
 - 7.1.3 Creación de sets de train y test
 - 7.2 Train del modelo
 - 7.3 Test del modelo
 - 7.3.1 Puntaje: R2
- 8 Tasador de propiedad

1 - Descripción

Utilizando datos de Properati de septiembre 2021, haremos un análisis del mercado inmobiliario para residencia de la Ciudad de Buenos Aires.

Se utilizarán medidas de estadística descriptiva, así como también visualizaciones a fin de entender y asegurar la correctitud de la información.

Finalizaremos modelando un algoritmo de regresión lineal que permita predecir el valor por M2 de una propiedad en base a otras de iguales características.

2 - Importación de librerías y paquetes

```
In [1]: import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
import shapely.wkt
import geojson
import geopandas
import rtree
from shapely.ops import cascaded_union, unary_union
```

3 - Carga de datos

```
In [2]: df = pd.read_csv("ar_properties.csv")
```

4 - Inspección inicial

```
In [3]: # Cinco líneas sample
df.sample(5)
```

Out[3]:

		id	ad_type	start_date	end_date	created_on	lat	lon	I1	I2	I3	...	bathrooms	surface_total	surface_covered	price	curren
575808	g9kzGv9kbWHU3a4sQghicQ==	Propiedad	2020-10-10	9999-12-31	2020-10-10	-37.116315	-56.856394	Argentina	Buenos Aires Costa Atlántica	Pinamar	...		1.0	NaN	NaN	111.0	U:
489319	KwrvW2KMNIhfKCN7J8qWLoQ==	Propiedad	2020-09-07	9999-12-31	2020-09-07	-34.720074	-58.395499	Argentina	Bs.As. G.B.A. Zona Sur	Lanús	...		2.0	NaN	NaN	260000.0	U:
455490	Q5MJYsJ9zz+6q7La2qIG9g==	Propiedad	2020-09-05	9999-12-31	2020-09-05	-34.600269	-58.369510	Argentina	Capital Federal	Catalinas	...		2.0	634.0	634.0	6340.0	U:
630796	vYcvL24S/LakShSxLOSHcA==	Propiedad	2020-10-01	9999-12-31	2020-10-01	-34.535444	-58.488952	Argentina	Bs.As. G.B.A. Zona Norte	Vicente López	...		3.0	NaN	340.0	450000.0	U:
44920	0IL1u4yvPgqCUBdonRzxGw==	Propiedad	2020-12-28	2021-01-16	2020-12-28	-34.753117	-58.393504	Argentina	Bs.As. G.B.A. Zona Sur	Lomas de Zamora	...		2.0	NaN	180.0	33000.0	A

5 rows x 25 columns

In [4]: df.describe()

Out[4]:

	lat	lon	l6	rooms	bedrooms	bathrooms	surface_total	surface_covered	price
count	894233.000000	894191.000000	0.0	714179.000000	649933.000000	765122.000000	477831.000000	4.877560e+05	9.582430e+05
mean	-34.377500	-59.491698	NaN	2.916567	2.097815	1.708581	513.760752	1.165622e+04	3.460940e+05
std	3.035987	2.800543	NaN	1.739685	2.079008	1.092221	4297.241604	4.443191e+06	5.713690e+06
min	-54.841484	-122.538399	NaN	1.000000	-16.000000	1.000000	-136.000000	-1.300000e+02	0.000000e+00
25%	-34.723328	-58.876930	NaN	2.000000	1.000000	1.000000	50.000000	4.500000e+01	3.800000e+04
50%	-34.593014	-58.493790	NaN	3.000000	2.000000	1.000000	95.000000	7.800000e+01	9.700000e+04
75%	-34.415841	-58.397472	NaN	4.000000	3.000000	2.000000	250.000000	1.680000e+02	2.100000e+05
max	49.633731	180.000000	NaN	40.000000	900.000000	20.000000	200000.000000	2.147484e+09	3.100000e+09

In [5]:

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 25 columns):
#   Column              Non-Null Count  Dtype  
---  --
0   id                   1000000 non-null object  
1   ad_type              1000000 non-null object  
2   start_date           1000000 non-null object  
3   end_date             1000000 non-null object  
4   created_on           1000000 non-null object  
5   lat                  894233 non-null float64  
6   lon                  894191 non-null float64  
7   l1                   1000000 non-null object  
8   l2                   1000000 non-null object  
9   l3                   965273 non-null object  
10  l4                   306162 non-null object  
11  l5                   5530 non-null  object  
12  l6                   0 non-null     float64  
13  rooms                714179 non-null float64  
14  bedrooms             649933 non-null float64  
15  bathrooms            765122 non-null float64  
16  surface_total        477831 non-null float64  
17  surface_covered      487756 non-null float64  
18  price               958243 non-null float64  
19  currency             955491 non-null object  
20  price_period         429870 non-null object  
21  title                999999 non-null object  
22  description          999958 non-null object  
23  property_type        1000000 non-null object  
24  operation_type       1000000 non-null object  
dtypes: float64(9), object(16)
memory usage: 190.7+ MB
```

5 - Limpieza de datos

Dado que la información proviene de una fuente externa de la que no tenemos mayores referencias, necesitamos verificar la orrectitud de los datos, para ello vamos a examinar las columnas que nos interesan.

5.1 Duplicados

In [6]:

```
df.shape[0]
```

Out[6]:

```
1000000
```

In [7]:

```
df.drop_duplicates(inplace=True)
```

In [8]:

```
df.shape[0]
```

Out[8]:

```
1000000
```

El drop suplicates no arrojó resultados, sin embargo parece que hay algunos inmuebles duplicados en los cuales cambia la feha de creación/finalización y id. Dado que esas columnas no serán de utilidad, las eliminamos y volvemos a limpiar duplicados.

In [9]:

```
df.drop(columns=["id", "ad_type", "start_date", "end_date", "created_on"], inplace=True)
```

In [10]:

```
df.drop_duplicates(inplace=True)
df.shape[0]
```

Out[10]:

```
902883
```

5.2 Ubicación geográfica

In [11]:

```
# Vemos que hay en las columnas 5 a 12
df.iloc[:,5:13]
```

Out[11]:

	l4	l5	l6	rooms	bedrooms	bathrooms	surface_total	surface_covered
0	NaN	NaN	NaN	NaN	NaN	NaN	133139.0	NaN
1	NaN	NaN	NaN	8.0	NaN	NaN	687.0	687.0
2	NaN	NaN	NaN	2.0	1.0	1.0	80.0	80.0
3	NaN	NaN	NaN	2.0	1.0	1.0	NaN	NaN
4	NaN	NaN	NaN	3.0	1.0	1.0	76.0	66.0
...
999995	NaN	NaN	NaN	NaN	9.0	7.0	400.0	300.0
999996	NaN	NaN	NaN	12.0	12.0	5.0	465.0	465.0
999997	NaN	NaN	NaN	14.0	13.0	2.0	615.0	425.0
999998	NaN	NaN	NaN	NaN	20.0	20.0	450.0	450.0
999999	Santa Barbara Barrio Cerrado	NaN	NaN	5.0	40.0	4.0	350.0	300.0

902883 rows x 8 columns

5.2.1 País

In [12]:

```
# Vemos contenido de l1
df.l1.value_counts()
```

Out[12]:

Argentina	886520
Uruguay	15247
Estados Unidos	873

```
Brasil                243
Name: l1, dtype: int64
```

```
In [13]: df.loc[df["l1"] == "Uruguay", "l2"].value_counts(dropna=False)
```

```
Out[13]: Maldonado      7621
Montevideo    5604
Colonia       1196
Canelones     706
Rocha         120
Name: l2, dtype: int64
```

```
In [14]: df.loc[df["l1"] == "Estados Unidos", "l2"].value_counts(dropna=False)
```

```
Out[14]: Florida        506
Michigan      257
Miami         69
Maryland      26
Pennsylvania  11
California    4
Name: l2, dtype: int64
```

```
In [15]: df.loc[df["l1"] == "Brasil", "l2"].value_counts(dropna=False)
```

```
Out[15]: São Paulo      141
Rio Grande do Norte    47
Santa Catarina         40
Rio de Janeiro         15
Name: l2, dtype: int64
```

```
In [16]: df.shape
```

```
Out[16]: (902883, 20)
```

```
In [17]: # Parece ser que efectivamente estas líneas corresponden a NO Argentina, las eliminamos
dropin = df.loc[df["l1"] != "Argentina",:].index
df.drop(index=dropin, inplace=True)
```

```
In [18]: df.shape
```

```
Out[18]: (886520, 20)
```

5.2.2 Ciudad

```
In [19]: # Parece que l2 corresponde a la ciudad
df.l2.value_counts()
```

```
Out[19]: Capital Federal      230901
Bs.As. G.B.A. Zona Norte    168830
Bs.As. G.B.A. Zona Sur      104864
Santa Fe                     88882
Bs.As. G.B.A. Zona Oeste     74018
Buenos Aires Costa Atlántica  70675
Córdoba                     59449
Buenos Aires Interior        22815
Neuquén                     14233
Mendoza                     8975
Río Negro                   7224
Entre Ríos                  6476
Tucumán                    6268
Salta                      5657
Misiones                   4287
San Luis                   2584
Chaco                      1633
La Pampa                   1621
Corrientes                 1560
Chubut                     1499
San Juan                   1364
Jujuy                      680
Santa Cruz                 445
Tierra Del Fuego           438
Catamarca                  405
La Rioja                   392
Santiago Del Estero         316
Formosa                     29
Name: l2, dtype: int64
```

```
In [20]: # Al igual que con los países, nos quedamos con las líneas de "Capital Federal"
dropin = df.loc[df["l2"] != "Capital Federal",:].index
df.drop(index=dropin, inplace=True)
```

```
In [21]: df.shape
```

```
Out[21]: (230901, 20)
```

```
In [22]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 230901 entries, 16 to 999997
Data columns (total 20 columns):
#   Column          Non-Null Count  Dtype
---  -
0   lat              216277 non-null  float64
1   lon              216277 non-null  float64
2   l1               230901 non-null  object
3   l2               230901 non-null  object
4   l3               228586 non-null  object
5   l4               10709 non-null   object
6   l5               0 non-null       object
7   l6               0 non-null       float64
8   rooms            187399 non-null  float64
9   bedrooms         156766 non-null  float64
10  bathrooms         200677 non-null  float64
11  surface_total     162867 non-null  float64
12  surface_covered   162059 non-null  float64
13  price             226664 non-null  float64
14  currency          226287 non-null  object
15  price_period      114818 non-null  object
16  title             230901 non-null  object
17  description        230900 non-null  object
18  property_type      230901 non-null  object
19  operation_type     230901 non-null  object
dtypes: float64(9), object(11)
memory usage: 37.0+ MB
```

