

Práctica Final AA

February 17, 2021

Autora: Helena Antich Homar

Asignatura: Aprendizaje automático

Ejercicio: Práctica final: *Dataset* de Airbnb

Fecha de entrega: 17 Febrero de 2021

Enlace a repositorio abierto: [Proyecto Airbnb de HelenaAH30](#)

Descripción del programa: Este programa analiza y limpia los datos del fichero de Airbnb. Al que posteriormente se le realiza un ajuste con un modelo de regresión. Para la selección del modelo se utilizan diversas métricas y modelos. El resultado es un modelo concreto ya entrenado con el *dataset* indicado.

1 Librerías y directorios

A continuación vamos a cargar todas las librerías a utilizar en el programa y a determinar el directorio de trabajo para poder leer los datos.

Para modificar el directorio de trabajo, en el que se ubica el fichero de datos y una imagen auxiliar (adjuntada), hay que modificar la variable `working_directory`.

```
[1]: """ Libraries
import os
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, TweedieRegressor,
↳SGDRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
#from sklearn.cross_validation import cross_val_score
```

```
[2]:
```

```

#%% Directories
working_directory = '/Users/Helena/Documents/MASTER MUSI/AA - ML - Aprendizaje_
→automático/FINAL' #cambiar para poder realizar el cargado de los datos y la_
→imagen auxiliar

os.chdir(working_directory)
path_data = os.path.join(os.getcwd(), 'airbnb.csv') # datos utilizados para el_
→entrenamiento del modelo
path_img_map = os.path.join(os.getcwd(), 'map.png') # mapa para la_
→representación de los datos sobre la isla de Mallorca

```

2 Preprocesado de los datos

En este apartado trataremos los datos para optimizar el entrenamiento de la IA que se realizará al final.

A continuación cargamos los datos y vemos una breve descripción:

```

[3]: #%% Loading data
df_airbnb_raw = pd.read_csv(path_data)
df_airbnb = df_airbnb_raw.copy()
#df_airbnb.head()
df_airbnb.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17608 entries, 0 to 17607
Data columns (total 74 columns):

```

#	Column	Non-Null Count	Dtype
0	id	17608 non-null	int64
1	listing_url	17608 non-null	object
2	scrape_id	17608 non-null	int64
3	last_scraped	17608 non-null	object
4	name	17607 non-null	object
5	description	17393 non-null	object
6	neighborhood_overview	8213 non-null	object
7	picture_url	17608 non-null	object
8	host_id	17608 non-null	int64
9	host_url	17608 non-null	object
10	host_name	17606 non-null	object
11	host_since	17606 non-null	object
12	host_location	17572 non-null	object
13	host_about	11696 non-null	object
14	host_response_time	15862 non-null	object
15	host_response_rate	15862 non-null	object

16	host_acceptance_rate	16098	non-null	object
17	host_is_superhost	17606	non-null	object
18	host_thumbnail_url	17606	non-null	object
19	host_picture_url	17606	non-null	object
20	host_neighbourhood	364	non-null	object
21	host_listings_count	17606	non-null	float64
22	host_total_listings_count	17606	non-null	float64
23	host_verifications	17608	non-null	object
24	host_has_profile_pic	17606	non-null	object
25	host_identity_verified	17606	non-null	object
26	neighbourhood	8213	non-null	object
27	neighbourhood_cleansed	17608	non-null	object
28	neighbourhood_group_cleansed	0	non-null	float64
29	latitude	17608	non-null	float64
30	longitude	17608	non-null	float64
31	property_type	17608	non-null	object
32	room_type	17608	non-null	object
33	accommodates	17608	non-null	int64
34	bathrooms	0	non-null	float64
35	bathrooms_text	17600	non-null	object
36	bedrooms	17333	non-null	float64
37	beds	17511	non-null	float64
38	amenities	17608	non-null	object
39	price	17608	non-null	object
40	minimum_nights	17608	non-null	int64
41	maximum_nights	17608	non-null	int64
42	minimum_minimum_nights	17608	non-null	int64
43	maximum_minimum_nights	17608	non-null	int64
44	minimum_maximum_nights	17608	non-null	int64
45	maximum_maximum_nights	17608	non-null	int64
46	minimum_nights_avg_ntm	17608	non-null	float64
47	maximum_nights_avg_ntm	17608	non-null	float64
48	calendar_updated	0	non-null	float64
49	has_availability	17608	non-null	object
50	availability_30	17608	non-null	int64
51	availability_60	17608	non-null	int64
52	availability_90	17608	non-null	int64
53	availability_365	17608	non-null	int64
54	calendar_last_scraped	17608	non-null	object
55	number_of_reviews	17608	non-null	int64
56	number_of_reviews_ltm	17608	non-null	int64
57	number_of_reviews_l30d	17608	non-null	int64
58	first_review	11173	non-null	object
59	last_review	11173	non-null	object
60	review_scores_rating	10957	non-null	float64
61	review_scores_accuracy	10951	non-null	float64
62	review_scores_cleanliness	10953	non-null	float64
63	review_scores_checkin	10949	non-null	float64

```

64 review_scores_communication      10951 non-null float64
65 review_scores_location           10950 non-null float64
66 review_scores_value              10949 non-null float64
67 license                          11431 non-null object
68 instant_bookable                 17608 non-null object
69 calculated_host_listings_count    17608 non-null int64
70 calculated_host_listings_count_entire_homes 17608 non-null int64
71 calculated_host_listings_count_private_rooms 17608 non-null int64
72 calculated_host_listings_count_shared_rooms 17608 non-null int64
73 reviews_per_month                11173 non-null float64
dtypes: float64(19), int64(21), object(34)
memory usage: 9.9+ MB

```

2.1 Eliminación de columnas nulas

Viendo que hay columnas que son todo “null” podemos eliminarlas directamente, ya que no aportan información.

```

[4]: df_airbnb = df_airbnb.drop(columns = [
    → ['calendar_updated', 'bathrooms', 'neighbourhood_group_cleansed'])
df_airbnb.head()

```

```

[4]:      id      listing_url      scrape_id last_scraped \
0   11547  https://www.airbnb.com/rooms/11547  20200919153121  2020-09-21
1   100831 https://www.airbnb.com/rooms/100831  20200919153121  2020-09-21
2   105891 https://www.airbnb.com/rooms/105891  20200919153121  2020-09-20
3   106833 https://www.airbnb.com/rooms/106833  20200919153121  2020-09-20
4   130669 https://www.airbnb.com/rooms/130669  20200919153121  2020-09-20

      name \
0      My home at the beach
1      HOUSE IN MALLORCA - WiFi(ET-3045)
2  VILLAGE HOUSE WITH POOL: IDEAL FOR FAMILIES
3      Villa with a big pool in Mallorca
4      Room great apartment

      description \
0  Sun, joy, relax, quality, beach & peace.<br />...
1  <b>The space</b><br />House situated in a quie...
2  The house is a street on the outskirts of the ...
3  <b>The space</b><br />This is a restored old b...
4  Located in a residential neighbourhood and 10m...

      neighborhood_overview \
0      NaN
1      NaN
2  The village's population does not reach two th...

```

3 NaN
 4 Located in the center of the city, within minu...

	picture_url	host_id	\
0	https://a0.muscache.com/pictures/494126/8c151b...	42942	
1	https://a0.muscache.com/pictures/675527/72b329...	529151	
2	https://a0.muscache.com/pictures/1036816/f36ce...	549192	
3	https://a0.muscache.com/pictures/710218/98134c...	551974	
4	https://a0.muscache.com/pictures/866653/58dc48...	643065	

	host_url	...	review_scores_communication	\
0	https://www.airbnb.com/users/show/42942	...	10.0	
1	https://www.airbnb.com/users/show/529151	...	10.0	
2	https://www.airbnb.com/users/show/549192	...	10.0	
3	https://www.airbnb.com/users/show/551974	...	10.0	
4	https://www.airbnb.com/users/show/643065	...	NaN	

	review_scores_location	review_scores_value	license	instant_bookable	\
0	10.0	10.0	NaN	f	
1	10.0	10.0	ETV-3045	t	
2	9.0	10.0	ETV/6127	t	
3	9.0	9.0	ET/1961	f	
4	NaN	NaN	NaN	t	

	calculated_host_listings_count	calculated_host_listings_count_entire_homes	\
0	1	1	
1	1	1	
2	2	2	
3	1	1	
4	2	0	

	calculated_host_listings_count_private_rooms	\
0	0	
1	0	
2	0	
3	0	
4	2	

	calculated_host_listings_count_shared_rooms	reviews_per_month
0	0	0.93
1	0	1.47
2	0	0.14
3	0	0.09
4	0	NaN

[5 rows x 71 columns]

2.2 Tratamiento de la columna precio

Además el precio, que es la variable a predecir, es de tipo “objeto” y debería ser de tipo valor, incluso para hacer el estudio de outliers.

```
[5]: df_airbnb.price = df_airbnb.price.str.replace("$", "").str.replace(",", "").  
      ↪astype(float)  
df_airbnb.price.dtype  
df_airbnb.price.head()
```

```
[5]: 0      89.0  
     1     175.0  
     2     140.0  
     3     200.0  
     4     110.0  
     Name: price, dtype: float64
```

Ahora que el precio es del tipo número de coma flotante, veamos un resumen estadístico de esta columna:

```
[6]: df_airbnb.price.describe()
```

```
[6]: count      17608.000000  
     mean         244.383561  
     std         409.958169  
     min           0.000000  
     25%         110.000000  
     50%         179.000000  
     75%         275.000000  
     max        20736.000000  
     Name: price, dtype: float64
```

Dado que el mínimo es un “0” absoluto, tiene que haber un valor al menos que no tenga ningún precio. Este objeto sin precio puede desplazar la media y complicar el análisis mientras que en la realidad no aporta información. Por ello lo eliminamos.

```
[7]: df_airbnb[df_airbnb.price == 0.]  
df_airbnb = df_airbnb[df_airbnb.price != 0.]  
df_airbnb.price.describe()
```

```
[7]: count      17607.000000  
     mean         244.397441  
     std         409.965674  
     min          10.000000  
     25%         110.000000  
     50%         179.000000  
     75%         275.000000  
     max        20736.000000
```

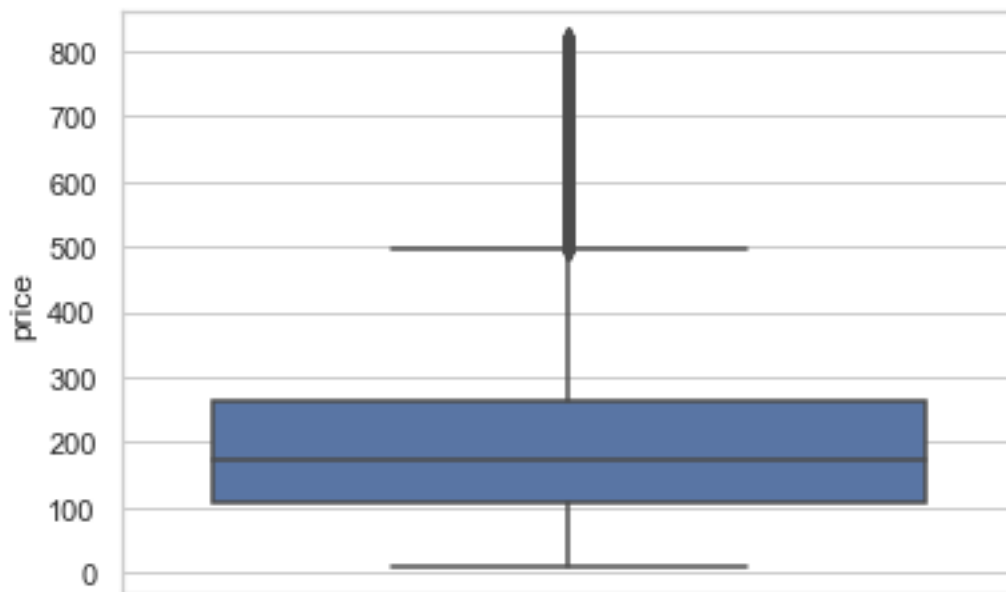
Name: price, dtype: float64

2.3 Análisis y eliminación de outliers

Teniendo en cuenta la desviación estándar, vamos a filtrar el dataframe para eliminar outliers analizando la distribución estadística del precio (boxplot) y la distribución por cuantiles (probplot).