```
In [23]: # Drop l5 y l6 que no tienen datos
df.drop(columns=["l5", "l6"], inplace=True)
```

```
In [24]: df.reset_index(inplace=True,drop=True)
```

5.2.3 Barrio

```
In [25]: # Vemos que datos encontramos en l3
df.l3.value_counts(dropna=False)
```

```
Out[25]: Palermo      35873
Belgrano      20566
Recoleta      15974
Caballito     13621
Almagro       9326
Villa Urquiza  8908
Villa Crespo  8267
Barrio Norte  7926
Nuñez         6767
Flores        6191
San Nicolás   6164
Puerto Madero 6025
Balvanera     5761
Retiro        4344
San Telmo     4145
Colegiales    4117
Villa Devoto  3893
San Cristobal 3718
Montserrat    3610
Saavedra      3417
Villa del Parque 3257
Boedo         2885
Barracas      2883
Mataderos     2752
Floresta      2656
Liniers       2452
NaN           2315
Congreso      2115
Parque Chacabuco 2092
Villa Pueyrredón 1972
Villa Luro     1948
Centro / Microcentro 1778
Parque Patricios 1759
Chacarita     1687
Once          1551
Coghlan       1545
Villa Lugano   1442
Constitución  1359
Paternal      1359
Tribunales    1327
Villa Ortuzar  1152
Boca          1048
Las Cañitas   971
Monte Castro  892
Villa General Mitre 829
Villa Santa Rita 759
Abasto        752
Pompeya       702
Parque Avellaneda 689
Velez Sarsfield 536
Versalles     519
Parque Centenario 504
Agronomía     473
Parque Chas   469
Catalinas     307
Villa Real    270
Villa Soldati 228
Villa Riachuelo 54
Name: l3, dtype: int64
```

```
In [26]: # Nulos en l3 "salvables"
df.loc[pd.isna(df.l3),"l4"].value_counts(dropna=False)
```

```
Out[26]: NaN      2315
Name: l4, dtype: int64
```

```
In [27]: df.loc[pd.isna(df.l3),"l3"].value_counts(dropna=False)
```

```
Out[27]: NaN      2315
Name: l3, dtype: int64
```

```
In [28]: # No hay salvables, drop de todos los nan en l3
dropin = df.loc[pd.isna(df.l3),:].index
df.drop(index=dropin,inplace=True)
```

```
In [29]: df.l3.value_counts(dropna=False)
```

```
Out[29]: Palermo      35873
Belgrano      20566
Recoleta      15974
Caballito     13621
Almagro       9326
Villa Urquiza  8908
Villa Crespo  8267
Barrio Norte  7926
Nuñez         6767
Flores        6191
San Nicolás   6164
Puerto Madero 6025
Balvanera     5761
Retiro        4344
San Telmo     4145
Colegiales    4117
Villa Devoto  3893
San Cristobal 3718
Montserrat    3610
Saavedra      3417
Villa del Parque 3257
Boedo         2885
Barracas      2883
Mataderos     2752
Floresta      2656
Liniers       2452
Congreso      2115
Parque Chacabuco 2092
Villa Pueyrredón 1972
Villa Luro     1948
Centro / Microcentro 1778
Parque Patricios 1759
Chacarita     1687
Once          1551
Coghlan       1545
Villa Lugano   1442
Constitución  1359
Paternal      1359
Tribunales    1327
```

Villa Ortuzar	1152
Boca	1048
Las Cañitas	971
Monte Castro	892
Villa General Mitre	829
Villa Santa Rita	759
Abasto	752
Pompeya	702
Parque Avellaneda	689
Velez Sarsfield	536
Versalles	519
Parque Centenario	504
Agronomía	473
Parque Chas	469
Catalinas	307
Villa Real	270
Villa Soldati	228
Villa Riachuelo	54

Name: l3, dtype: int64

```
In [30]: df.reset_index(inplace=True,drop=True)
```

```
In [31]: # Lista de barrios
barrios_prop = list(df.l3.unique())
```

```
In [32]: # importamos datos de barrios provistos por GCBA
b_df = pd.read_csv("barrios.csv")
b_df.head()
```

```
Out[32]:
```

	WKT	barrio	comuna	perimetro	area
0	POLYGON ((-58.4528200492791 -34.5959886570639,...	CHACARITA	15	7724.852955	3.115707e+06
1	POLYGON ((-58.4655768128541 -34.5965577078058,...	PATERNAL	15	7087.513295	2.229829e+06
2	POLYGON ((-58.4237529813037 -34.5978273383243,...	VILLA CRESPO	15	8131.857075	3.615978e+06
3	POLYGON ((-58.4946097568899 -34.6148652395239,...	VILLA DEL PARQUE	11	7705.389797	3.399596e+06
4	POLYGON ((-58.4128700313089 -34.6141162515854,...	ALMAGRO	5	8537.901368	4.050752e+06

```
In [33]: barrios = b_df.barrio.unique()
```

```
In [34]: # Vemos que barrios están en el dataset y no considerados por el GCBA
l = []
for x in barrios_prop:
    if x.upper() not in barrios:
        l.append(x)
l
```

```
Out[34]: ['San Nicolás',
'Barrio Norte',
'Constitución',
'Catalinas',
'Tribunales',
'Once',
'Villa Pueyrredón',
'Congreso',
'Centro / Microcentro',
'Villa General Mitre',
'Abasto',
'Las Cañitas',
'Agronomía',
'Parque Centenario',
'Pompeya']
```

```
In [35]: # Armos dict para cambiar los barrios que faltan
change_b = {
    'San Nicolás': 'SAN NICOLAS',
    'Barrio Norte': 'RECOLETA',
    'Constitución': 'CONSTITUCION',
    'Catalinas': 'RETIRO',
    'Tribunales': 'SAN NICOLAS',
    'Once': 'BALVANERA',
    'Villa Pueyrredón': 'VILLA PUEYRRREDON',
    'Congreso': 'BALVANERA',
    'Centro / Microcentro': 'SAN NICOLAS',
    'Villa General Mitre': 'VILLA GRAL. MITRE',
    'Abasto': 'BALVANERA',
    'Agronomía': 'AGRONOMIA',
    'Parque Centenario': 'CABALLITO',
    'Pompeya': 'NUEVA POMPEYA',
    'Las Cañitas': 'PALERMO',
}
```

Agregamos a l4 los sub-barrios que tenemos en l3, antes de cambiarlos por los datos del GCBA

```
In [36]: df.l4.fillna(df.l3,inplace=True)
```

```
In [37]: df["l3"].replace(change_b,inplace=True)
```

```
In [38]: df["l3"] = df["l3"].str.upper()
```

```
In [39]: # Verificamos que no queden barrios por cambiar
barrios_prop = list(df.l3.unique())
l = []
for x in barrios_prop:
    if x.upper() not in barrios:
        l.append(x)
l
```

```
Out[39]: []
```

```
In [40]: df.reset_index(inplace=True,drop=True)
```

5.2.4 Coordenadas geográficas

```
In [41]: # Vemos los nulos de lat lon para ver si se pueden imputar
df.loc[pd.isna(df["lat"]), "l3"].value_counts()
```

```
Out[41]: PALERMO          3137
RECOLETA          1981
BELGRANO          1483
ALMAGRO           688
BALVANERA         671
```

```

PUERTO MADERO      615
CABALLITO          560
NUÑEZ              478
SAN NICOLAS        464
SAN TELMO          339
FLORES             332
VILLA URQUIZA      302
SAN CRISTOBAL      293
VILLA CRESPO       289
CONSTITUCION       229
RETIRO             210
MONSERRAT          191
MATADEROS          186
LINIERS            108
VILLA DEL PARQUE   93
COGHLAN            61
VILLA LURO         60
BARRACAS          53
FLORESTA          48
VILLA DEVOTO       46
PATERNAL           41
COLEGIALES         40
SAAVEDRA           39
VILLA LUGANO       36
BOEDO              34
PARQUE CHACABUCO   29
PARQUE PATRICIOS   21
VILLA PUEYRREDON   19
VERSALLES          14
MONTE CASTRO       10
PARQUE AVELLANEDA  9
CHACARITA          9
BOCA               9
PARQUE CHAS        9
NUEVA POMPEYA      7
VILLA GRAL. MITRE  6
VILLA ORTUZAR      5
VILLA SANTA RITA   4
AGRONOMIA           4
VILLA REAL         3
VILLA SOLDATI      1
Name: l3, dtype: int64
```

```
In [42]: # imputación de las lat/lon faltantes con el centroide por barrio
barrios_faltantes = df.loc[pd.isna(df["lat"]), "l3"].unique()
geo_barrios = {}
for b in barrios_faltantes:
    sliced_df = df.loc[df["l3"]==b, ["lat", "lon"]].dropna()
    lat = sliced_df.lat.mean()
    lon = sliced_df.lon.mean()
    geo_barrios[b] = (lat, lon)
```

```
In [43]: # Nulos antes de imputacion
df.lat.isnull().sum()
```

Out[43]: 13266

```
In [44]: for x in range(df.shape[0]):
    if np.isnan(df.iloc[x,0]):
        l3 = df.iloc[x,4]
        df.iloc[x,0] = geo_barrios[l3][0]
    if np.isnan(df.iloc[x,1]):
        l3 = df.iloc[x,4]
        df.iloc[x,1] = geo_barrios[l3][1]
```

```
In [45]: # Nulos luego de la imputacion
df.lat.isnull().sum()
```

Out[45]: 0

5.3 Características del inmueble

```
In [46]: df.describe()
```

	lat	lon	rooms	bedrooms	bathrooms	surface_total	surface_covered	price
count	228586.000000	228586.000000	186337.000000	155679.000000	199230.000000	161213.000000	160499.000000	2.245810e+05
mean	-34.596233	-58.432531	2.593500	1.876059	1.535055	163.021090	151.042524	2.124335e+05
std	0.267708	0.243224	1.499394	1.599289	0.974373	1150.018499	1320.670184	6.836799e+05
min	-38.587804	-100.469651	1.000000	-2.000000	1.000000	10.000000	-130.000000	0.000000e+00
25%	-34.614603	-58.460212	2.000000	1.000000	1.000000	43.000000	40.000000	4.500000e+04
50%	-34.597070	-58.431503	2.000000	2.000000	1.000000	66.000000	59.000000	1.100000e+05
75%	-34.580004	-58.401219	3.000000	2.000000	2.000000	120.000000	104.000000	2.100000e+05
max	38.052475	-5.490771	40.000000	154.000000	20.000000	140380.000000	170000.000000	9.630000e+07

Veamos las características del inmueble: habitaciones, baños, superficie total/cubierta, tipo de propiedad

Buscaremos encontrar datos faltantes utilizando expresiones regulares en el título/descripción del anuncio.

5.3.1 Tipo de propiedad

```
In [47]: # Arrancamos viendo tipos de propiedad anunciadas
df.property_type.value_counts()
```

```
Out[47]: Departamento    165965
PH                        14322
Oficina                  13329
Local comercial          12945
Casa                     8267
Lote                     6292
Cochera                  3747
Otro                     2223
Depósito                 1488
Casa de campo            8
Name: property_type, dtype: int64
```

Dado que estamos interesados en estudiar el mercado inmobiliario residencial, eliminamos los registros de inmuebles comerciales

```
In [48]: # Eliminamos los registros con inmuebles comerciales
dropin = df[df.property_type.isin(["Oficina", "Lote", "Local comercial", "Cochera", "Otro", "Depósito", "Casa de campo"])]
df.drop(index=dropin, inplace=True)
df.reset_index(inplace=True, drop=True)
```

```
In [49]: # Vemos las propiedades que nos quedan
```

```
df.property_type.value_counts()
```

```
Out[49]: Departamento    165965
PH                      14322
Casa                     8267
Name: property_type, dtype: int64
```

5.3.2 Tipo de operación

En este análisis nos vamos a centrar en los inmuebles destinados a la venta.

```
In [50]: df.operation_type.value_counts()
```

```
Out[50]: Venta                133829
Alquiler                    41968
Alquiler temporal          12757
Name: operation_type, dtype: int64
```

```
In [51]: dropin = df.loc[df["operation_type"].isin(["Alquiler", "Alquiler temporal"])].index
df.drop(index=dropin, inplace=True)
```

5.3.3 Superficie cubierta

Una de las métricas más utilizadas en el análisis inmobiliario es el precio (USD) por metro cuadrado. Para ello necesitamos la mayor cantidad de datos de superficie total del inmueble. Vamos a buscar imputar estos datos faltantes utilizando el título y descripción de la publicación.

Primero vamos a utilizar los datos de título y descripción para imputar datos faltantes, utilizando [expresiones regulares](#).

```
In [52]: for index, value in df[df.surface_total.isnull()][["title", "description"]].sample(5).iterrows():
        print(value[0])
        print("-----")
        print(value[1])
        print("_____")
```

DEPARTAMENTO EN VENTA 4 AMBIENTES

DEPARTAMENTO EN VENTA,

4 AMBIENTES, LIVING COMEDOR, COCINA, 3 DORMITORIOS Y UN BAÑO COMPLETO.

PISO 5º;

Ref#500889.

Entrega inmediata - Consulta financiación post posesión

Venta de Departamento 1 AMBIENTE en San Cristobal, Capital Federal.

Monoambiente al frente súper divisible. Al ingresar nos encontramos con la cocina americana con su respectiva barra desayunadora, 4 hornallas eléctricas con horno visor y muebles bajo mesada. Baño completo con bañera. Amplio living en "L" con salida al balcón, y perfectamente divisible para armar el dormitorio. Equipo de Aire acondicionado y pisos de porcelanato.

El edificio es a estrenar, con excelentes detalles de terminación.

Amenities: Sum con parrilla y laundry.

PROYECTO DE EXCELENTES DETALLES DE TERMINACIÓN, IDEAL COMO VIVIENDA Y COMO INVERSIÓN POR ESTAR UBICADO EN UN PUNTO ESTRATÉGICO DE LA CIUDAD. NO DUDES EN CONSULTARNOS!!!.

LEPORE SAN CRISTOBAL. LEPORE PROPIEDADES sancristobal@lepore.com.ar 4941-4100. LEPORE Propiedades S.A. CUIT : 33-60234274-9. C.U.C.I.C.B.A. Matrícula Nº 1139 (Caballito, Villa Crespo, San Cristóbal). AVISO LEGAL: Las descripciones arquitectónicas y funcionales, valores de expensas, impuestos y servicios, fotos y medidas de este inmueble son aproximados. Los datos fueron proporcionados por el propietario y pueden no estar actualizados a la hora de la visualización de este aviso por lo cual pueden arrojar inexactitudes y discordancias con las que surgen de los las facturas, títulos y planos legales del inmueble. El interesado deberá realizar las verificaciones respectivas previamente a la realización de cualquier operación, requiriendo por sí o sus profesionales las copias necesarias de la documentación que corresponda. Venta supeditada al cumplimiento por parte del propietario de los requisitos de la resolución general Nº 2371 de la AFIP (pedido de COTI).

Esta unidad es apta para personas con movilidad reducida.

XINTEL(LEP-LE5-18264)

Departamento en Venta en Balvanera, Capital federal U\$S 77500

VENTA DEPARTAMENTO 1 AMBIENTE BALVANERA

Cómoda unidad ubicada sobre Av. Jujuy a pocos metros de avenidas Belgrano e Independencia respectivamente. Con accesibilidad a líneas de colectivo y subte "H". Constará de gran ambiente con cocina integrada equipada, frente de placard y baño completo. Cocheras disponibles optativas.

LEPORE CENTRO. LEPORE PROPIEDADES centro@lepore.com.ar 4331-3030. LEPORE Propiedades S.A. CUIT : 33-60234274-9. C.U.C.I.C.B.A. Matrícula Nº 1139 (Caballito, Villa Crespo, San Cristóbal). AVISO LEGAL: Las descripciones arquitectónicas y funcionales, valores de expensas, impuestos y servicios, fotos y medidas de este inmueble son aproximados. Los datos fueron proporcionados por el propietario y pueden no estar actualizados a la hora de la visualización de este aviso por lo cual pueden arrojar inexactitudes y discordancias con las que surgen de los las facturas, títulos y planos legales del inmueble. El interesado deberá realizar las verificaciones respectivas previamente a la realización de cualquier operación, requiriendo por sí o sus profesionales las copias necesarias de la documentación que corresponda. Venta supeditada al cumplimiento por parte del propietario de los requisitos de la resolución general Nº 2371 de la AFIP (pedido de COTI).

Esta unidad es apta para personas con movilidad reducida.

XINTEL(LEP-LE2-19513)

CASA LOTE PROPIO - SALIDA A 2 CALLES - TALLER - ETC.-

Venta de Casa 5 AMBIENTES en Mataderos, Capital Federal

CASA EN DOBLE LOTE PROPIO - INGRESO POR 2 CALLES (GUAMINI y AV. EVA PERON) - "LOTE 1" : CASA EN LOTE PROPIO CON 3 DORMITORIOS - BAÑO - COCINA - PATIO - LIVING COMEDOR -

PATIO - TERRAZA TRANSITABLE - EN ESTADO GENERAL BUENO - "LOTE 2" : TALLER 10 x 10 CON BAÑO Y DUCHA - OPORTUNIDAD.-

ACLARACIÓN: LAS MEDIDAS SON APROXIMADAS Y SOLAMENTE A EFECTO ORIENTATIVO - LAS REALES SURGIRAN DE LA ESCRITURA CORRESPONDIENTE.-

LORIA - MATRICULA CUCICBA 1300

XINTEL(LOR-LOR-1381)

Departamento Monoambiente en Venta ubicado en Arenales 1100, Recoleta, Capital Federal

A pasos de la 9 de Julio, departamento apto profesional. Baño completo, kitchenette. Cocina a gas. Muy buen estado.

AySA por expensas. ABL \$322.73

```
In [53]: rex1 = "(\\d+)\\s*m2"
        rex2 = "{superficie}\\s*[total]*: (\\d+)\\s*m"
```

```
In [54]: sliced = df[df.surface_total.isnull()][["title", "description"]]
        rexs = {"r1": rex1, "r2": rex2}
        for x in rexs.keys():
            sliced[str("title" + x)] = sliced.title.str.findall(reats[x], flags=re.IGNORECASE | re.U)
            sliced[str("description" + x)] = sliced.description.str.findall(reats[x], flags=re.IGNORECASE | re.U)
```

```
In [55]: resultados = []
        for x in range(sliced.shape[0]):
            res = []
            for y in range(1, len(reats.keys())*2+1):
                celda = sliced.iloc[x, y]
                for k in range(len(celda)):
                    res.append(celda[k])
            if len(res) > 0:
                resultados.append(res)
            else:
                resultados.append(np.nan)
```

```
In [56]: sliced["resultados"] = resultados
```

```
In [57]: sliced["m2"] = [max([int(i) for i in l]) if type(l)==list else l for index,l in sliced.resultados.iteritems()]
```

Imputación de los M2 conseguidos con los regex

```
In [58]: df.surface_total.fillna(sliced["m2"],inplace=True)
```

Ahpra revisamos la calidad de los datos, previamente importados y conseguidos con las expresiones regulares.

```
In [59]: df[["surface_total", "surface_covered"]].describe().applymap("{0:.2f}".format)
```

```
Out[59]:
```

	surface_total	surface_covered
count	105218.00	97195.00
mean	25719.61	100.18
std	5929223.80	1166.93
min	0.00	1.00
25%	45.00	40.00
50%	66.00	58.00
75%	107.00	92.00
max	1537033122.00	170000.00

Evidentemente hay algun tipo de error en los datos, pues parece haber inmuebles con 15.000M m2 o 159M m2 cubiertos. Vamos a considerar como superficie mínima los 20 m2 y como máxima los 600 m2.