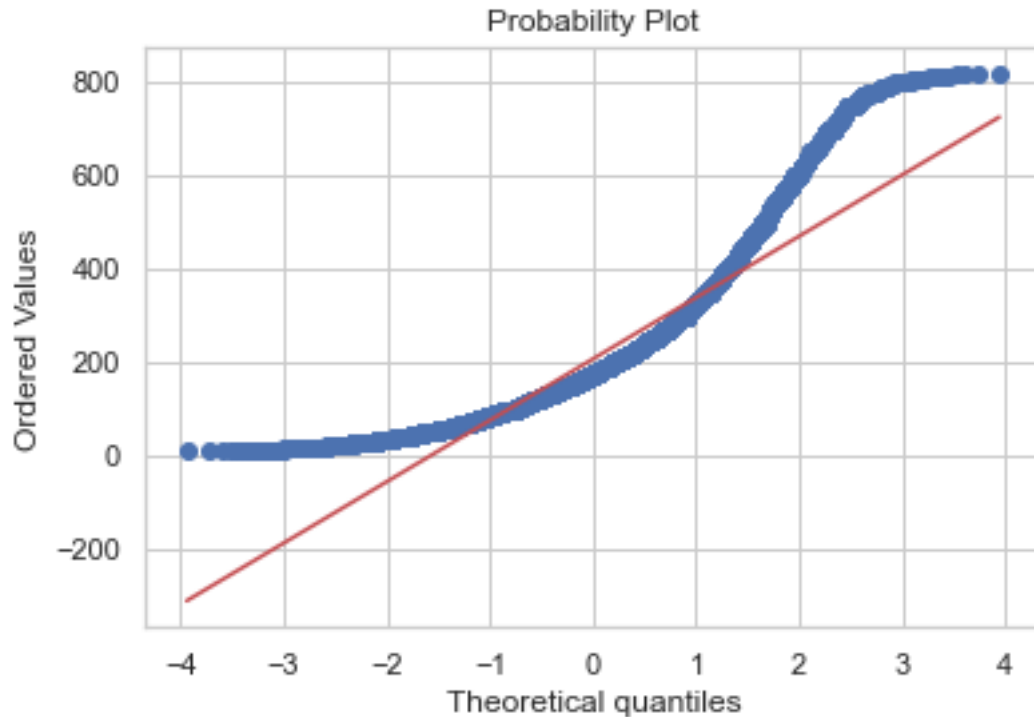
Nota: A lo largo del procesado de las columnas se ha ido realizando este estudio a modo de monitorización del dataset.

```
[8]: std = df_airbnb.price.describe()[2]
df = df_airbnb.loc[(df_airbnb.price <= 2*std)]
sns.set_theme(style="whitegrid")
ax = sns.boxplot(y="price", data=df)
```



```
[9]: stats.probplot(df.price, plot=sns.mpl.pyplot)
```

```
[9]: ((array([-3.94235327, -3.72430797, -3.6049135 , ...,  3.6049135 ,
              3.72430797,  3.94235327])),
      array([ 10.,  10.,  10., ..., 819., 819., 819.])),
      (131.29438762345103, 207.04338629881855, 0.9385583739817922))
```



Habiendo analizado las distribuciones nos hemos quedado con el 95,4% de los datos. Lo que equivale a dos veces la desviación estándar. Se conservan algunos outliers para poder preceder ciertos comportamientos anómalos con el modelo.

2.4 Tratamiento de columnas y variables

A continuación vamos a seguir tratando las columnas del dataset. Para empezar, leyendo las primeras filas del dataset, podemos determinar si es posible utilizar la información que hay en las columnas.

```
[10]: df.columns
      df.head()
```

```
[10]:
```

	id	listing_url	scrape_id	last_scraped	\
0	11547	https://www.airbnb.com/rooms/11547	20200919153121	2020-09-21	
1	100831	https://www.airbnb.com/rooms/100831	20200919153121	2020-09-21	
2	105891	https://www.airbnb.com/rooms/105891	20200919153121	2020-09-20	
3	106833	https://www.airbnb.com/rooms/106833	20200919153121	2020-09-20	
4	130669	https://www.airbnb.com/rooms/130669	20200919153121	2020-09-20	


```

                                name \
0                                My home at the beach
1                HOUSE IN MALLORCA - WiFi(ET-3045)
```



```

2 VILLAGE HOUSE WITH POOL: IDEAL FOR FAMILIES
3     Villa with a big pool in Mallorca
4     Room great apartment

```

```

description \
0 Sun, joy, relax, quality, beach & peace.<br />...
1 <b>The space</b><br />House situated in a quie...
2 The house is a street on the outskirts of the ...
3 <b>The space</b><br />This is a restored old b...
4 Located in a residential neighbourhood and 10m...

```

```

neighborhood_overview \
0 NaN
1 NaN
2 The village's population does not reach two th...
3 NaN
4 Located in the center of the city, within minu...

```

```

picture_url host_id \
0 https://a0.muscache.com/pictures/494126/8c151b... 42942
1 https://a0.muscache.com/pictures/675527/72b329... 529151
2 https://a0.muscache.com/pictures/1036816/f36ce... 549192
3 https://a0.muscache.com/pictures/710218/98134c... 551974
4 https://a0.muscache.com/pictures/866653/58dc48... 643065

```

```

host_url ... review_scores_communication \
0 https://www.airbnb.com/users/show/42942 ... 10.0
1 https://www.airbnb.com/users/show/529151 ... 10.0
2 https://www.airbnb.com/users/show/549192 ... 10.0
3 https://www.airbnb.com/users/show/551974 ... 10.0
4 https://www.airbnb.com/users/show/643065 ... NaN

```

```

review_scores_location review_scores_value license instant_bookable \
0 10.0 10.0 NaN f
1 10.0 10.0 ETV-3045 t
2 9.0 10.0 ETV/6127 t
3 9.0 9.0 ET/1961 f
4 NaN NaN NaN t

```

```

calculated_host_listings_count calculated_host_listings_count_entire_homes \
0 1 1
1 1 1
2 2 2
3 1 1
4 2 0

```

```

calculated_host_listings_count_private_rooms \

```

0	0
1	0
2	0
3	0
4	2

	calculated_host_listings_count_shared_rooms	reviews_per_month
0	0	0.93
1	0	1.47
2	0	0.14
3	0	0.09
4	0	NaN

[5 rows x 71 columns]

Ahora vamos a seleccionar las columnas que creemos que pueden ser relevantes. Algunos de los motivos de descarte pueden ser: * Descartamos las columnas, por ejemplo: `listing_url`, `scrape_id`, `name`, `description` y `neighborhood_overview`, porque la información de estas columnas se podría encontrar en otras. * Se descartan las columnas, por ejemplo: `picture_url`, `host_id`, y `host_url`, por ser valores aleatorios u arbitrarios asignados a cada hospedaje. * Se eliminan las columnas que se refieren de forma exclusiva al dueño del hospedaje, como nombre o su descripción, que no se refieren al hospedaje o influyen de ninguna manera sobre tiempos de respuesta del comprador.

```
[11]: df = df.drop(columns = [ 'last_scraped', 'host_since',
    → 'host_location', 'host_name', 'host_about', 'host_thumbnail_url',
    → 'host_picture_url', 'listing_url', 'scrape_id', 'name', 'description',
    → 'neighborhood_overview', 'picture_url', 'host_id', 'host_url',
    → 'host_response_rate', 'host_acceptance_rate', 'host_neighbourhood',
    → 'host_listings_count', 'host_total_listings_count', 'host_verifications',
    → 'host_has_profile_pic', 'host_response_time', 'availability_30', 'availability_60',
    → 'availability_90', 'availability_365', 'minimum_minimum_nights',
    → 'maximum_minimum_nights', 'minimum_maximum_nights',
    → 'maximum_maximum_nights', 'minimum_nights_avg_ntm',
    → 'maximum_nights_avg_ntm', 'number_of_reviews_ltm', 'number_of_reviews_l30d',
    → 'first_review', 'last_review', 'calculated_host_listings_count',
    → 'calculated_host_listings_count_entire_homes', 'calculated_host_listings_count_private_rooms',
    → 'calculated_host_listings_count_shared_rooms', 'reviews_per_month',
    → 'neighbourhood', 'calendar_last_scraped'])
df.columns
```

```
[11]: Index(['id', 'host_is_superhost', 'host_identity_verified',
    'neighbourhood_cleansed', 'latitude', 'longitude', 'property_type',
    'room_type', 'accommodates', 'bathrooms_text', 'bedrooms', 'beds',
    'amenities', 'price', 'minimum_nights', 'maximum_nights',
    'has_availability', 'number_of_reviews', 'review_scores_rating',
    'review_scores_accuracy', 'review_scores_cleanliness',
    'review_scores_checkin', 'review_scores_communication',
```

```

'review_scores_location', 'review_scores_value', 'license',
'instant_bookable'],
dtype='object')

```

```
[12]: df.head()
```

```

[12]:      id host_is_superhost host_identity_verified \
0    11547                f                t
1   100831                f                t
2   105891                t                t
3   106833                f                t
4   130669                f                f

      neighbourhood_cleansed latitude longitude      property_type \
0           Calvià      39.51888    2.48182    Entire apartment
1      Santa Margalida    39.76347    3.16255    Entire house
2      Maria de la Salut    39.66044    3.07165    Entire townhouse
3  Sant Llorenç des Cardassar    39.61600    3.30121    Entire villa
4      Palma de Mallorca    39.56478    2.60333  Private room in apartment

      room_type accommodates bathrooms_text ... number_of_reviews \
0  Entire home/apt            2         1 bath ...            103
1  Entire home/apt            8         3 baths ...             30
2  Entire home/apt            6         2 baths ...             14
3  Entire home/apt            4         1 bath ...              9
4    Private room            2         1 bath ...              0

      review_scores_rating review_scores_accuracy review_scores_cleanliness \
0                96.0                10.0                9.0
1                100.0                10.0                10.0
2                97.0                10.0                10.0
3                98.0                10.0                10.0
4                 NaN                 NaN                 NaN

      review_scores_checkin review_scores_communication review_scores_location \
0                10.0                10.0                10.0
1                10.0                10.0                10.0
2                10.0                10.0                9.0
3                10.0                10.0                9.0
4                 NaN                 NaN                 NaN

      review_scores_value license instant_bookable
0                10.0      NaN                f
1                10.0  ETV-3045                t
2                10.0  ETV/6127                t
3                 9.0   ET/1961                f
4                 NaN      NaN                t

```

[5 rows x 27 columns]

2.4.1 Columnas de tipo booleano con *strings*

A continuación trataremos aquellas columnas que tienen valores ‘f’ o ‘t’ y las convertiremos a valores binarios:

```
[13]: df = df.replace({'t': 1, 'f': 0})
```

Columna `bathrooms_text` A continuación se traduce la columna `bathrooms_text` a un valor numérico. En primer lugar vemos qué valores deberemos tratar:

```
[14]: df_airbnb.bathrooms_text.unique()
```

```
[14]: array(['1 bath', '3 baths', '2 baths', '1 private bath', '0 baths',  
          '1.5 baths', '2 shared baths', '7 baths', '4.5 baths', '2.5 baths',  
          '4 baths', '5 baths', 'Shared half-bath', '1.5 shared baths',  
          '3.5 baths', '1 shared bath', '6 baths', '8 baths', nan,  
          '6.5 baths', '9.5 baths', '5.5 baths', '7.5 baths',  
          '12 shared baths', '4 shared baths', '8.5 baths', '0 shared baths',  
          '2.5 shared baths', 'Half-bath', '12 baths', '11 baths',  
          '13 baths', '9 baths', '3 shared baths', '3.5 shared baths',  
          '12.5 baths', '32 baths', '19 baths', 'Private half-bath',  
          '13 shared baths', '10 baths', '14 baths', '16 baths'],  
          dtype=object)
```

En segundo lugar reemplazamos los “medios baños” por 0.5.

```
[15]: df.bathrooms_text = df.bathrooms_text.replace({"\w+ half-bath$": "0.5",  
          ↪ "Half-bath": "0.5"}, regex = True)  
df.bathrooms_text.unique()
```

```
[15]: array(['1 bath', '3 baths', '2 baths', '1 private bath', '0 baths',  
          '1.5 baths', '2 shared baths', '7 baths', '4.5 baths', '2.5 baths',  
          '4 baths', '5 baths', '0.5', '1.5 shared baths', '1 shared bath',  
          '3.5 baths', '6 baths', nan, '6.5 baths', '5.5 baths', '7.5 baths',  
          '9.5 baths', '4 shared baths', '8 baths', '0 shared baths',  
          '2.5 shared baths', '11 baths', '9 baths', '3 shared baths',  
          '3.5 shared baths', '12.5 baths', '32 baths', '8.5 baths',  
          '13 shared baths', '12 baths', '10 baths', '13 baths'],  
          dtype=object)
```

Ahora extraemos el valor numérico que hay dentro del string y lo almacenamos en una nueva columna.

```
[16]: df['bathrooms'] = df['bathrooms_text'].str.extract('(\d*\.\d+|\d+)', expand =_
      ↪False).astype(float)
df.bathrooms = df.bathrooms.fillna(0)
df.bathrooms.unique()
```

```
[16]: array([ 1. ,  3. ,  2. ,  0. ,  1.5,  7. ,  4.5,  2.5,  4. ,  5. ,  0.5,
          3.5,  6. ,  6.5,  5.5,  7.5,  9.5,  8. , 11. ,  9. , 12.5, 32. ,
          8.5, 13. , 12. , 10. ])
```

Ahora ya se puede eliminar la columna con los strings.

```
[17]: df = df.drop(columns = ['bathrooms_text'])
df.columns
```

```
[17]: Index(['id', 'host_is_superhost', 'host_identity_verified',
          'neighbourhood_cleansed', 'latitude', 'longitude', 'property_type',
          'room_type', 'accommodates', 'bedrooms', 'beds', 'amenities', 'price',
          'minimum_nights', 'maximum_nights', 'has_availability',
          'number_of_reviews', 'review_scores_rating', 'review_scores_accuracy',
          'review_scores_cleanliness', 'review_scores_checkin',
          'review_scores_communication', 'review_scores_location',
          'review_scores_value', 'license', 'instant_bookable', 'bathrooms'],
          dtype='object')
```

```
[18]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17181 entries, 0 to 17607
Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                    17181 non-null  int64
1   host_is_superhost                    17179 non-null  float64
2   host_identity_verified                17179 non-null  float64
3   neighbourhood_cleansed                17181 non-null  object
4   latitude                             17181 non-null  float64
5   longitude                             17181 non-null  float64
6   property_type                        17181 non-null  object
7   room_type                            17181 non-null  object
8   accommodates                         17181 non-null  int64
9   bedrooms                             16911 non-null  float64
10  beds                                 17097 non-null  float64
11  amenities                             17181 non-null  object
12  price                                 17181 non-null  float64
13  minimum_nights                       17181 non-null  int64
14  maximum_nights                       17181 non-null  int64
15  has_availability                     17181 non-null  int64
16  number_of_reviews                    17181 non-null  int64
```

```

17 review_scores_rating      10833 non-null float64
18 review_scores_accuracy    10829 non-null float64
19 review_scores_cleanliness 10831 non-null float64
20 review_scores_checkin     10827 non-null float64
21 review_scores_communication 10829 non-null float64
22 review_scores_location    10828 non-null float64
23 review_scores_value       10827 non-null float64
24 license                   11184 non-null object
25 instant_bookable          17181 non-null int64
26 bathrooms                 17181 non-null float64
dtypes: float64(15), int64(7), object(5)
memory usage: 3.7+ MB

```

2.4.2 Columna *license*

La siguiente columna a tratar será la de licencia. Vamos a codificar la columna en licencia o no licencia:

```
[19]: df.license.unique()
```

```
[19]: array([nan, 'ETV-3045', 'ETV/6127', ..., 'ETV/5754', 'LIZE84/2017',
        '00551ETV'], dtype=object)
```

```
[20]: df.license = df.license.fillna(0)
df.license[df.license != 0] = 1
df.license.astype(int)
df.license
```

```
[20]: 0      0
1      1
2      1
3      1
4      0
..
17603   1
17604   0
17605   1
17606   0
17607   0
Name: license, Length: 17181, dtype: object
```

Antes de empezar con los tipos de alojamiento o habitación, vamos a realizar una separación de *amenities* y haremos un conteo de cada uno para las distintas propiedades. Para ello se definirá un nuevo dataframe que luego podremos concatenar con el anterior.

2.4.3 Columna *amenities*

Para la creación de este dataframe vamos a utilizar la función `CountVectorizer` del paquete `scikit-learn`. Se ha entendido su funcionamiento a partir de una búsqueda en *stack overflow* y posteriormente de las instrucciones de la librería ([enlace](#)).

Esta función permite identificar distintas etiquetas de un *string* y además genera una matriz para identificar las posiciones de éstas. Utilizando estas dos características podemos generar el nuevo *data frame*.

En primer lugar extraemos la matriz de posiciones y los valores de las etiquetas:

```
[21]: vectorizer = CountVectorizer(tokenizer=lambda row: row.split(','))

matriz = vectorizer.fit_transform(df.amenities)
etiquetas = vectorizer.get_feature_names()
```

Ahora corregimos algunas de las etiquetas, que, por cómo están contruidos los datos, incluyen algunos caracteres que las harán difíciles de utilizar:

```
[22]: for ind, amenit in enumerate(etiquetas):
        amenit_limpias = amenit.replace('"', '').replace('[', '').replace(']', '').
        ↪strip()
        etiquetas[ind] = amenit_limpias
```

Finalmente creamos el dataframe nuevo y borramos las columnas a reemplazar:

```
[23]: df_amenities = pd.DataFrame(matriz.toarray(), columns=etiquetas)
df_amenities.drop('', axis=1, inplace=True)
df_amenities.columns
```

```
[23]: Index(['air conditioning', 'air conditioning', 'baby bath', 'baby bath',
          'baby monitor', 'babysitter recommendations',
          'babysitter recommendations', 'baking sheet', 'balcony',
          'barbecue utensils',
          ...,
          'record player', 'room-darkening shades', 'shampoo', 'smoke alarm',
          'table corner guards', 'terrace', 'tv', 'tv', 'washer', 'wifi'],
          dtype='object', length=229)
```

Comprobamos si hay columnas duplicadas:

(Para hacerlo más intuitivo, la siguiente línea devuelve un `true` si hay columnas duplicadas y un `false` si no las hay)

```
[24]: not(len(df_amenities.columns) == len(df_amenities.columns.unique()))
```

```
[24]: True
```

Efectivamente, hay alguna o algunas columnas duplicadas. El paso siguiente es convertir estas

múltiples columnas en una sola, de manera que esta indique si el hospedaje tiene o no esta característica, de esta manera conseguimos columnas que coinciden con la técnica de “OneHotEncoding”.

```
[25]: cols = df_amenities.columns.tolist()
new_df = pd.DataFrame()

for ncol in range(len(cols)):
    col_name = cols[ncol]
    if col_name not in new_df.columns:
        indices = [i for i, x in enumerate(cols) if x == col_name]
        new_df.loc[:, col_name] = df_amenities.iloc[:, indices].sum(axis=1).
        ↳tolist()

new_df[new_df > 0] = 1 #Convertimos a binario
```

Realizamos de nuevo la comprobación, para estar seguros de que no hay duplicados:
(En este caso un valor True representa que no hay duplicaciones)

```
[26]: (new_df.columns == new_df.columns.unique()).all()
```

[26]: True

Ahora que tenemos la categoría de **amenities** tratada, se procede a la agregación de estas columnas al *dataframe* original y la eliminación de la columna que contenía la información. Ello se realiza mediante una concatenación de columnas ya que no se ha alterado el orden de los índices en todo el proceso.

```
[27]: df = df.drop(columns = ['amenities'])
df = pd.concat([df, new_df], axis=1)
```

```
[28]: df.head()
```

```
[28]:
```

	id	host_is_superhost	host_identity_verified	\
0	11547.0	0.0	1.0	
1	100831.0	0.0	1.0	
2	105891.0	1.0	1.0	
3	106833.0	0.0	1.0	
4	130669.0	0.0	0.0	

	neighbourhood_cleansed	latitude	longitude	property_type	\
0	Calvià	39.51888	2.48182	Entire apartment	
1	Santa Margalida	39.76347	3.16255	Entire house	
2	Maria de la Salut	39.66044	3.07165	Entire townhouse	
3	Sant Llorenç des Cardassar	39.61600	3.30121	Entire villa	
4	Palma de Mallorca	39.56478	2.60333	Private room in apartment	

	room_type	accommodates	bedrooms	...	trash compactor	tv	\
0	Entire home/apt	2.0	1.0	...	0.0	1.0	

1	Entire home/apt	8.0	4.0	...	0.0	1.0
2	Entire home/apt	6.0	3.0	...	0.0	1.0
3	Entire home/apt	4.0	2.0	...	0.0	1.0
4	Private room	2.0	1.0	...	0.0	1.0

	walk in closet	washer	waterfront	wifi	window guards	wine cooler	\
0	0.0	1.0	0.0	1.0	0.0	0.0	
1	0.0	1.0	0.0	1.0	0.0	0.0	
2	0.0	1.0	0.0	1.0	0.0	0.0	
3	0.0	0.0	0.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	0.0	

	wood-burning fireplace	terrace
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

[5 rows x 154 columns]

2.4.4 Columnas de tipo de alojamiento y propiedad

A continuación se procede a analizar cuál de las columnas `property_type` o `room_type` conviene más tratar y posteriormente se aplicará "OneHotEncoding". Para ello, analizamos las dos columnas.

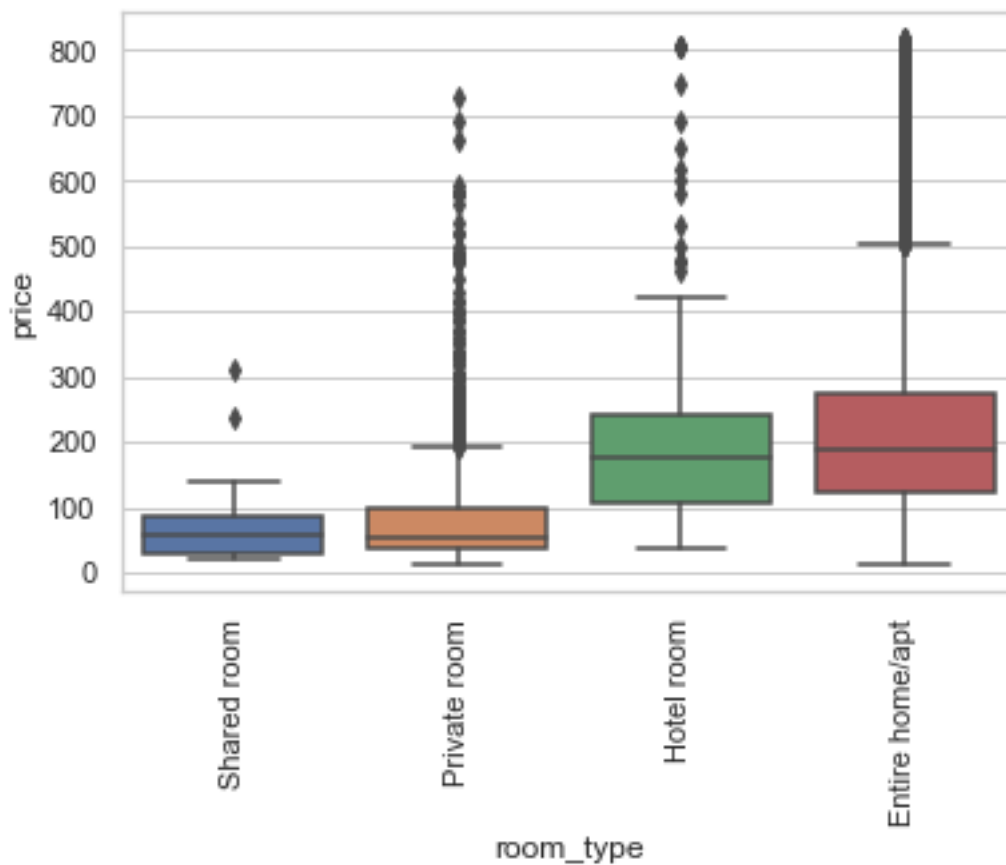
En primer lugar analizamos `room_type`:

```
[29]: df.room_type.unique()
```

```
[29]: array(['Entire home/apt', 'Private room', 'Hotel room', nan,
           'Shared room'], dtype=object)
```

```
[30]: fig, ax = plt.subplots()
sorted_average = df.groupby('room_type').mean().
    ↳sort_values('price', ascending=True)
ax = sns.boxplot(x="room_type", y="price", data=df, order=sorted_average.index)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```

```
[30]: [Text(0, 0, 'Shared room'),
       Text(1, 0, 'Private room'),
       Text(2, 0, 'Hotel room'),
       Text(3, 0, 'Entire home/apt')]
```



```
[31]: df_rooms = pd.get_dummies(df.room_type)
      df_rooms
```

```
[31]:
```

	Entire home/apt	Hotel room	Private room	Shared room
0	1	0	0	0
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
4	0	0	1	0
...
17603	1	0	0	0
17604	1	0	0	0
17605	1	0	0	0
17606	0	0	1	0
17607	1	0	0	0

```
[17603 rows x 4 columns]
```

Con el nuevo dataframe creado únicamente falta eliminar la columna previa y añadir las nuevas ya codificadas.