```
In [60]: maximo = 600
minimo = 20
dropin = df[(df["surface_total"]<=minimo) | (df["surface_total"]>=maximo)].index
df.drop(index=dropin,inplace=True)
```

```
In [61]: df[["surface_total", "surface_covered"]].describe()
```

```
Out[61]:
```

	surface_total	surface_covered
count	103228.000000	96154.000000
mean	91.507372	83.813944
std	75.912928	688.660371
min	21.000000	1.000000
25%	45.000000	40.000000
50%	66.000000	58.000000
75%	106.000000	91.000000
max	599.000000	170000.000000

```
In [62]: dropin = df[(df["surface_covered"]<=minimo) | (df["surface_covered"]>=maximo)].index
df.drop(index=dropin,inplace=True)
```

```
In [63]: df.describe().applymap("{0:.2f}".format)
```

```
Out[63]:
```

	lat	lon	rooms	bedrooms	bathrooms	surface_total	surface_covered	price
count	131543.00	131543.00	123614.00	107659.00	124947.00	103041.00	95858.00	129800.00
mean	-34.60	-58.44	2.72	1.96	1.51	91.55	79.43	242001.19
std	0.29	0.27	1.37	1.16	0.84	75.87	64.21	616770.61
min	-38.45	-100.47	1.00	-1.00	1.00	21.00	21.00	0.00
25%	-34.62	-58.47	2.00	1.00	1.00	45.00	40.00	100000.00
50%	-34.60	-58.44	3.00	2.00	1.00	66.00	58.00	150000.00
75%	-34.58	-58.41	3.00	3.00	2.00	106.00	91.00	249000.00
max	29.75	-5.49	35.00	60.00	15.00	599.00	587.00	56939221.00

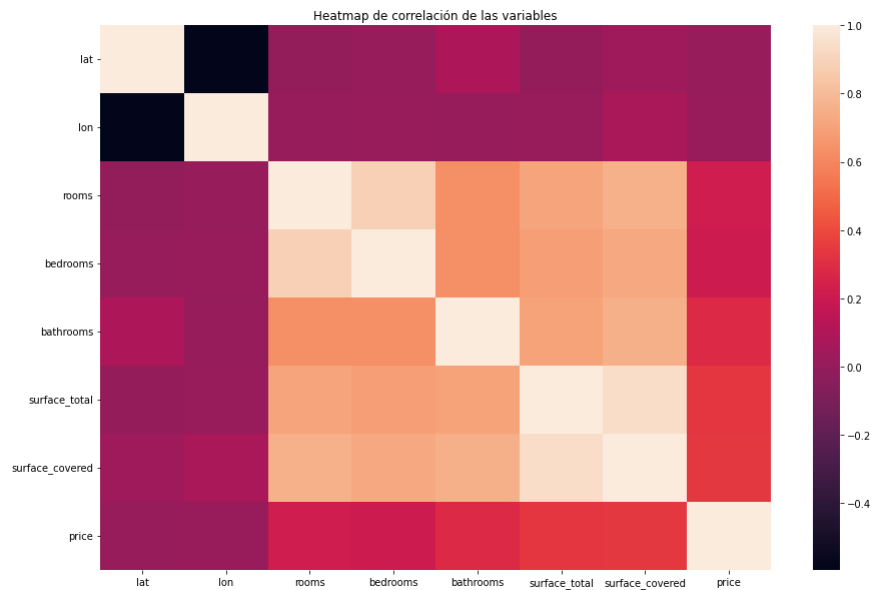
```
In [64]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 131543 entries, 20 to 188551
Data columns (total 18 columns):
#   Column              Non-Null Count  Dtype
---  -
0   lat                  131543 non-null float64
1   lon                  131543 non-null float64
2   ll                    131543 non-null object
3   l2                    131543 non-null object
4   l3                    131543 non-null object
5   l4                    131543 non-null object
6   rooms                123614 non-null float64
7   bedrooms             107659 non-null float64
8   bathrooms            124947 non-null float64
9   surface_total        103041 non-null float64
10  surface_covered      95858 non-null float64
11  price                129800 non-null float64
12  currency             129612 non-null object
13  price_period         58689 non-null object
14  title                131543 non-null object
15  description           131543 non-null object
16  property_type        131543 non-null object
17  operation_type       131543 non-null object
dtypes: float64(8), object(10)
memory usage: 19.1+ MB
```

Los nulos de superficie total siguen siendo la mayoría, buscamos otra forma de imputar esos datos faltantes utilizando otras columnas. Vemos si existe alguna correlación entre la superficie total y otra característica.

```
In [65]: matrix = df.corr()
```

```
In [66]: fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(matrix)
ax.set_title("Heatmap de correlación de las variables")
plt.show()
```

Parece haber una correlación ente la cantidad de ambientes ("rooms") y los superficie cubierta del inmueble. Vemos los datos de rooms.

```
In [67]: df.rooms.value_counts()
```

```
Out[67]:
```

2.0	34925
3.0	34485
1.0	23506
4.0	20687
5.0	6338
6.0	2072
7.0	895
8.0	347
9.0	131
10.0	115
11.0	34
12.0	31
14.0	15
15.0	9
16.0	4
18.0	4
20.0	3
13.0	3
21.0	3
30.0	2
19.0	2
25.0	1
35.0	1
22.0	1

```
Name: rooms, dtype: int64
```


Eliminamos todos los inmuebles con rooms ≥ 6

```
In [68]: dropin = df[df["rooms"]>=6].index
df.drop(index=dropin, inplace=True)
```

```
In [69]: s = df.rooms.value_counts()
sns.barplot(s.index, s.values)
```

```
Users/yagopajarino/jupyter/.venv/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be "data", and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:~>
```

Out [69]: `<AxesSubplot:~>`



Category	Count
1.0	23500
2.0	35000
3.0	34500
4.0	20500
5.0	6000

```
In [70]: df.groupby("rooms").surface_covered.mean().sort_values()
```

```
Out[70]:
```

	rooms
1.0	35.477218
2.0	47.407259
3.0	73.928104
4.0	119.108922
5.0	181.817112

Name: surface covered, dtype: float64

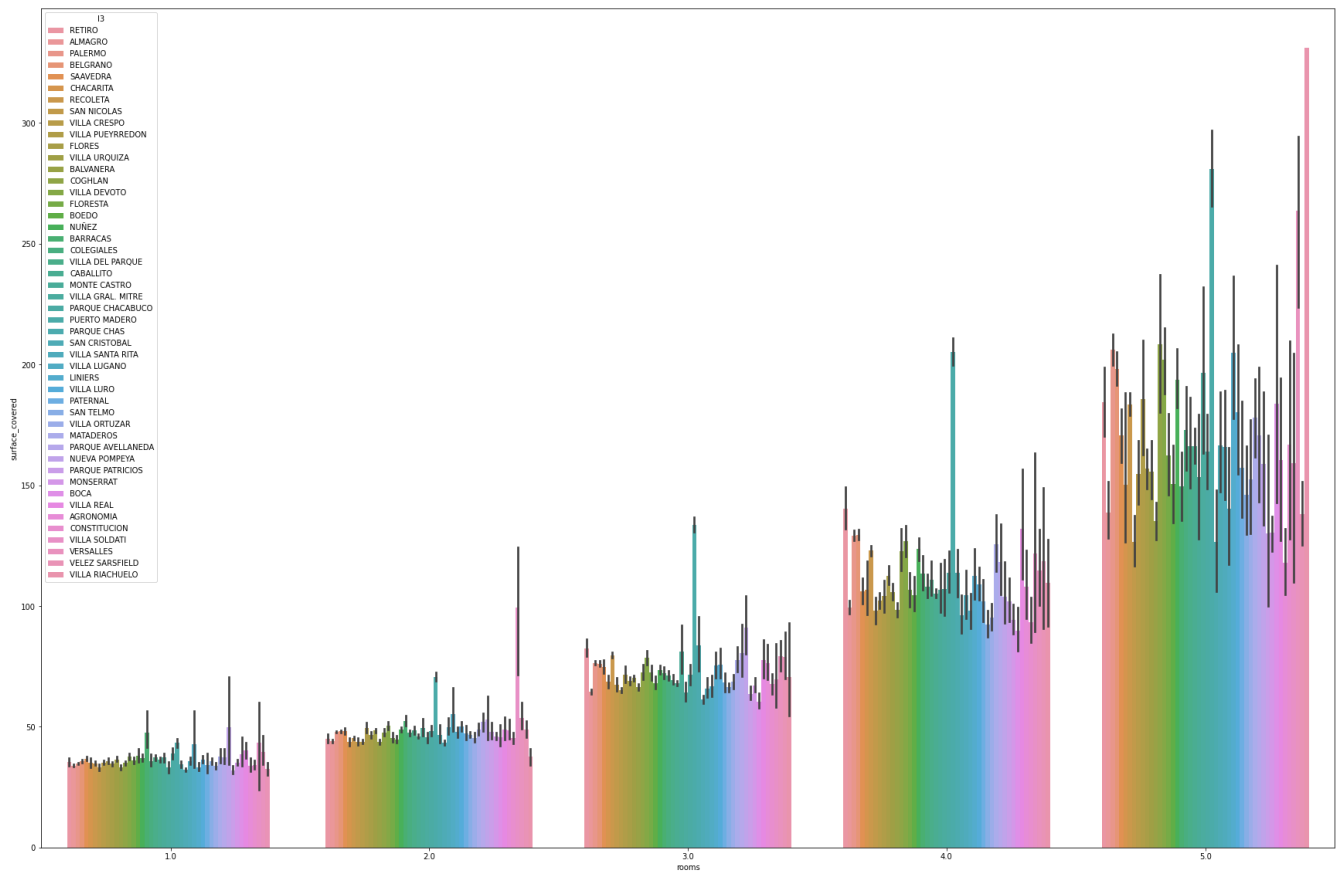
```
In [71]: fig, ax = plt.subplots(figsize=(30,20))
sns.barplot("rooms", "surface covered", "13", data=df, orient="v")
```

```

/Users/yagopajarino/Jupyter/.venv/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y, hue. From vers
ion 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(

```

```
Out[71]: <AxesSubplot:xlabel='rooms', ylabel='surface_covered'>
```



Vemos el caso de Villa Soldati en dos ambientes.

```
In [72]: df[df.rooms == 2].surface_covered.mean()
```

```
Out[72]: 47.40725944526881
```

```
In [73]: df.loc[(df.rooms == 2) & (df.l3 == "VILLA SOLDATI"), "surface_covered"].mean()
```

```
Out[73]: 99.5
```

Parecen estar todos cerca de la media de m2 por barrio. Vamos a imputar los datos de superficie faltantes con la media de superficie ponderada por la cantidad de ambientes.

```
In [74]: superficieMediaPorRooms = df.groupby("rooms").surface_covered.mean()
```

```
In [75]: df["temp_superficie"] = df.rooms.replace(superficieMediaPorRooms)
```

```
In [76]: df.surface_covered.fillna(df["temp_superficie"], inplace=True)
```

```
In [77]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 127870 entries, 20 to 188548
Data columns (total 19 columns):
#   Column                Non-Null Count  Dtype
---  -
0   lat                   127870 non-null float64
1   lon                   127870 non-null float64
2   l1                    127870 non-null object
3   l2                    127870 non-null object
4   l3                    127870 non-null object
5   l4                    127870 non-null object
6   rooms                 119941 non-null float64
7   bedrooms             104110 non-null float64
8   bathrooms            121384 non-null float64
9   surface_total         99914 non-null float64
10  surface_covered       123331 non-null float64
11  price                 126227 non-null float64
12  currency              126045 non-null object
13  price_period          56677 non-null object
14  title                 127870 non-null object
15  description            127870 non-null object
16  property_type          127870 non-null object
17  operation_type         127870 non-null object
18  temp_superficie       119941 non-null float64
dtypes: float64(9), object(10)
memory usage: 23.5+ MB
```

5.3.4 Rooms

Imputación de rooms usando completando con la misma cantidad de ambientes de la observación con m2 más cercana.