```
[32]: df = df.drop(columns = ['room_type'])
df = pd.concat([df, df_rooms], axis=1)
df.head()
```

```
[32]:      id  host_is_superhost  host_identity_verified \
0   11547.0                0.0                    1.0
1   100831.0                0.0                    1.0
2   105891.0                1.0                    1.0
3   106833.0                0.0                    1.0
4   130669.0                0.0                    0.0

      neighbourhood_cleansed  latitude  longitude  property_type \
0                Calvià  39.51888    2.48182    Entire apartment
1          Santa Margalida  39.76347    3.16255    Entire house
2          Maria de la Salut  39.66044    3.07165    Entire townhouse
3  Sant Llorenç des Cardassar  39.61600    3.30121    Entire villa
4          Palma de Mallorca  39.56478    2.60333  Private room in apartment

      accommodates  bedrooms  beds  ...  waterfront  wifi  window guards \
0                2.0        1.0  1.0  ...        0.0  1.0            0.0
1                8.0        4.0  7.0  ...        0.0  1.0            0.0
2                6.0        3.0  4.0  ...        0.0  1.0            0.0
3                4.0        2.0  4.0  ...        0.0  1.0            0.0
4                2.0        1.0  2.0  ...        0.0  1.0            0.0

      wine cooler  wood-burning fireplace  terrace  Entire home/apt  Hotel room \
0                0.0                    0.0    0.0                1            0
1                0.0                    0.0    0.0                1            0
2                0.0                    0.0    0.0                1            0
3                0.0                    0.0    0.0                1            0
4                0.0                    0.0    0.0                0            0

      Private room  Shared room
0                0            0
1                0            0
2                0            0
3                0            0
4                1            0

[5 rows x 157 columns]
```

En segundo lugar analizamos `property_type`:

```
[33]: df.property_type.unique()
```

```
[33]: array(['Entire apartment', 'Entire house', 'Entire townhouse',
        'Entire villa', 'Private room in apartment', 'Entire guesthouse',
```

```

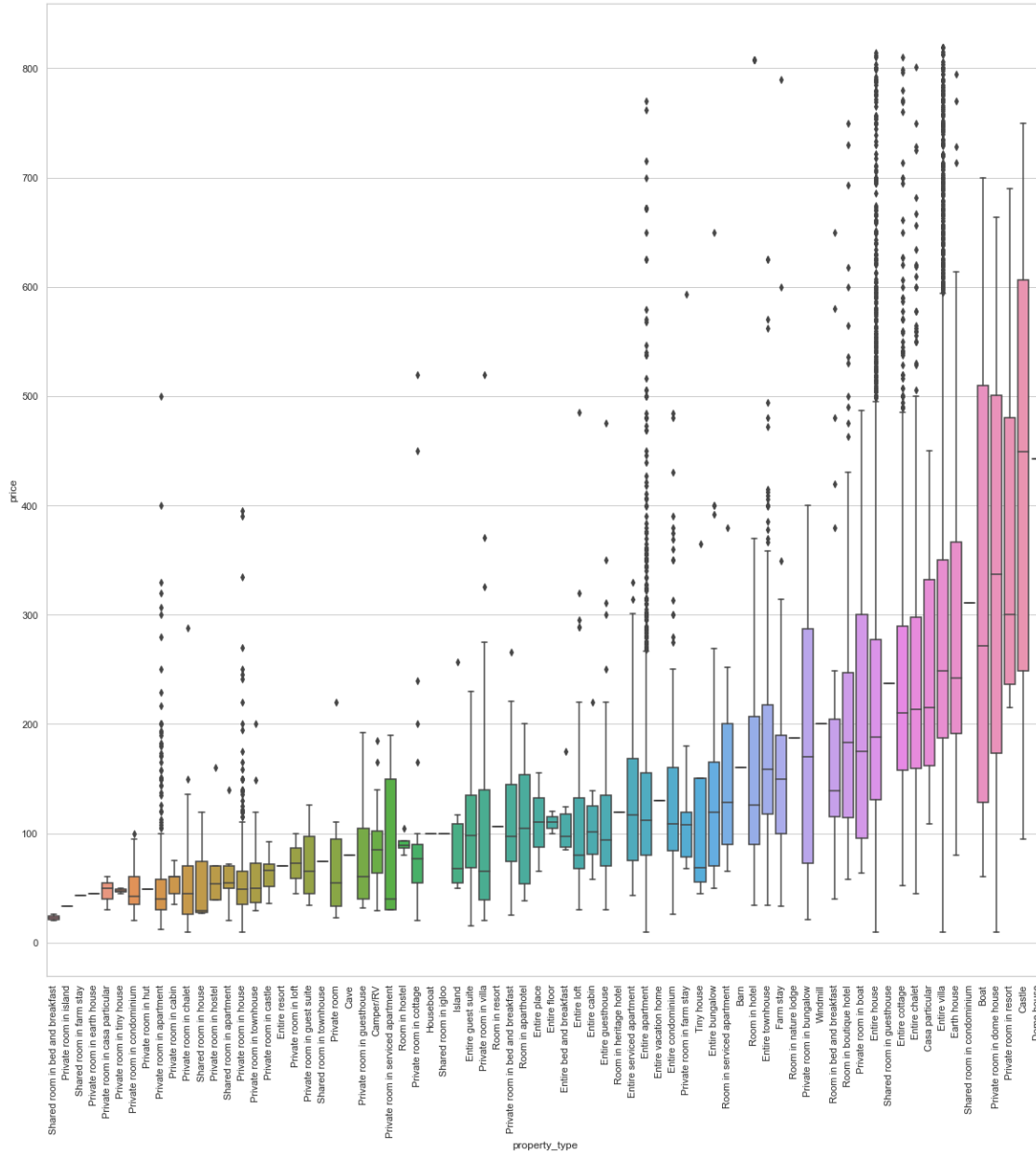
'Private room in guest suite', 'Private room in villa',
'Entire loft', 'Entire cottage', 'Castle', 'Entire condominium',
'Room in boutique hotel', 'Private room', 'Private room in house',
'Entire chalet', 'Entire cabin', nan,
'Private room in condominium', 'Entire guest suite',
'Private room in boat', 'Private room in cottage', 'Camper/RV',
'Boat', 'Private room in townhouse', 'Private room in hostel',
'Private room in bed and breakfast', 'Entire serviced apartment',
'Private room in guesthouse', 'Shared room in apartment',
'Farm stay', 'Room in serviced apartment',
'Room in bed and breakfast', 'Private room in chalet',
'Entire bungalow', 'Room in hotel', 'Island',
'Shared room in igloo', 'Private room in farm stay',
'Shared room in bed and breakfast', 'Entire bed and breakfast',
'Entire place', 'Room in aparthotel', 'Entire vacation home',
'Private room in bungalow', 'Shared room in house', 'Earth house',
'Entire floor', 'Room in hostel', 'Room in resort',
'Room in nature lodge', 'Private room in cabin', 'Tiny house',
'Shared room in condominium', 'Shared room in guesthouse',
'Private room in hut', 'Entire resort', 'Windmill', 'Dome house',
'Private room in loft', 'Private room in serviced apartment',
'Private room in earth house', 'Private room in casa particular',
'Casa particular', 'Cave', 'Private room in castle',
'Private room in resort', 'Private room in dome house',
'Private room in tiny house', 'Houseboat',
'Shared room in farm stay', 'Room in heritage hotel',
'Private room in island', 'Shared room in townhouse', 'Barn'],
dtype=object)

```

```

[34]: fig, ax = plt.subplots(figsize=(20, 20))
sorted_average = df.groupby('property_type').mean().
    ↪sort_values('price', ascending=True)
ax = sns.boxplot(x="property_type", y="price", data=df, order=sorted_average.
    ↪index)
ls = ax.get_xticklabels()
_ = ax.set_xticklabels(ls, rotation=90)

```



La columna de tipo de propiedad, aunque intuitivamente tiene relevancia para el análisis, contiene una alta variabilidad y al ver la distribución estadística de los valores por etiqueta no parece un sistema fiable, ni siquiera ordenado por la media del precio agrupado por tipo de propiedad. Por todo esto se decide eliminar la columna.

```
[35]: df = df.drop(columns = ['property_type']) #Se elimina la columna descartada
```

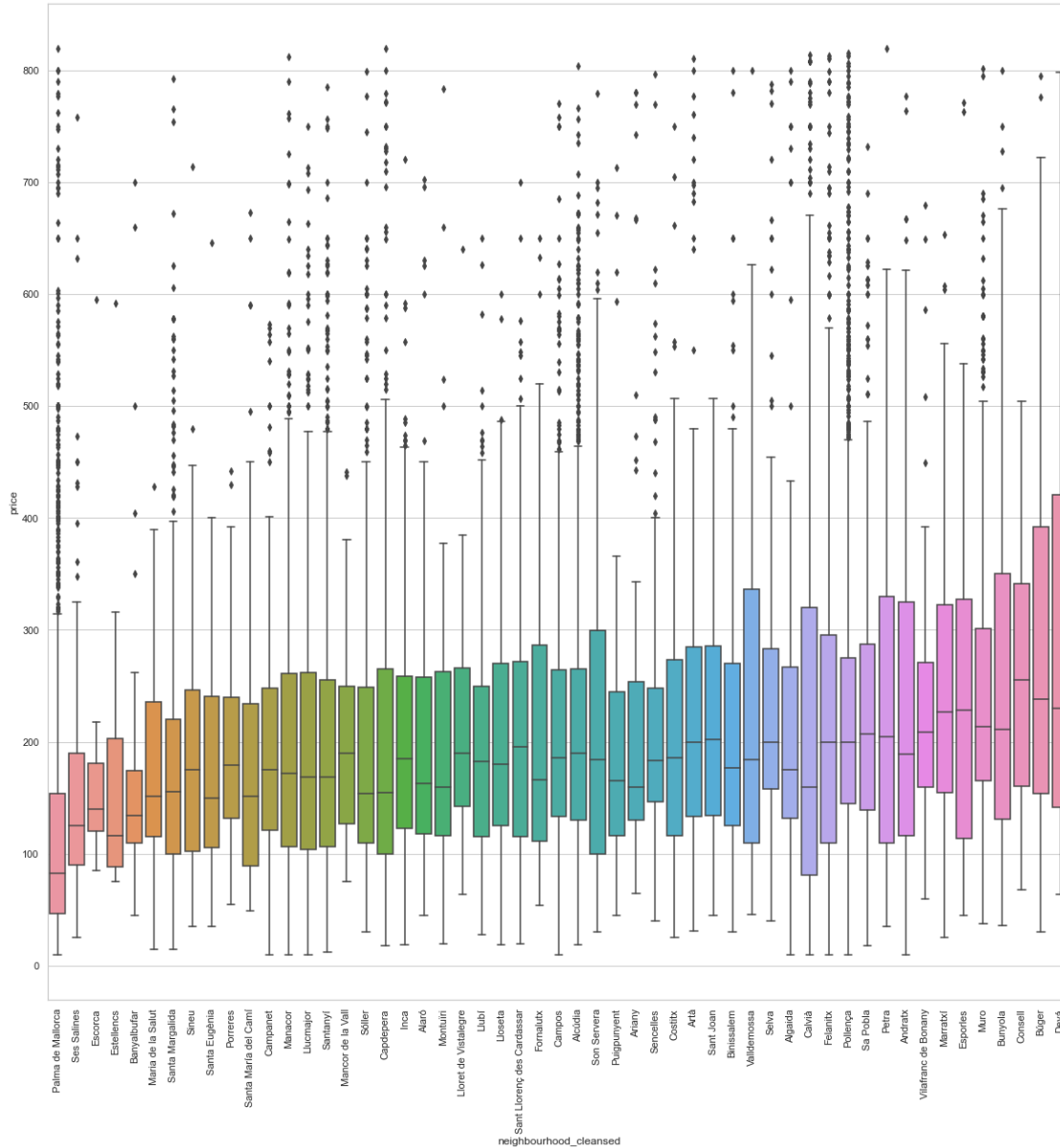
2.4.5 Columna neighbourhood_cleansed:

Esta columna accede a las propiedades de “barrio”, que son menos localizadas que latitud y longitud. Esta ligera deslocalización puede ser beneficiosa a la hora de aplicar el modelo.