```
In [78]: df = df.sort_values("surface_covered")
```

```
In [79]: df.rooms.fillna(method="ffill", inplace=True)
```

```
In [80]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 127870 entries, 6820 to 187907
Data columns (total 19 columns):
#   Column                Non-Null Count  Dtype
---  -
0   lat                   127870 non-null float64
```

```

1 lon 127870 non-null float64
2 ll 127870 non-null object
3 l2 127870 non-null object
4 l3 127870 non-null object
5 l4 127870 non-null object
6 rooms 127870 non-null float64
7 bedrooms 104110 non-null float64
8 bathrooms 121384 non-null float64
9 surface_total 99914 non-null float64
10 surface_covered 123331 non-null float64
11 price 126227 non-null float64
12 currency 126045 non-null object
13 price_period 56677 non-null object
14 title 127870 non-null object
15 description 127870 non-null object
16 property_type 127870 non-null object
17 operation_type 127870 non-null object
18 temp_superficie 119941 non-null float64
dtypes: float64(9), object(10)
memory usage: 19.5+ MB

```

```
In [81]: df.reset_index(inplace=True,drop=True)
```

5.3.5 Bathrooms

Primero vemos los datos que tenemos.

```
In [82]: df.bathrooms.value_counts()
```

```
Out[82]: 1.0    80568
2.0    29765
3.0     7952
4.0     2354
5.0      614
6.0       97
7.0       20
8.0        10
9.0         2
10.0        1
12.0        1
Name: bathrooms, dtype: int64
```

Eliminamos todos los registros con baños ≥ 5

```
In [83]: dropin = df.loc[df["bathrooms"]>= 5].index
df.drop(index=dropin,inplace=True)
```

Luego imputamos los baños considerando 1 y 2 ambientes con un baño y el resto con dos baños.

```
In [84]: baños = {1:1,2:1,3:2,4:2,5:2}
df["temp_baños"] = df.rooms.replace(baños)
df.bathrooms.fillna(df["temp_baños"],inplace=True)
```

```
In [85]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 127125 entries, 0 to 127869
Data columns (total 20 columns):
#   Column              Non-Null Count  Dtype
---  -
0   lat                 127125 non-null float64
1   lon                 127125 non-null float64
2   ll                  127125 non-null object
3   l2                  127125 non-null object
4   l3                  127125 non-null object
5   l4                  127125 non-null object
6   rooms               127125 non-null float64
7   bedrooms            103425 non-null float64
8   bathrooms           127125 non-null float64
9   surface_total       99290 non-null float64
10  surface_covered     122615 non-null float64
11  price               125557 non-null float64
12  currency            125377 non-null object
13  price_period        56209 non-null object
14  title               127125 non-null object
15  description         127125 non-null object
16  property_type       127125 non-null object
17  operation_type      127125 non-null object
18  temp_superficie     119307 non-null float64
19  temp_baños          127125 non-null float64
dtypes: float64(10), object(10)
memory usage: 20.4+ MB

```

5.3.6 Bedrooms

Imputación con la moda por rooms.

```
In [86]: df.bedrooms.value_counts()
```

```
Out[86]: 1.0    39915
2.0    34122
3.0    20864
4.0     4521
0.0    3543
5.0     274
6.0     116
7.0      25
8.0      15
10.0      6
11.0      5
20.0      3
13.0      2
9.0       2
17.0      2
12.0      1
-1.0      1
15.0      1
22.0      1
21.0      1
32.0      1
16.0      1
60.0      1
30.0      1
19.0      1
Name: bedrooms, dtype: int64
```

Eliminamos los registros con bedrooms ≥ 5

```
In [87]: dropin = df.loc[df["bedrooms"]>= 5].index
df.drop(index=dropin,inplace=True)
```

Drop de los que tienen rooms = -1

```
In [88]: dropin = df.loc[df["bedrooms"]== -1].index
df.drop(index=dropin,inplace=True)
```

Estudiamos los que tienen rooms = 0

```
In [89]: df.loc[df["bedrooms"]==0,"rooms"].value_counts()
```

```
Out[89]: 1.0    3515
2.0     14
4.0      6
3.0      5
5.0      3
Name: rooms, dtype: int64
```

Dropeamos los departamentos que más de un ambiente que no tengan dormitorios.

```
In [90]: dropin = df.loc[(df["bedrooms"]==0) & (df["rooms">>1)].index
df.drop(index=dropin,inplace=True)
```

```
In [91]: df.bedrooms.value_counts()
```

```
Out[91]: 1.0    39915
2.0    34122
3.0    20864
4.0     4521
0.0     3515
Name: bedrooms, dtype: int64
```

```
In [92]: grouped = df.groupby("rooms").bedrooms.value_counts()
```

```
In [93]: grouped
```

```
Out[93]: rooms  bedrooms
1.0      1.0      7913
        0.0     3515
        2.0      110
        3.0       54
        4.0       15
2.0      1.0    30540
        2.0      566
        3.0       17
        4.0        4
3.0      2.0    30957
        3.0      682
        1.0      560
        4.0      107
4.0      3.0    17542
        2.0    1454
        4.0      682
        1.0       38
5.0      4.0     3713
        3.0     2569
        2.0     1035
        1.0       864
Name: bedrooms, dtype: int64
```

```
In [94]: modas = {}
for x in range(1,6):
    modas[x] = grouped[x].index[0]
modas
```

```
Out[94]: {1: 1.0, 2: 1.0, 3: 2.0, 4: 3.0, 5: 4.0}
```

Imputación de bedrooms usando las modas calculadas.

```
In [95]: df["temp_bedrooms"] = df.rooms.replace(modas)
df.bedrooms.fillna(df["temp_bedrooms"],inplace=True)
```

5.3.7 Superficie total

Imputación de superficie total con el porcentual de cubierta/total de los datos válidos.

```
In [96]: df[["surface_total","surface_covered"]].describe()
```

```
Out[96]:
```

	surface_total	surface_covered
count	98890.000000	122176.000000
mean	83.546951	71.308721
std	62.002810	47.219815
min	21.000000	21.000000
25%	45.000000	40.000000
50%	64.000000	56.000000
75%	100.000000	82.000000
max	599.000000	587.000000

```
In [97]: datosTotal = df.surface_total.dropna()
datosCubiertos = df.surface_covered.dropna()
indexes = list(set(datosTotal.index).intersection(set(datosCubiertos.index)))
len(indexes)
```

```
Out[97]: 97534
```

```
In [98]: datosTotal = datosTotal.loc[indexes]
datosCubiertos = datosCubiertos.loc[indexes]
porcentaje = datosCubiertos.sum()/datosTotal.sum()
```

```
In [99]: porcentaje
```

```
Out[99]: 0.8605819802880862
```

Con este 87.29% vamos a imputar los datos de superficie total.

```
In [100]: df.surface_total.fillna(df.surface_covered / porcentaje, inplace=True)
df.surface_covered.fillna(df.surface_total * porcentaje, inplace=True)
```

```
In [101]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 126637 entries, 0 to 127869
Data columns (total 21 columns):
#   Column                Non-Null Count  Dtype
---  -
0   lat                    126637 non-null float64
1   lon                    126637 non-null float64
2   ll                     126637 non-null object
3   l2                     126637 non-null object
4   l3                     126637 non-null object
5   l4                     126637 non-null object
6   rooms                  126637 non-null float64
7   bedrooms               126637 non-null float64
8   bathrooms              126637 non-null float64
9   surface_total          123532 non-null float64
10  surface_covered        123532 non-null float64
11  price                  125095 non-null float64
12  currency               124915 non-null object
13  price_period           55871 non-null object
14  title                  126637 non-null object
15  description             126637 non-null object
16  property_type          126637 non-null object
17  operation_type         126637 non-null object
18  temp_superficie        119037 non-null float64
19  temp_baños             126637 non-null float64
20  temp_bedrooms          126637 non-null float64
dtypes: float64(11), object(10)
memory usage: 21.3+ MB

Drop de líneas que quedan sin superficie.
```

```
In [102... dropin = df.loc[df["surface_total"].isna()].index
df.drop(index=dropin,inplace=True)
```

5.3.8 Currency

Vemos los datos de currency nulos.

```
In [103... df.loc[df["currency"].isna()].describe()
```

```
Out[103...
      lat      lon      rooms  bedrooms  bathrooms  surface_total  surface_covered  price  temp_superficie  temp_baños  temp_bedrooms
count  1644.000000  1644.000000  1644.000000  1644.000000  1644.000000  1644.000000  1644.000000  173.0  1559.000000  1644.000000  1644.000000
mean   -34.594898  -58.433328   2.723236   1.865572   1.602798   99.639507   87.348425   0.0    75.620675   1.552311   1.908151
std     0.025399   0.037583   1.202825   0.931393   0.860164   80.560901   69.668565   0.0    40.298172   0.497407   0.979750
min    -34.695465  -58.528299   1.000000   0.000000   1.000000   21.000000   21.000000   0.0    35.477218   1.000000   1.000000
25%    -34.613220  -58.457105   2.000000   1.000000   1.000000   47.000000   43.000000   0.0    47.407259   1.000000   1.000000
50%    -34.592892  -58.434608   3.000000   2.000000   1.000000   73.500000   64.500000   0.0    73.928104   2.000000   2.000000
75%    -34.577712  -58.405388   4.000000   3.000000   2.000000  125.000000  110.000000   0.0   119.108922   2.000000   3.000000
max    -34.539067  -58.356899   5.000000   4.000000   4.000000  676.286529  582.000000   0.0   181.817112   2.000000   4.000000
```

Son todos aquellos que tienen el precio igual a cero. Los dropeamos.

```
In [104... dropin = df.loc[df["currency"].isna()].index
df.drop(index=dropin,inplace=True)
```

```
In [105... df.currency.value_counts()
```

```
Out[105...
USD      121760
ARS       128
Name: currency, dtype: int64
```

```
In [106... dropin = df.loc[df["currency"]=="ARS"].index
df.drop(index=dropin,inplace=True)
```

5.3.9 Precio

No vamos a imputar precios, por lo que dropeamos los datos faltantes.

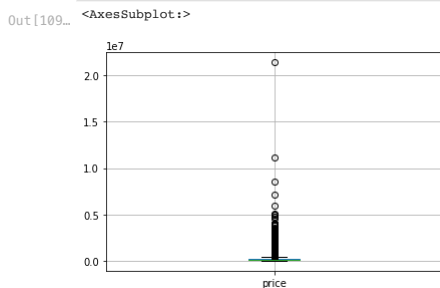
```
In [107... dropin = df.loc[df["price"].isna()].index
df.drop(index=dropin,inplace=True)
```

Vemos los datos de precio

```
In [108... df.price.describe().apply(lambda x: "{0:.2f}".format(x))
```

```
Out[108...
count      121760.00
mean       208901.63
std        232045.42
min           0.00
25%         99500.00
50%        148000.00
75%        235000.00
max       21400000.00
Name: price, dtype: object
```

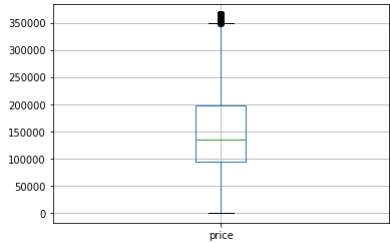
```
In [109... df.boxplot("price")
```



```
In [110... whiskerB = df.price.quantile(0.75) + (df.price.quantile(0.75)-df.price.quantile(0.5))*1.5
dropin = df.loc[df["price"]>whiskerB].index
df.drop(index=dropin,inplace=True)
```

```
In [111... df.boxplot("price")
```

Out[111]: <AxesSubplot:~>



In [112]: df.sort_values("price").head()

	lat	lon	I1	I2	I3	I4	rooms	bedrooms	bathrooms	surface_total	...	price	currency	price_period	title	description	property_type	op
15762	-34.638337	-58.504331	Argentina	Capital Federal	CABALLITO	Caballito	1.0	3.0	1.0	300.000000	...	0.0	USD	Mensual	Casa en venta de 3 dormitorios c/ cochera en B...	NO OFERTAR - Impecable departamento sobre la c...	Departamento	
118397	-34.569801	-58.430868	Argentina	Capital Federal	PALERMO	Las Cañitas	5.0	2.0	2.0	211.272274	...	11.0	USD	NaN	Chenaut y el Polo LAS CAÑITAS A ESTRENAR VISTA...	Chenaut y el Polo LAS CAÑITAS A ESTRENAR VI...	Departamento	
119504	-34.610046	-58.361382	Argentina	Capital Federal	PUERTO MADERO	Puerto Madero	4.0	3.0	4.0	194.000000	...	5000.0	USD	Mensual	Departamento - Puerto Madero	3 SUITES AMOBLADAS EN RENJOIR 2 TIPOLOGÍ...	Departamento	
123090	-34.604666	-58.364059	Argentina	Capital Federal	PUERTO MADERO	Puerto Madero	4.0	3.0	4.0	435.000000	...	6000.0	USD	NaN	DEPTO - 435 M2 4 AMB - MADERO CENTER - P. MADERO	Corredor Responsable: NATALIN ODOGUARDI - CUCI...	Departamento	
83042	-34.612121	-58.360313	Argentina	Capital Federal	PUERTO MADERO	Puerto Madero	3.0	2.0	2.0	85.904778	...	6000.0	USD	NaN	Departamento Piso en Venta ubicado en Puerto ...	Preciosa residencia amoblada y equipada, una "...	Departamento	

5 rows × 21 columns

In [113]: dropin = df.loc[df["price"]<12].index
df.drop(index=dropin,inplace=True)

5.3.10 Limpieza de columnas

Drop de columnas que no se van a utilizar en el análisis. Las columnas de title y descripción, como en el caso de la superficie, se pueden utilizar para conocer más datos del inmueble, como amenities, piso, etc. En este caso las eliminamos dado que nos quedamos con los datos que tenemos hasta el momento.

In [114]: cols = ["I1","I2","currency","operation_type","temp_baños","temp_bedrooms","temp_superficie","price_period","title","description"]
df.drop(columns=cols,inplace=True)

In [115]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 108455 entries, 0 to 127864
Data columns (total 11 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   lat                  108455 non-null float64
1   lon                  108455 non-null float64
2   I3                   108455 non-null object
3   I4                   108455 non-null object
4   rooms                108455 non-null float64
5   bedrooms             108455 non-null float64
6   bathrooms            108455 non-null float64
7   surface_total        108455 non-null float64
8   surface_covered      108455 non-null float64
9   price                108455 non-null float64
10  property_type        108455 non-null object
dtypes: float64(8), object(3)
memory usage: 9.9+ MB
```

6 - Visualizaciones

Lo siguiente es entender la composición de los datos y seguir detectando posibles fallas en los mismos.

In [116]: df["precio_M2"] = df.price / df.surface_total

6.1 Mapa de precios por M2

In [117]: def from_wkt(df, wkt_column):
import shapely.wkt
df["coordinates"] = df[wkt_column].apply(shapely.wkt.loads)
gdf = geopandas.GeoDataFrame(df, geometry="coordinates")
return gdf

In [118]: gpd = from_wkt(b_df, "WKT")

In [119]: gpd["precioM2"] = gpd.barrio.replace(df.groupby("I3").precio_M2.mean())

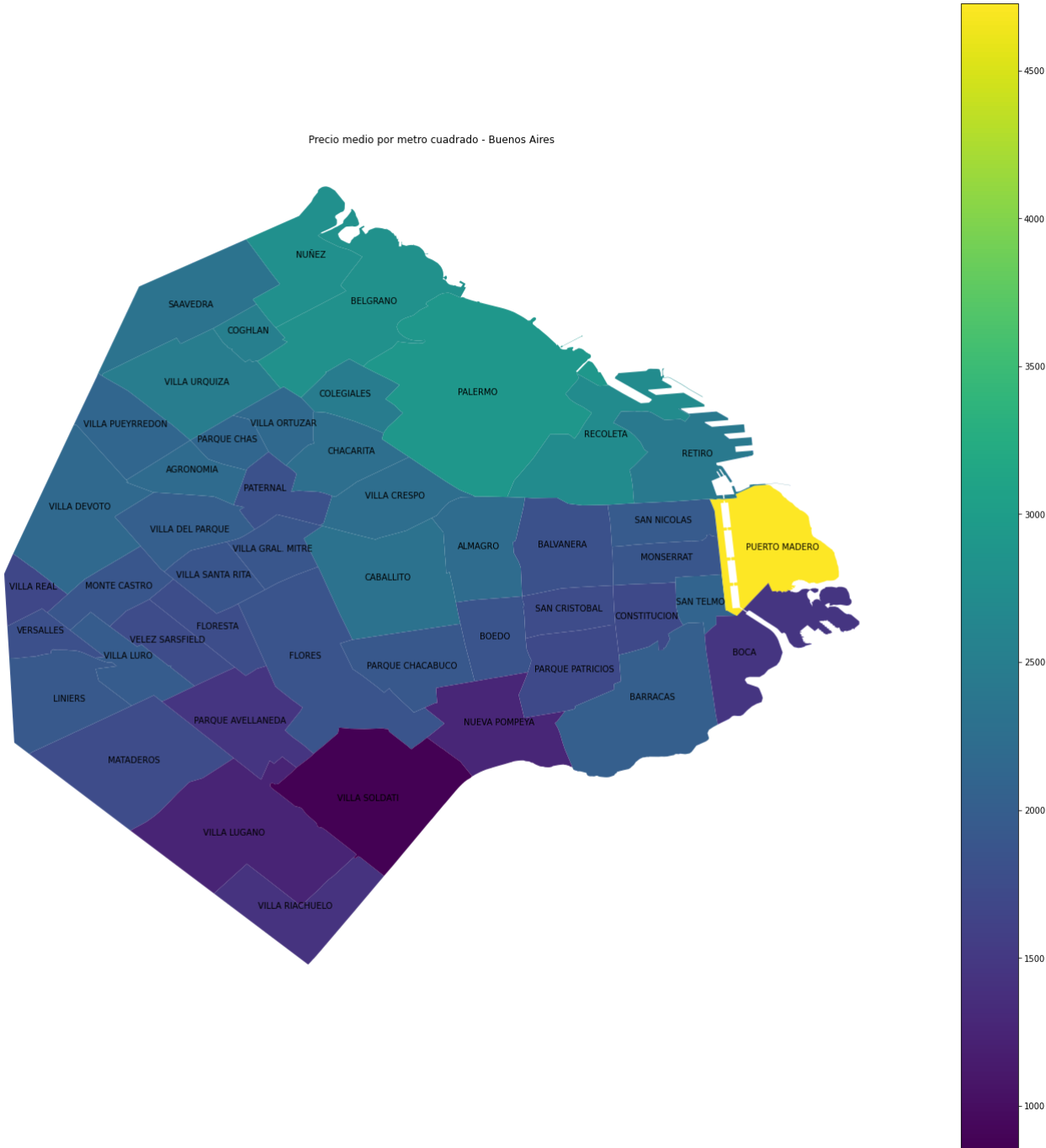
In [120]: gpd['coords'] = gpd['coordinates'].apply(lambda x: x.representative_point().coords[:])
gpd['coords'] = [coords[0] for coords in gpd['coords']]

In [121]: fig, ax = plt.subplots(figsize=(25,25))
gpd.plot("precioM2", legend=True, ax=ax, cmap="viridis")
plt.axis('off')
for idx, row in gpd.iterrows():
ax.annotate(s=row['barrio'], xy=row['coords'],
horizontalalignment='center')
ax.set_title("Precio medio por metro cuadrado - Buenos Aires")
plt.show()

```

/var/folders/E0/53nrwskl5q5fyykx0xbq6840000gn/T/ipykernel_6032/4163819650.py:5: MatplotlibDeprecationWarning: The 's' parameter of annotate() has been renamed 'text' since Matplotlib 3.3; support for the old name will be dropped two minor releases later.
  ax.annotate(s=row['barrio'], xy=row['coords'],

```



Parece que los datos son correctos, con mayores precios por metro cuadrado en el norte de la ciudad de Buenos Aires y máximos de 4500 USD/M2 en el barrio de Puerto Madero.