```
[36]: df.neighbourhood_cleansed.unique()
```

```
[36]: array(['Calvià', 'Santa Margalida', 'Maria de la Salut',  
        'Sant Llorenç des Cardassar', 'Palma de Mallorca', 'Selva',  
        'Esporles', 'Banyalbufar', 'Manacor', 'Sa Pobla', 'Alcúdia',  
        'Pollença', 'Artà', 'Santanyí', 'Fornalutx', 'Campos', 'Llucmajor',  
        'Sineu', 'Lloseta', 'Marratxí', 'Felanitx', 'Llubí', 'Muro',  
        'Lloret de Vistalegre', 'Ariany', 'Campanet', 'Capdepera',  
        'Puigpunyent', 'Binissalem', 'Valldemossa', 'Algaida',  
        'Son Servera', 'Sóller', 'Ses Salines', nan, 'Mancor de la Vall',  
        'Andratx', 'Santa María del Camí', 'Inca', 'Costitx', 'Bunyola',  
        'Deyá', 'Búger', 'Porreres', 'Alaró', 'Montuïri', 'Escorca',  
        'Petra', 'Consell', 'Sencelles', 'Vilafranc de Bonany',  
        'Estellencs', 'Sant Joan', 'Santa Eugènia'], dtype=object)
```

```
[37]: fig, ax = plt.subplots(figsize=(20, 20))  
sorted_average = df.groupby('neighbourhood_cleansed').mean().  
    ↳ sort_values('price', ascending=True)  
ax = sns.boxplot(x="neighbourhood_cleansed", y="price", data=df,   
    ↳ order=sorted_average.index)  
_ = ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```



Según el *boxplot* hay una linealidad entre el lugar y la distribución de la media del precio, pese a los *outliers*. Por tanto, se decide que es una columna de utilidad. A continuación, se trata con la técnica *OHE*.

```
[38]: df_neigh = pd.get_dummies(df.neighbourhood_cleansed)
df_neigh
```

```
[38]:
```

	Alaró	Alcúdia	Algaida	Andratx	Ariany	Artà	Banyalbufar	\
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	

3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
...
17603	0	0	0	0	0	0	0
17604	0	0	0	0	0	0	0
17605	0	0	0	0	0	0	0
17606	0	0	0	1	0	0	0
17607	0	0	0	0	0	0	0

	Binissalem	Bunyola	Búger	...	Santa Maria del Camí	Santanyí	Selva \
0	0	0	0	...	0	0	0
1	0	0	0	...	0	0	0
2	0	0	0	...	0	0	0
3	0	0	0	...	0	0	0
4	0	0	0	...	0	0	0
...
17603	0	0	0	...	0	0	0
17604	0	0	0	...	0	0	0
17605	0	0	0	...	0	0	1
17606	0	0	0	...	0	0	0
17607	0	0	0	...	0	0	0

	Sencelles	Ses Salines	Sineu	Son Servera	Sóller	Valldemossa \
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
...
17603	0	0	0	0	0	0
17604	0	0	0	0	0	0
17605	0	0	0	0	0	0
17606	0	0	0	0	0	0
17607	0	0	0	0	0	0

	Vilafranc de Bonany
0	0
1	0
2	0
3	0
4	0
...	...
17603	0
17604	0
17605	0
17606	0
17607	0

[17603 rows x 53 columns]

```
[39]: df = df.drop(columns = ['neighbourhood_cleansed'])
df = pd.concat([df, df_neigh], axis=1)
df.head()
```

```
[39]:      id  host_is_superhost  host_identity_verified  latitude  longitude \
0   11547.0              0.0                1.0   39.51888    2.48182
1   100831.0             0.0                1.0   39.76347    3.16255
2   105891.0             1.0                1.0   39.66044    3.07165
3   106833.0             0.0                1.0   39.61600    3.30121
4   130669.0             0.0                0.0   39.56478    2.60333

      accommodates  bedrooms  beds  price  minimum_nights  ... \
0              2.0        1.0   1.0   89.0              5.0  ...
1              8.0        4.0   7.0  175.0              7.0  ...
2              6.0        3.0   4.0  140.0              6.0  ...
3              4.0        2.0   4.0  200.0              5.0  ...
4              2.0        1.0   2.0  110.0              2.0  ...

      Santa María del Camí  Santanyí  Selva  Sencelles  Ses Salines  Sineu  \
0              0          0        0          0          0          0
1              0          0        0          0          0          0
2              0          0        0          0          0          0
3              0          0        0          0          0          0
4              0          0        0          0          0          0

      Son Servera  Sóller  Valldemossa  Vilafranc de Bonany
0              0        0          0          0
1              0        0          0          0
2              0        0          0          0
3              0        0          0          0
4              0        0          0          0
```

[5 rows x 208 columns]

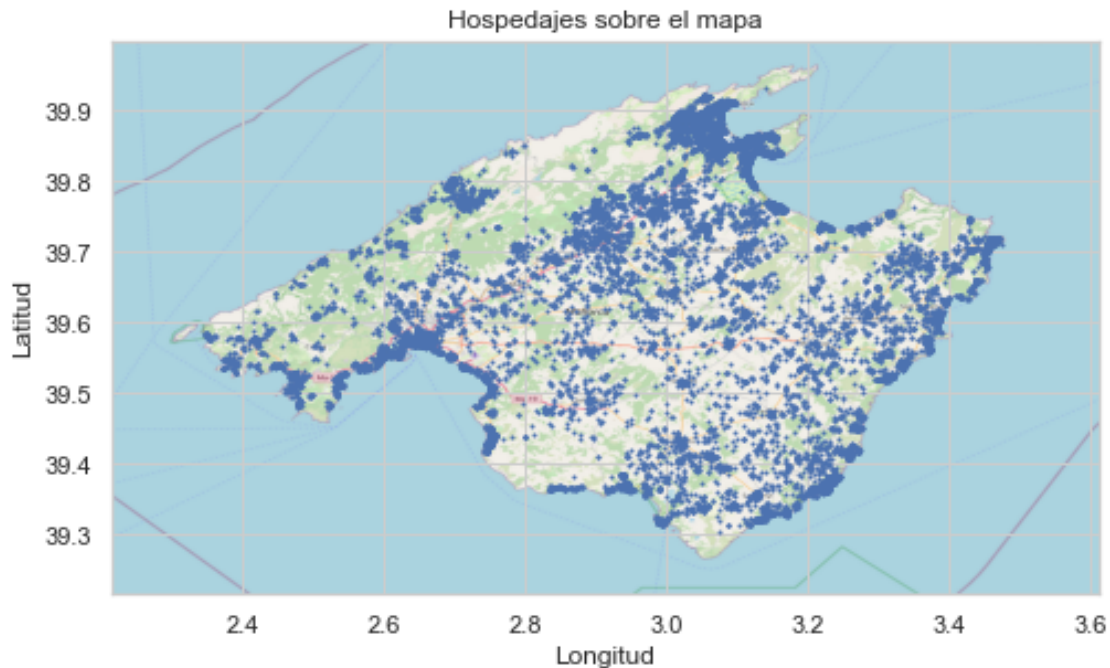
2.5 Columnas de latitud y longitud

El siguiente paso es ver si la ubicación precisa de los hospedajes tiene algún tipo de colinearidad con el precio. Para empezar vamos a ver la distribución de los alojamientos sobre la isla:

```
[40]: box = ((2.2165, 3.6118, 39.2152, 39.9982)) #lonmin, lonmax, latmin, latmax
mapping = plt.imread(path_img_map)
fig, ax = plt.subplots(figsize = (8,7))
```

```
ax.scatter(df.longitude, df.latitude, zorder=1, c='b', s=1)
ax.set_title('Hospedajes sobre el mapa')
ax.set_xlabel('Longitud')
ax.set_xlim(box[0],box[1])
ax.set_ylabel('Latitud')
ax.set_ylim(box[2],box[3])
ax.imshow(mapping, zorder=0, extent = box, aspect= 'equal')
```

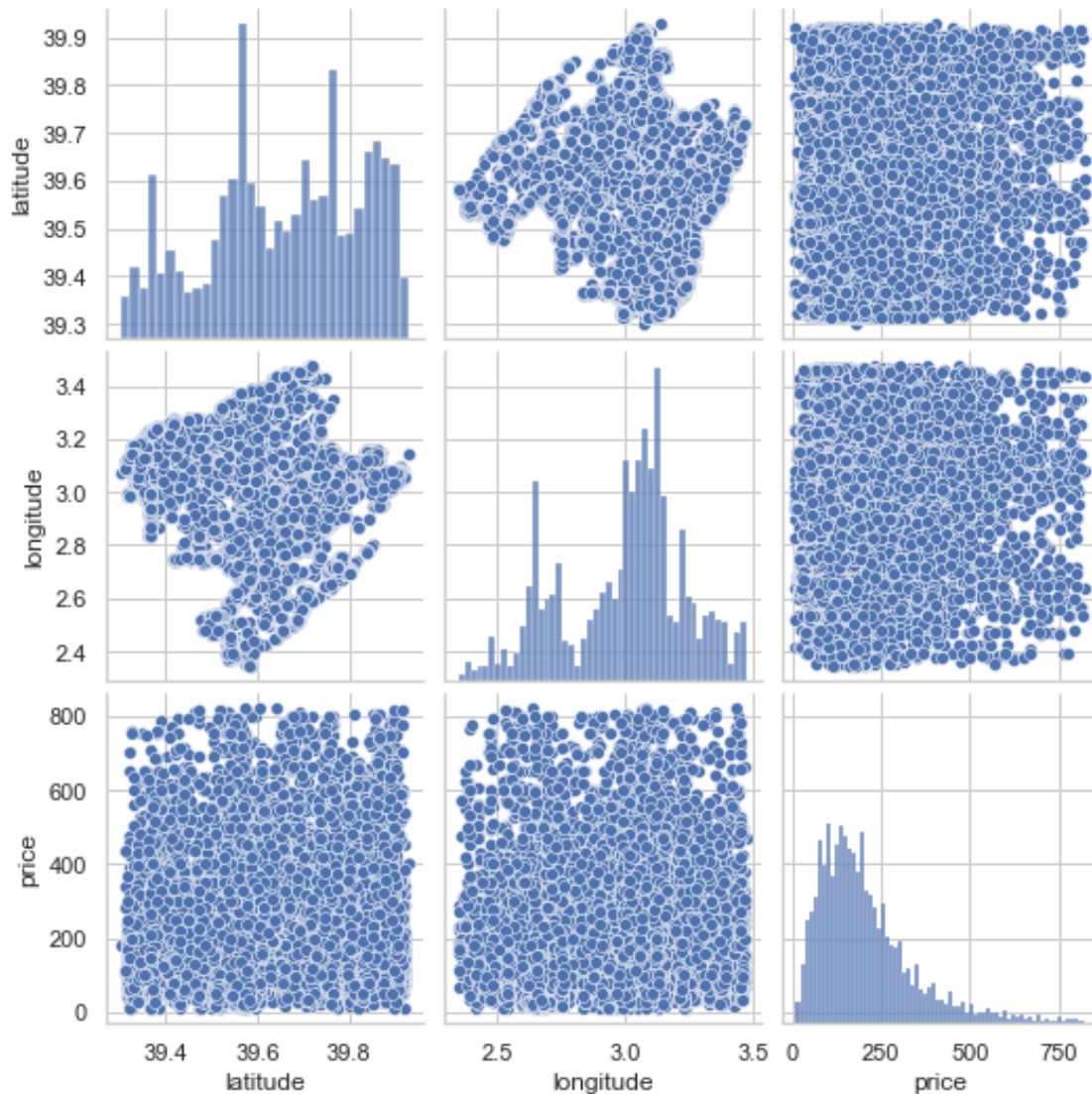
[40]: <matplotlib.image.AxesImage at 0x7f8d257dd910>



Vista la distribución general de los hospedajes en la isla, aunque hay algunos núcleos, no se puede decir que la distribución a lo largo de los ejes de latitud y longitud sea relevante. A diferencia de las concentraciones en algunas zonas, que probablemente se vean representadas por la agrupación por vecindarios mejor que por estas dos variables. Aún así veamos si hay algún tipo de relación entre ellas y el precio con un *pairplot*;

```
[41]: sns.pairplot(df[['latitude', 'longitude', 'price']])
```

[41]: <seaborn.axisgrid.PairGrid at 0x7f8d252dd2d0>



Como decíamos no se ve ninguna relación. Tampoco con el *pairplot*. Podemos decir que las columnas de latitud y longitud no tienen relevancia para el estudio, por tanto, las eliminamos.

```
[42]: df = df.drop(columns = ['latitude', 'longitude'])
```

Estudio final de valores nulos A continuación se presenta la información sobre el *dataset* ya modificado:

```
[43]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17603 entries, 0 to 17607
Columns: 206 entries, id to Vilafranc de Bonany
```

```
dtypes: float64(148), object(1), uint8(57)
memory usage: 21.6+ MB
```

Convertimos las variables al tipo más óptimo utilizando una función de la librería `pandas`: [convert_dtypes](#).

```
[44]: df.convert_dtypes()
```

```
[44]:
```

	id	host_is_superhost	host_identity_verified	accommodates	\
0	11547	0	1	2	
1	100831	0	1	8	
2	105891	1	1	6	
3	106833	0	1	4	
4	130669	0	0	2	
...	
17603	45489412	0	1	6	
17604	45489550	0	1	9	
17605	45493152	0	1	6	
17606	45496032	0	0	2	
17607	45499210	0	0	6	

	bedrooms	beds	price	minimum_nights	maximum_nights	\
0	1	1	89.0	5	60	
1	4	7	175.0	7	365	
2	3	4	140.0	6	365	
3	2	4	200.0	5	365	
4	1	2	110.0	2	365	
...	
17603	3	4	195.0	1	365	
17604	5	8	110.0	1	1125	
17605	3	4	179.0	1	365	
17606	1	1	42.0	1	7	
17607	2	4	100.0	5	1120	

	has_availability	...	Santa María del Camí	Santanyí	Selva	\
0	1	...	0	0	0	
1	1	...	0	0	0	
2	1	...	0	0	0	
3	1	...	0	0	0	
4	1	...	0	0	0	
...	
17603	1	...	0	0	0	
17604	1	...	0	0	0	
17605	1	...	0	0	1	
17606	1	...	0	0	0	
17607	1	...	0	0	0	

	Sencelles	Ses Salines	Sineu	Son Servera	Sóller	Valldemossa	\
--	-----------	-------------	-------	-------------	--------	-------------	---

0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
...
17603	0	0	0	0	0	0
17604	0	0	0	0	0	0
17605	0	0	0	0	0	0
17606	0	0	0	0	0	0
17607	0	0	0	0	0	0

Vilafranc de Bonany	
0	0
1	0
2	0
3	0
4	0
...	...
17603	0
17604	0
17605	0
17606	0
17607	0

[17603 rows x 206 columns]

Ahora que ya se ha visto la forma del dataset, falta hacer un último análisis de valores nulos.

```
[45]: nan_df = pd.DataFrame(df.isna().sum())
nan_df.reset_index(level = 0, inplace= True)
nan_df.columns = ['columnas', 'num_nans']
nan_df
```

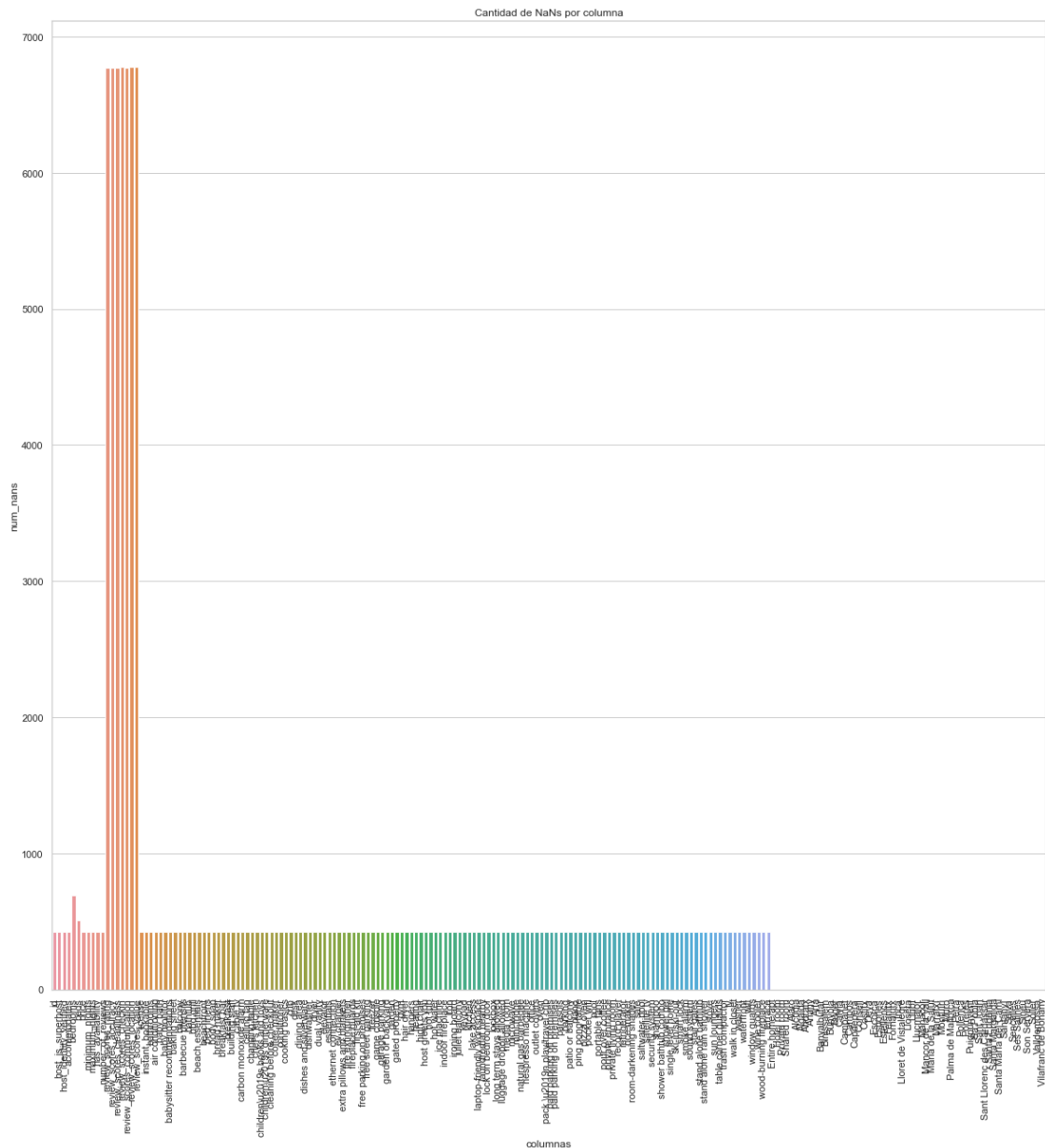
```
[45]:
```

	columnas	num_nans
0	id	422
1	host_is_superhost	424
2	host_identity_verified	424
3	accommodates	422
4	bedrooms	692
..
201	Sineu	0
202	Son Servera	0
203	Sóller	0
204	Valldemossa	0
205	Vilafranc de Bonany	0

[206 rows x 2 columns]

```
[46]: fig, ax = plt.subplots(figsize=(20, 20))
      ax = sns.barplot(x = nan_df.columnas, y= nan_df.num_nans, data=nan_df)
      ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
      ax.set_title('Cantidad de NaNs por columna')
```

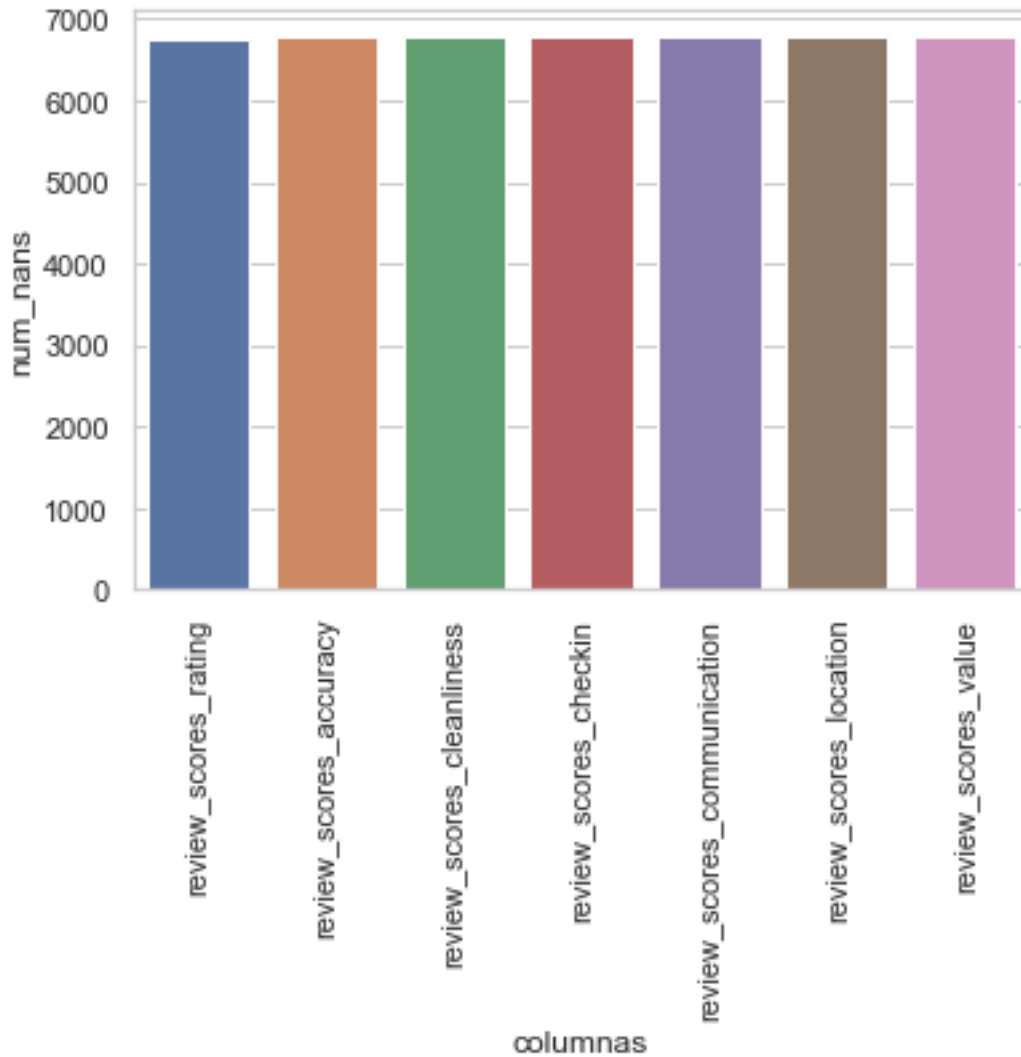
```
[46]: Text(0.5, 1.0, 'Cantidad de NaNs por columna')
```



Teniendo en cuenta la longitud del dataset, se puede identificar una serie de columnas que tienen

gran cantidad de nans y que, por tanto no aportan información. Se identifican a continuación utilizando un plot reducido:

```
[47]: fig, ax = plt.subplots()
      aux = nan_df.loc[nan_df.num_nans > 1000,:]
      ax = sns.barplot(x = aux.columnas, y= aux.num_nans, data=aux)
      _ = ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
```



```
[48]: aux.num_nans.max() / len(df) * 100
```

```
[48]: 38.493438618417315
```