6.2 Ubicación de los inmuebles

```
In [122... df["point"] = "POINT (" + df.lon.astype(str) + " " + df.lat.astype(str) + ")"
df["point"] = df["point"].apply(shapely.wkt.loads)
```

```
In [123... points = geopandas.GeoDataFrame(df, geometry='point')
```

Parece que hay algunos puntos que no se corresponden con la ciudad de Buenos Aires. Vamos a acotar los datos de lat long.

```
In [124... # Polígono perímetro
polygons = list(gpd.coordinates)
boundary = geopandas.GeoSeries(unary_union(polygons))
```

```
In [125... pol_gpd = geopandas.GeoDataFrame()
pol_gpd['geometry'] = None
pol_gpd.loc[0, 'geometry'] = boundary(0
```

```
In [126... dat_fin = geopandas.sjoin(points, pol_gpd, predicate = 'within')
```

```
In [127... df = pd.DataFrame(dat fin)
```

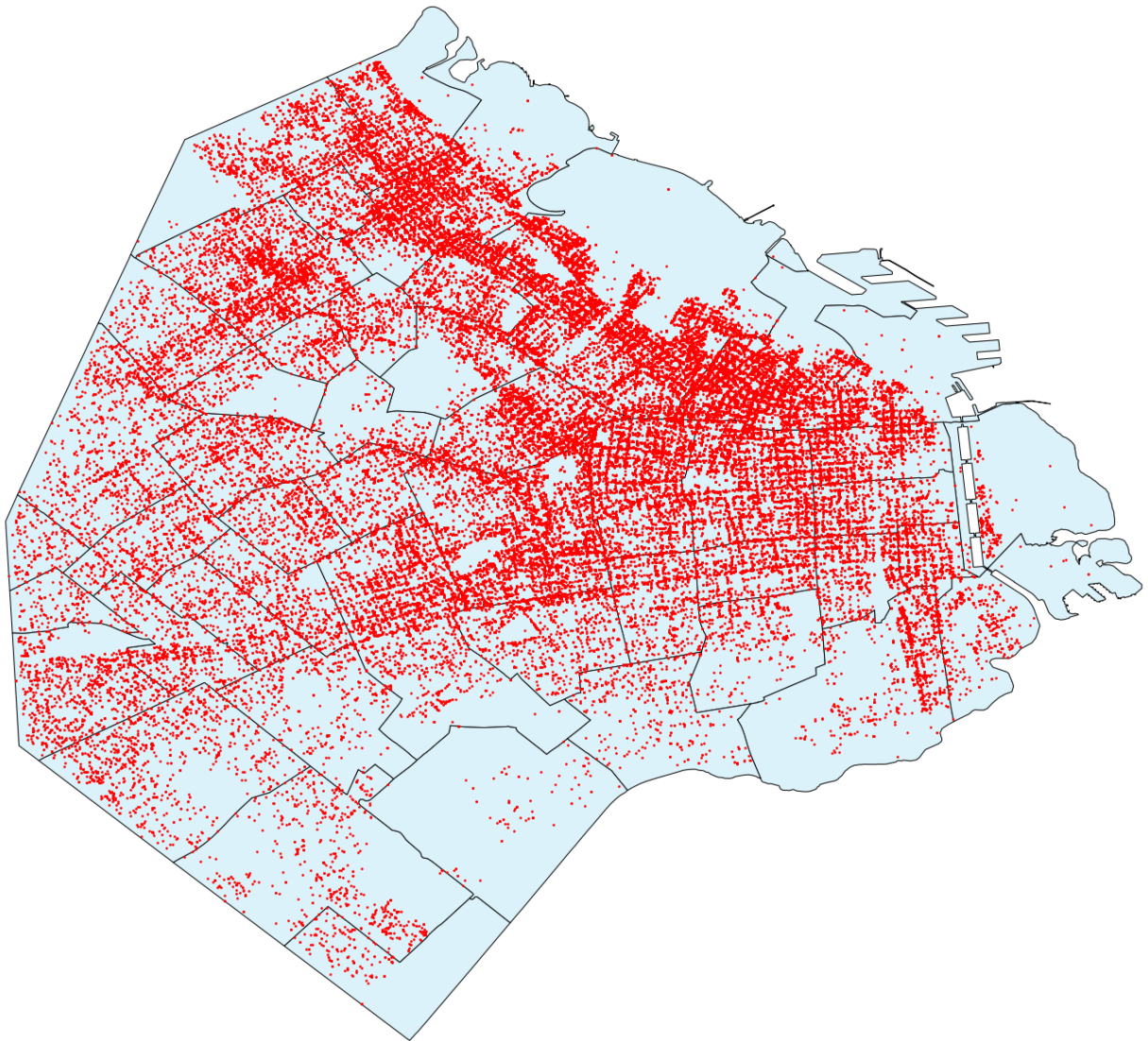
```
In [128... points = geopandas.GeoDataFrame(df.point, geometry="point")
```

```
In [129]: fig, ax = plt.subplots(figsize=(25,25))
ax.set_aspect('equal')
gpd.plot("coords", ax=ax, color="#DCF2FA", edgecolor='black')
points.plot(ax=ax, marker='o', color='red', markersize=3)
ax.set title("Ubicación de inmuebles")
```

```
plt.axis('off')
plt.show()
```

```
/Users/yagopajarino/Jupyter/.venv/lib/python3.9/site-packages/geopandas/plotting.py:644: UserWarning: Only specify one of 'column' or 'color'. Using 'color'.
warnings.warn(
```

Ubicación de inmuebles

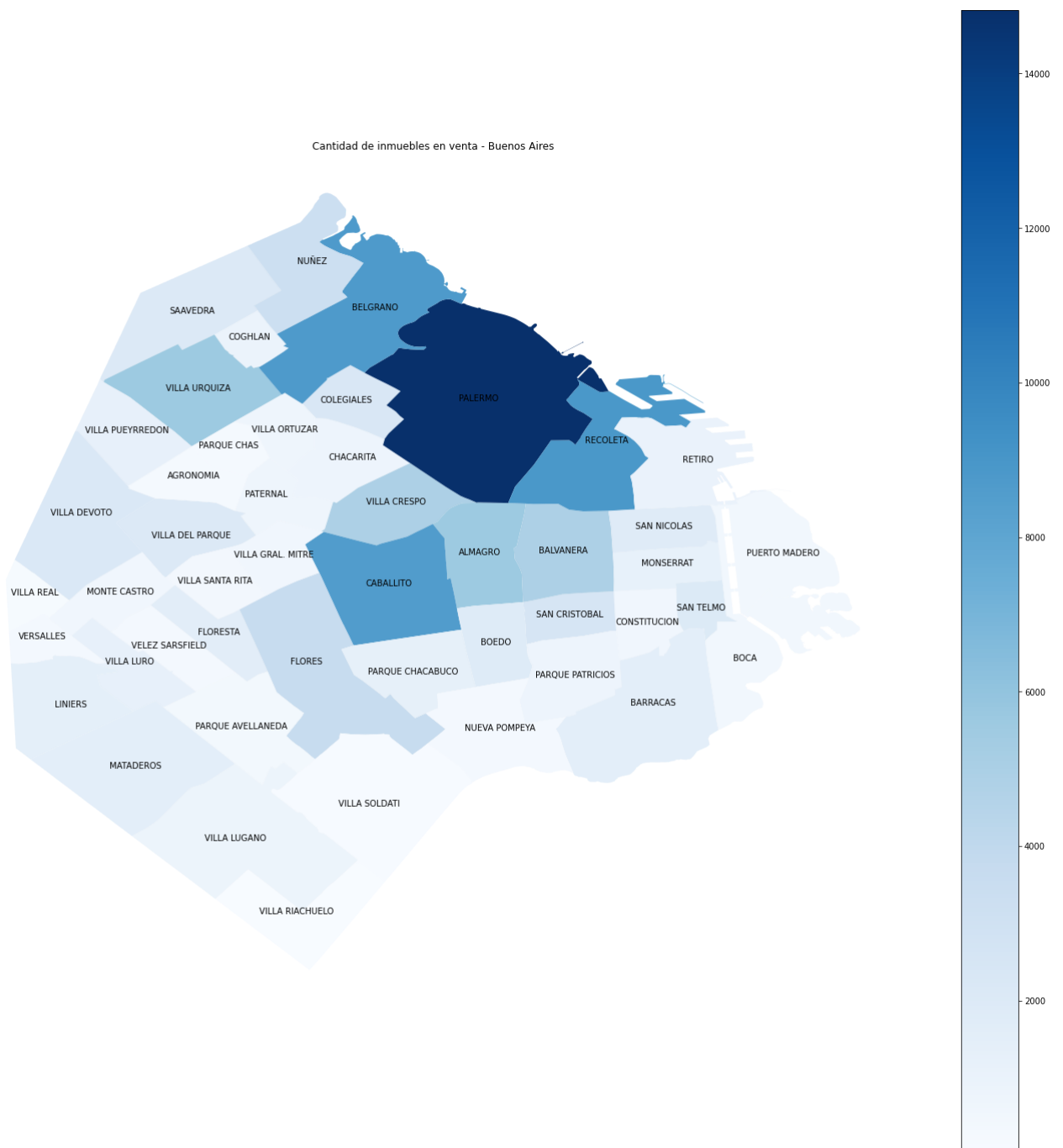


6.3 Mapa cantidad de inmuebles en venta

```
In [130]: gpd["cantidad"] = gpd.barrio.replace(df.l3.value_counts())
```

```
In [131]: fig, ax = plt.subplots(figsize=(25,25))
gpd.plot("cantidad", legend=True, ax=ax, cmap="Blues")
plt.axis('off')
for idx, row in gpd.iterrows():
    ax.annotate(s=row['barrio'], xy=row['coords'],
               horizontalalignment='center')
ax.set_title("Cantidad de inmuebles en venta - Buenos Aires")
plt.show()
```

```
/var/folders/f0/53n8wsk15q5fyk0xbqk6840000gn/T/ipykernel_6032/3859238686.py:5: MatplotlibDeprecationWarning: The 's' parameter of annotate() has been renamed 'text' since Matplotlib 3.3; support for the old name will be dropped two minor releases later.
ax.annotate(s=row['barrio'], xy=row['coords'],
```