Eliminar las filas con NaNs del dataset significaría perder un 40% de los datos. La mejor manera de realizar esta operación es eliminar las columnas, que aportan poca información al análisis.

```
[49]: columnas_a_borrar = nan_df.loc[nan_df.num_nans > 1000,:].columnas.tolist()
df = df.drop(columns = columnas_a_borrar)
df.head()
```

```
[49]:      id  host_is_superhost  host_identity_verified  accommodates \
0   11547.0                0.0                    1.0             2.0
1   100831.0                0.0                    1.0             8.0
2   105891.0                1.0                    1.0             6.0
3   106833.0                0.0                    1.0             4.0
4   130669.0                0.0                    0.0             2.0

      bedrooms  beds  price  minimum_nights  maximum_nights  has_availability \
0           1.0   1.0   89.0                5.0             60.0             1.0
1           4.0   7.0  175.0                7.0            365.0             1.0
2           3.0   4.0  140.0                6.0            365.0             1.0
3           2.0   4.0  200.0                5.0            365.0             1.0
4           1.0   2.0  110.0                2.0            365.0             1.0

      ...  Santa María del Camí Santanyí  Selva  Sencelles  Ses Salines  Sineu \
0  ...                                0      0      0          0          0
1  ...                                0      0      0          0          0
2  ...                                0      0      0          0          0
3  ...                                0      0      0          0          0
4  ...                                0      0      0          0          0

      Son Servera  Sóller  Valldemossa  Vilafranc de Bonany
0              0      0              0              0
1              0      0              0              0
2              0      0              0              0
3              0      0              0              0
4              0      0              0              0

[5 rows x 199 columns]
```

Ahora podemos borrar las filas con nans, que parecen coincidir en muchas de las columnas originales del dataset y que no son una cantidad de datos significativa con respecto al dataset.

```
[50]: df = df.dropna()
```

Tras la eliminación de NaNs lo comprobamos con el mismo gráfico con el que habíamos estimado la cantidad de NaNs:

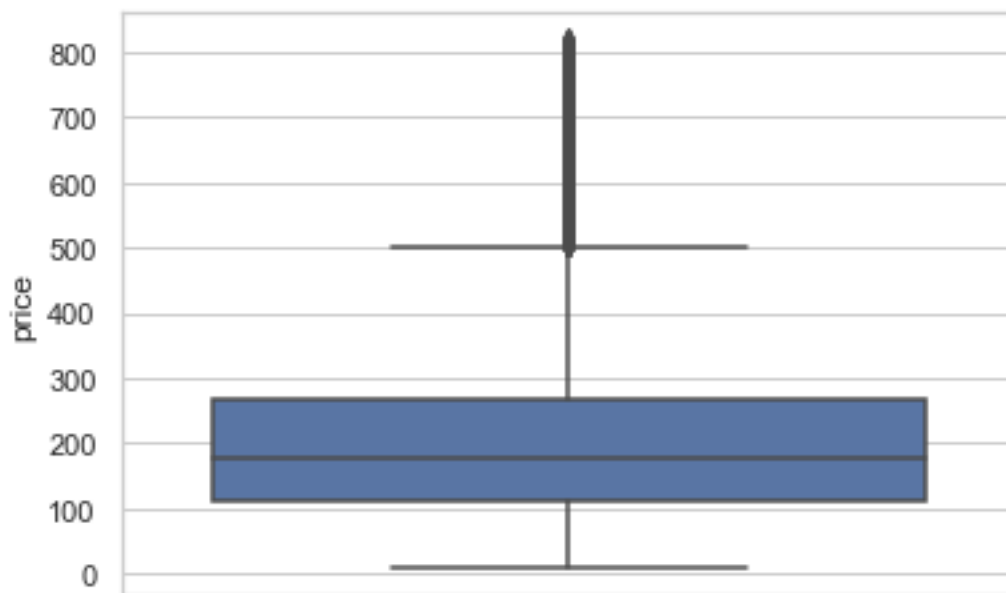
```
[51]: nan_df = pd.DataFrame(df.isna().sum())
nan_df.reset_index(level = 0, inplace= True)
nan_df.columns = ['columnas', 'num_nans']

fig, ax = plt.subplots(figsize=(20, 20))
ax = sns.barplot(x = nan_df.columnas, y= nan_df.num_nans, data=nan_df)
```



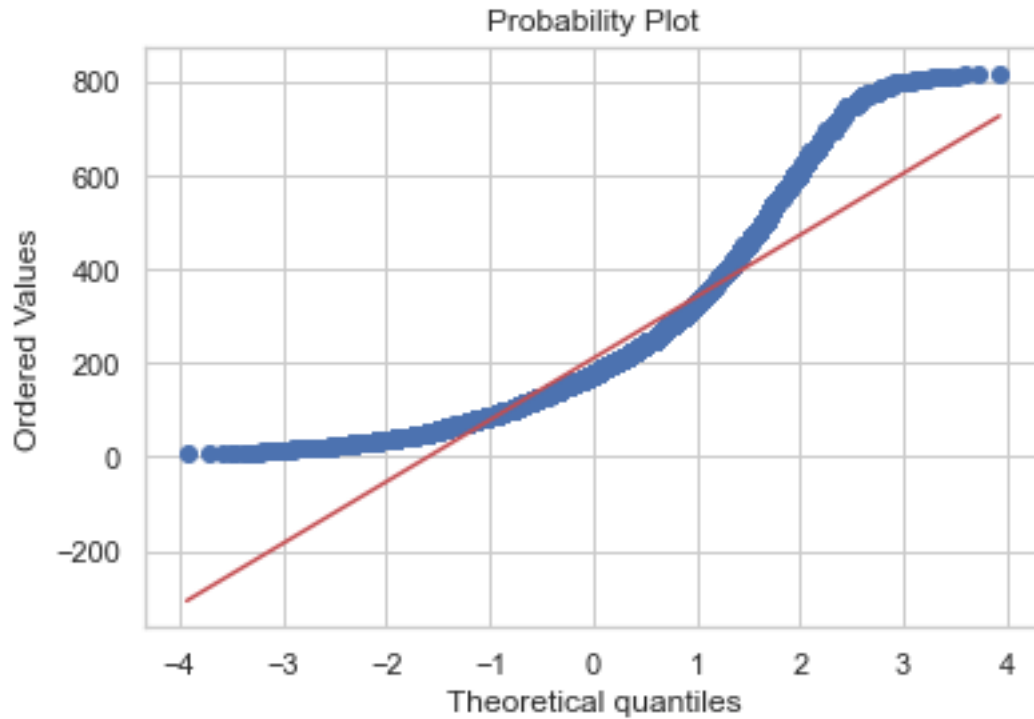
```
[52]: count    16419.00000
      mean      209.86629
      std       140.12414
      min        10.00000
      25%       111.00000
      50%       177.00000
      75%       267.00000
      max       819.00000
      Name: price, dtype: float64
```

```
[53]: sns.set_theme(style="whitegrid")
      ax = sns.boxplot(y="price", data=df)
```



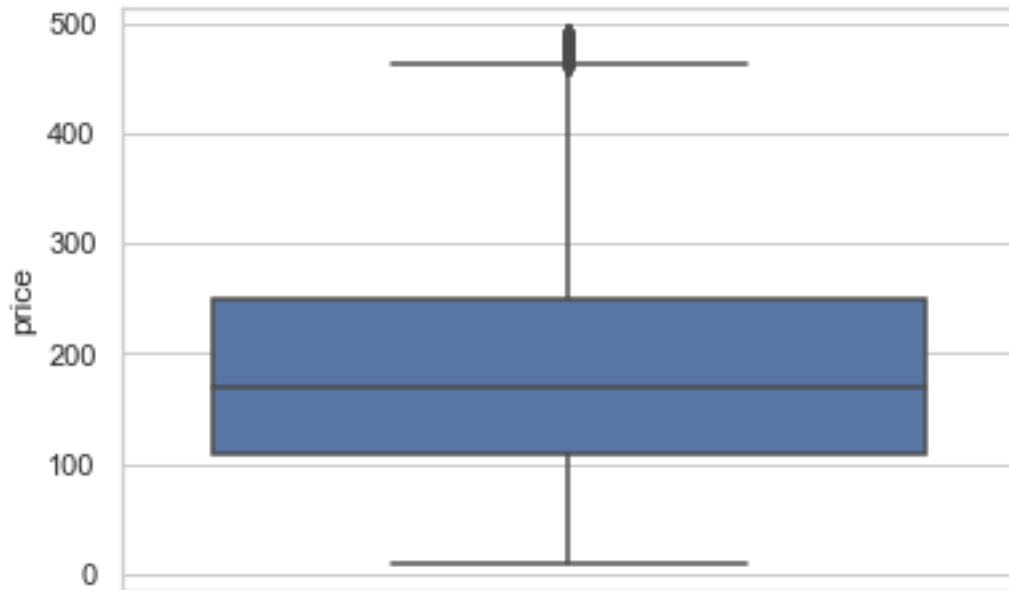
```
[54]: stats.probplot(df.price, plot=sns.mpl.pyplot)
```

```
[54]: ((array([-3.93146285, -3.71284682, -3.59311497, ...,  3.59311497,
              3.71284682,  3.93146285])),
      array([ 10.,  10.,  10., ..., 815., 819., 819.])),
      (131.5617713853184, 209.8662896644132, 0.9387085022662417))
```



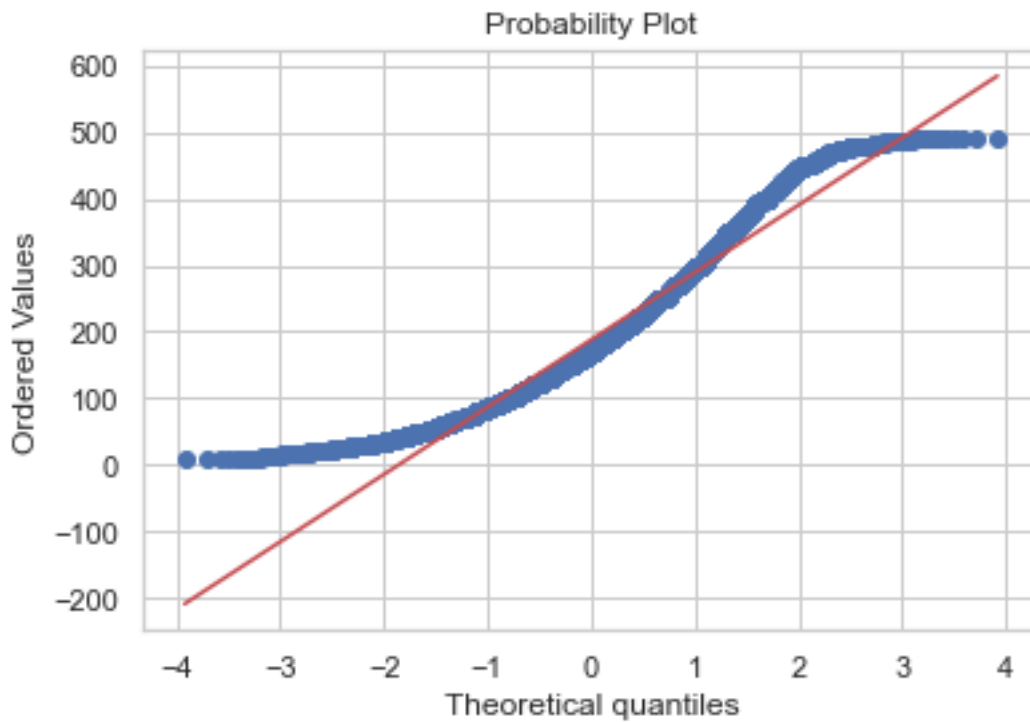
La distribución de precios continua en la misma línea, se pueden apreciar cambios poco significativos, como por ejemplo un descenso de la media (de 244 a 207). De ello se deduce que a pesar del tratamiento, el dataset conserva la estructura inicial, no así la estructura con respecto a la distribución por quantiles, lo que hace necesario un recorte mayor de *outliers*. En este caso nos quedamos únicamente con los datos

```
[55]: std = df.price.describe()[2]
dfB = df.loc[(df.price <= 3.5*std)]
sns.set_theme(style="whitegrid")
ax = sns.boxplot(y="price", data=dfB)
```



```
[56]: stats.probplot(dfB.price, plot=sns.mpl.pyplot)
```

```
[56]: ((array([-3.91854322, -3.6992463 , -3.5791117 , ...,  3.5791117 ,
              3.6992463 ,  3.91854322])),
      array([ 10.,  10.,  10., ..., 490., 490., 490.])),
      (101.24360619490804, 187.57238545080648, 0.9759210757637155))
```



Filtrando por más del 99% de los datos (tres veces y media la desviación estándar) conseguimos limitar el efecto de *outliers* y el ajuste a cuantiles teóricos en la distribución por cuantiles.