6.4 Cantidad ponderada por superficie

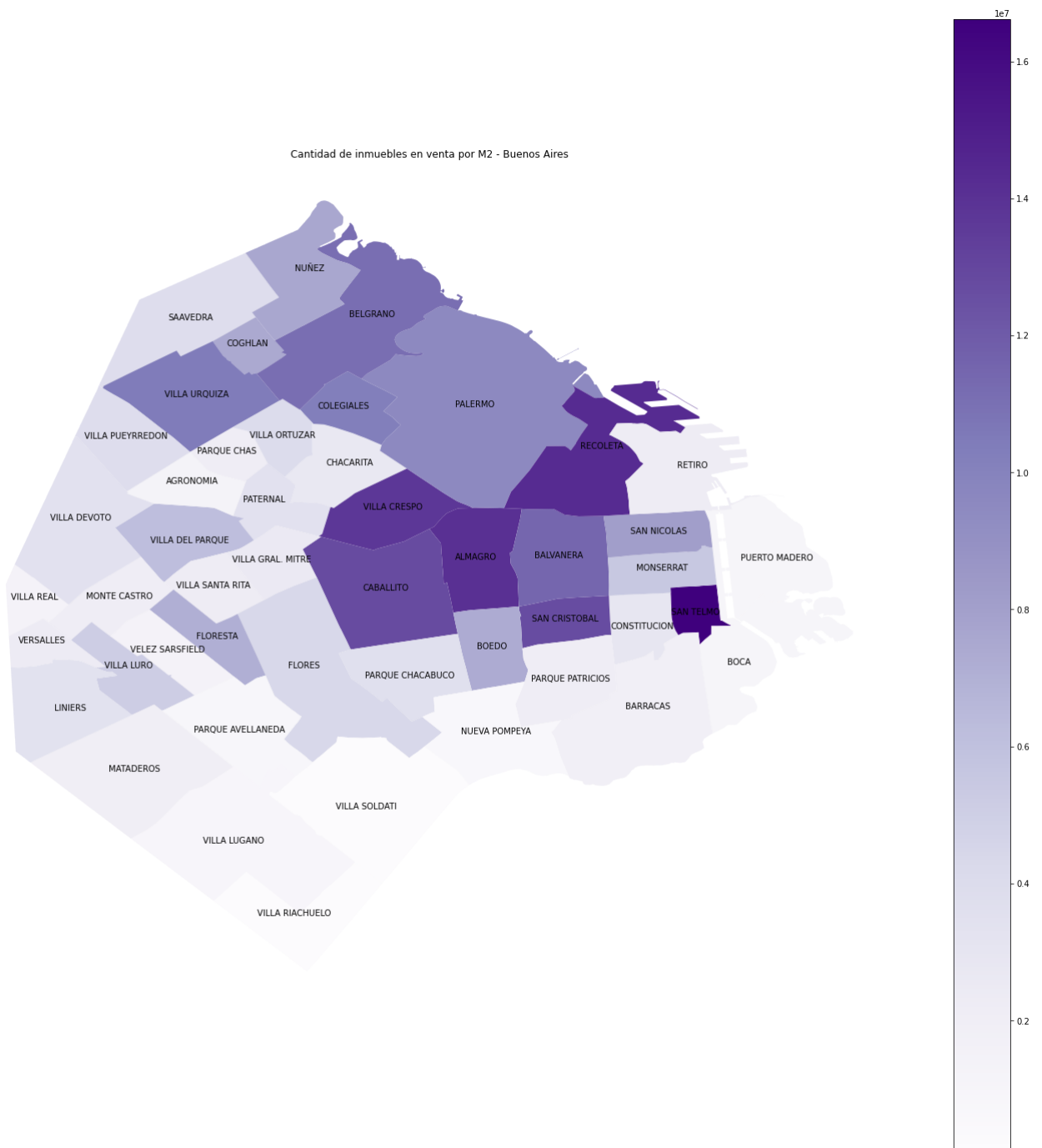
Cantidad de inmuebles por barrio ajustado por superficie.

```
In [132... gpd["inmPorM2"] = gpd.cantidad/gpd.area
```

```
In [133... fig, ax = plt.subplots(figsize=(25,25))
gpd.plot("inmPorM2", legend=True, ax=ax, cmap="Purples")
plt.axis('off')
for idx, row in gpd.iterrows():
    ax.annotate(s=row['barrio'], xy=row['coords'],
               horizontalalignment='center')
ax.set_title("Cantidad de inmuebles en venta por M2 - Buenos Aires")
plt.show()
```

/var/folders/f0/53n8wsk15q5fyyyk0xbqk6840000gn/T/ipykernel_6032/3127728140.py:5: MatplotlibDeprecationWarning: The 's' parameter of annotate() has been renamed 'text' since Matplotlib 3.3; support for the old name will be dropped two minor releases later.

```
ax.annotate(s=row['barrio'], xy=row['coords'],
```



7 - Regresión lineal

Con los datos vamos a generar un modelo de regresión lineal para predecir el valor por metro cuadrado de futuros inmuebles.

```
In [134... from sklearn.linear_model import LinearRegression
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

7.1 Procesamiento de datos

Para poder ajustar la regresión lineal es necesario primero ajustar el DataFrame.

```
In [135... df.drop(columns=["point", "index_right"], inplace=True)
```

7.1.1 Variables categóricas

Es necesario generar variables numéricas a partir de las categóricas que tenemos: barrio, sub-barrio y tipo de propiedad.

```
In [136... df = pd.get_dummies(df, columns=["13", "14", "property_type"])
```

7.1.2 Estandarización de variables numéricas

Estandarizamos las variables numéricas.

```
In [137... scaler = StandardScaler()
```

7.1.3 Creación de set de train y test

```
In [138... x = df.drop(columns=["precio_M2", "price"])
y = df.precio_M2
x_train, x_test, y_train, y_test = train_test_split(x, y)
```

7.2 Train del modelo

Generación de la regresión lineal.

```
In [139-- model = LinearRegression()

In [140-- pipe = make_pipeline(scaler, model)

In [141-- pipe.fit(X_train,y_train)

Out[141-- Pipeline(steps=[('standardscaler', StandardScaler()),
                  ('linearregression', LinearRegression())])]
```

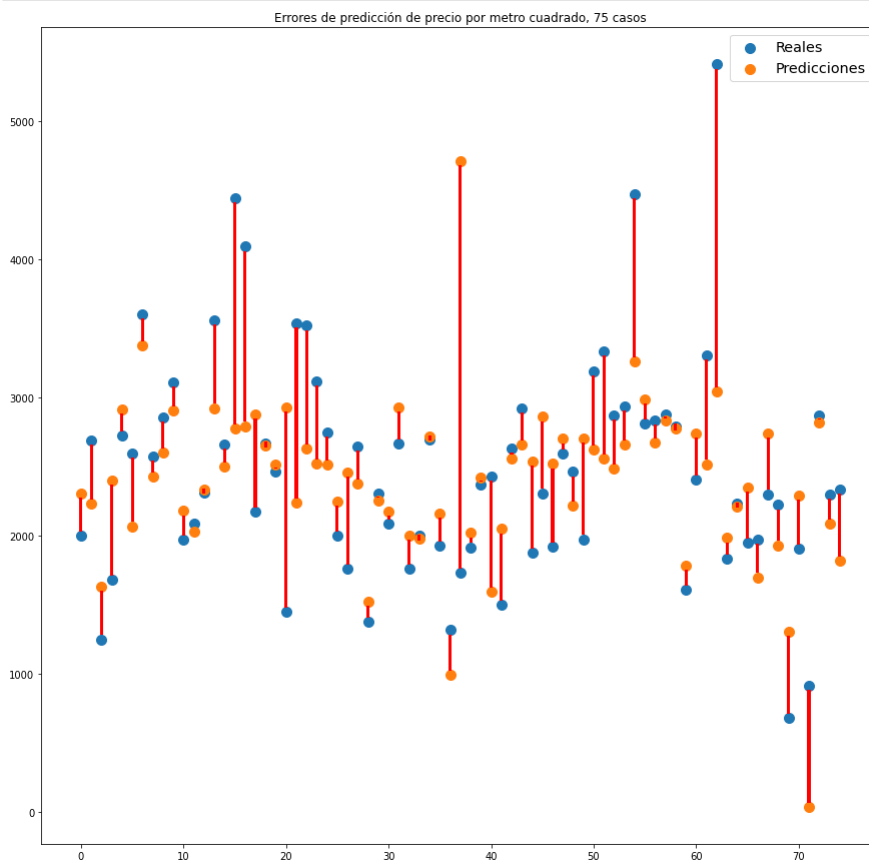
7.3 Test del modelo

```
In [142-- predicciones = pipe.predict(X_test)
reales = list(y_test)
```

Veamos el error en las primeras 50 predicciones.

```
In [143-- n = 75
d = 30
markerSize = 100
fig, ax = plt.subplots(figsize=(15,15))
pred = predicciones[:n]
real = reales[:n]
ax.scatter(range(n),real, s = markerSize, label="Reales")
ax.scatter(range(n),pred, s = markerSize, label="Predicciones")
for x in range(n):
    b = min(pred[x], real[x])
    h = max(pred[x], real[x])
    ax.bar(x,height=h-b-2*d, bottom=b+d,width=0.3,align="center",color="red")

ax.legend(loc='upper right', fontsize="x-large")
ax.set_title("Errores de predicción de precio por metro cuadrado, {} casos".format(n))
plt.show()
```



7.3.1 Puntaje: R2

Puntaje que el modelo utiliza para calcular la eficacia en la predicción vs los datos reales. Mas info en [sklearn docs](#).

```
In [144-- pipe.score(X_test, y_test)

Out[144-- 0.427446007012118]
```

8 - Tasador de propiedad

Ejemplo de uso del modelo.

Vamos a tasar el siguiente inmueble [link a publicación](#).

```
In [145-- info = {
    "lat": -34.580889,
    "lon": -58.442925,
    "l3": "PALERMO",
    "l4": "Palermo",
    "rooms": 1,
    "bedrooms": 1,
    "bathrooms": 1,
    "surface_total": 41,
    "surface_covered": 41,
    "property_type": "Departamento",
}
```

In [146...

```
def predecirM2(datos, pipeline):
    cols = pipeline.feature_names_in_
    d = {}
    for x in cols:
        k = info.keys()
        if x in info:
            d[x] = [info[x]]
        else:
            d[x] = [0]
    a = "l3_" + info["l3"]
    d[a] = [1]
    b = "l4_" + info["l4"]
    d[b] = [1]
    c = "property_type_" + info["property_type"]
    d[c] = [1]
    test = pd.DataFrame(data=d)
    return pipeline.predict(test)[0]
```

In [147...

```
precioM2 = predecirM2(info, pipe)
```

In [148...

```
print("Precio publicado: USD 145000.00 \nPrecio predicción: USD {:.2f}".format(precioM2*info["surface_total"]))
```

Precio publicado: USD 145000.00
Precio predicción: USD 119241.22

Datos provistos por [Properati](#)