2.6 Escalado de datos: *train* y *test*

Ya por último se escalan los datos. Para este proceso separaremos en dos *datasets* uno para el entrenamiento y otro para el test. Haremos un entrenamiento de un modelo de escalado ([StandardScaler](#)), propio de la librería `sklearn` y lo aplicaremos a ambos.

En primer lugar haremos la separación entre los conjuntos de entrenamiento y test:

```
[57]: df_train, df_test = train_test_split(dfB, test_size= 0.33)
```

A continuación comprobamos que el análisis estadístico de ambos es similar.

```
[58]: df_train.price.describe()
```

```
[58]: count      10425.000000
      mean        187.536045
      std         103.588526
      min          10.000000
      25%         108.000000
      50%         170.000000
      75%         250.000000
      max         490.000000
      Name: price, dtype: float64
```

```
[59]: df_test.price.describe()
```

```
[59]: count      5136.000000
      mean        187.646149
      std         103.996544
      min          10.000000
      25%         108.000000
      50%         170.000000
      75%         249.000000
      max         490.000000
      Name: price, dtype: float64
```

Una vez realizada esta comprobación, se escalan los datos:

```
[60]: funcion_escalado = StandardScaler().fit(df_train) #entrenamiento con el dataset
      ↪ de entrenamiento

      df_train_scaled = funcion_escalado.transform(df_train)
```

```
df_test_scaled = funcion_escalado.transform(df_test)
```

```
[61]: df_train = pd.DataFrame(df_train_scaled, index=df_train.index, columns=df_train.  
    ↪columns)  
df_train.describe().price
```

```
[61]: count      1.042500e+04  
mean      -8.894564e-17  
std       1.000048e+00  
min       -1.713940e+00  
25%       -7.678443e-01  
50%       -1.692937e-01  
75%        6.030296e-01  
max        2.920000e+00  
Name: price, dtype: float64
```

```
[62]: df_test = pd.DataFrame(df_test_scaled, index=df_test.index, columns=df_test.  
    ↪columns)  
df_test.describe().price
```

```
[62]: count      5136.000000  
mean         0.001063  
std          1.003987  
min         -1.713940  
25%         -0.767844  
50%         -0.169294  
75%          0.593376  
max          2.920000  
Name: price, dtype: float64
```

Como se puede observar en las dos cajas anteriores, la media tiende a cero en ambos casos y el resto de valores son aproximadamente iguales en ambos conjuntos de datos.

Con este último tratamiento se cierra el preprocesado y podemos pasar a la selección, entrenamiento y validación del modelo.

3 Modelización

En este apartado utilizaremos los conjuntos ya procesados para la selección del modelo y su entrenamiento. El tipo de modelo seleccionado para este ejercicio es un modelo de regresión.

A lo largo de este apartado se aplicarán distintos modelos de regresión a fin de utilizar distintos modelos y poder compararlos. Para esta comparación se utilizarán distintas métricas útiles en modelos de regresión.

Con el fin de poder explicar el funcionamiento de las métricas, se aplicará un modelo simple. Este modelo simple también servirá como punto de partida para seleccionar modelos más complejos y

poder compararlos entre ellos. Los modelos y las métricas utilizadas se extraerán de la librería de `sklearn`.

3.1 Separación en entradas y salidas

A continuación separamos entre entradas y salidas en ambos conjuntos:

```
[63]: nombres_columnas = df_train.columns
X_train = df_train.loc[:, nombres_columnas != 'price'].to_numpy()
y_train = df_train.loc[:, nombres_columnas == 'price'].to_numpy()
X_test = df_test.loc[:, nombres_columnas != 'price'].to_numpy()
y_test = df_test.loc[:, nombres_columnas == 'price'].to_numpy()

X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
[63]: ((10425, 198), (10425, 1), (5136, 198), (5136, 1))
```

3.2 Aplicación de un modelo simple

En primer lugar se aplicará un modelo lineal basado en la minimización del cuadrado de la suma de los residuos, [LinearRegression](#). A continuación se presenta la ecuación de minimización que realiza el algoritmo:

$$\min_w ||Xw - y||_2^2$$

Esta formula también podría escribirse como:

$$\min_w ||y_p(w) - y||_2^2$$

El entrenamiento de este modelo devuelve un conjunto de pesos w , que, multiplicado por la entrada X , da lugar a la salida predecida y_p . Estos corresponden a la relevancia de las distintas variables al predecir la salida y en base a la entrada.

```
[64]: reg = LinearRegression()
reg.fit(X_train, y_train)
y_predict = reg.predict(X_test)
```

3.3 Métricas para modelos de regresión

Utilizando la técnica de *cross-validation*[\[2\]](#) comparamos la predicción del modelo con la predicción esperada del conjunto de test. Para esto, deberemos utilizar métricas que nos permitan calificar los distintos ajustes. De entre las métricas más adecuadas para modelos de regresión, se han escogido las métricas más utilizadas. Estas métricas son: * *Mean absolute error* o error absoluto medio * *Mean squared error* o error cuadrado medio * *Median absolute error* o error absoluto mediano * R^2 o coeficiente de determinación

[[2]]Pese a que se puede implementar la técnica mediante una función propia de la librería `sklearn` se ha decidido implementar “a mano” el cálculo de errores con las métricas de la misma librería, para afianzar los conocimientos adquiridos durante el curso.

A continuación se presentan las definiciones y significados de las métricas seleccionadas.

3.3.1 *Mean absolute error (MAE)* o error absoluto medio

Esta métrica calcula la media aritmética de los errores cometidos por el modelo al comparar los valores esperados con los valores predichos por el modelo. Este error se da en la misma escala que los datos originales y por tanto hace una estimación directa del error cometido al predecir. Se puede calcular con la siguiente fórmula:

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|$$

Dónde y_i son los valores esperados y \hat{y}_i los valores predichos por el modelo.

Para la implementación en el código se utiliza la función `mean_absolute_error`.

3.3.2 *Mean squared error (MSE)* o error cuadrático medio

Este término se refiere a la media del error cuadrático cometido por el modelo entre la variable esperada y y la variable predicha \hat{y} . Se calcula con la formula siguiente:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2$$

Para la implementación en el código se utiliza la función `mean_squared_error`.

3.3.3 R^2 o coeficiente de determinación

Este coeficiente determina la calidad del modelo para replicar los resultados, y la proporción de variación de los resultados que puede explicarse por este. Habitualmente esta métrica se presenta en el rango de cero a uno o expresado en tanto por ciento. El valor igual a uno explica una calidad excelente, el modelo se ajusta perfectamente a los datos. En cambio, valores cercanos a cero determinan un fracaso completo del modelo para estos datos.

Para la implementación en el código se utiliza la función `r2_score`.

Para el almacenamiento de los valores de las métricas aplicadas a distintos modelos se define un *dataframe* inicialmente vacío con los nombres de las columnas iguales a las siglas de las métricas:

```
[65]: df_error = pd.DataFrame(columns=['modelo', 'MAE', 'MSE', 'R2'])  
  
df_error
```



```
[65]: Empty DataFrame
      Columns: [modelo, MAE, MSE, R2]
      Index: []
```

También se define una función para el almacenamiento de los errores en el dataframe:

```
[66]: def function_error_save(nombre_modelo, y_predict, y_test = y_test):
      '''
      Función para la adición de valores en el dataframe de error.
      nombre_modelo:: el nombre del modelo que se ha aplicado para obtener
      ↪ y_predict
      y_predict:: conjunto de salidas predecidas
      y_test:: conjunto de salidas esperadas
      df_error:: dataframe para almacenamiento de errores
      '''
      df_error_part = pd.DataFrame( data = {'modelo': nombre_modelo, 'MAE':
      ↪ mean_absolute_error(y_test,y_predict), 'MSE':
      ↪ mean_squared_error(y_test,y_predict), 'R2': r2_score(y_test,y_predict)}, index=
      ↪ [0])

      return df_error_part
```

Por tanto para almacenar los datos del modelo aplicado anteriormente (regresión lineal) deberemos aplicar la función y esta devolverá el *dataframe* de errores actualizado:

```
[67]: df_error = df_error.append(function_error_save('LinearRegression', y_test,
      ↪ y_predict))
      df_error
```

```
[67]:
```

	modelo	MAE	MSE	R2
0	LinearRegression	1.913400e+11	7.982741e+25	-0.000077

Como se puede observar, los valores de errores son muy grandes. Por ello vamos a realizar pruebas con otros modelos y a aplicar la función para el almacenamiento de las métricas según el modelo.

3.4 Aplicación de modelos más complejos

Se han decidido aplicar dos modelos más complejos. Estos son: * Regresión *Ridge* * Regresión lineal generalizada

A continuación explicaremos cada una de ellas y se realizará el cálculo de los errores cometidos por ambas.

3.4.1 Regresión *Ridge*

Este modelo resuelve algunos de los problemas del modelo de regresión lineal clásico añadiendo una penalización al tamaño de los pesos según su valor. Esto lo hace minimizando la función:

$$\min_w ||Xw - y||_2^2 + \alpha ||w||_2^2$$

Dónde α es la penalización sobre los pesos w .

```
[68]: ridge = Ridge()
      ridge.fit(X_train, y_train)
      y_predict_ridge = ridge.predict(X_test)
      ridge.coef_
```

```
[68]: array([[ -7.74888413e-03, -2.60039327e-02,  3.37416185e-02,
         3.20663681e-01,  9.40793319e-02, -8.72983503e-02,
        -1.90451144e-02,  2.51045830e-02,  0.00000000e+00,
        -1.26280276e-01,  1.43843821e-01, -2.93780595e-02,
         3.13165093e-01, -8.52095932e-03, -1.04924198e-02,
         1.14801570e-02,  5.95952658e-03,  3.28011184e-03,
        -1.21021814e-04,  6.62745324e-03, -6.64985408e-03,
        -1.22880869e-04,  7.34210992e-03,  5.35419880e-03,
        -8.05651730e-03,  8.10078122e-03,  1.66754117e-03,
        -1.21021814e-04,  1.84222122e-02,  8.66035649e-04,
        -4.24620318e-03,  1.95758139e-03, -1.12634655e-02,
         5.99468500e-03,  0.00000000e+00, -4.03662499e-03,
         6.29304386e-03,  1.76722434e-02,  8.53685159e-03,
         4.35954522e-03,  4.18674801e-03,  1.20819865e-02,
        -1.21021814e-04,  0.00000000e+00,  1.06952379e-02,
         3.09817449e-03,  6.65075974e-03, -1.21021814e-04,
         1.66832343e-02,  9.74944901e-03,  1.34438396e-02,
         8.95790693e-03, -1.15918543e-02,  1.05267259e-02,
        -1.36195578e-03,  1.63865751e-03, -6.64154677e-03,
        -3.44814675e-03,  1.43725390e-02, -1.20830619e-02,
         0.00000000e+00,  7.70763298e-03,  0.00000000e+00,
        -1.21021814e-04,  5.70497554e-03,  2.34618885e-03,
        -1.01746814e-02,  5.47693805e-03,  1.47694395e-02,
        -5.79179626e-03,  2.65129830e-03,  3.68871837e-03,
         0.00000000e+00, -1.82105939e-02,  3.62558175e-03,
         0.00000000e+00,  0.00000000e+00, -4.03651955e-03,
         3.94779642e-03, -3.32949680e-03,  8.64749971e-03,
         3.65417464e-03, -7.28303046e-03,  1.45937969e-03,
        -4.48904691e-03,  6.36864935e-03,  0.00000000e+00,
        -3.21348551e-03, -1.14036892e-02, -1.21021814e-04,
         1.84665245e-03,  0.00000000e+00,  4.28528576e-03,
        -1.38297833e-02,  3.85426118e-03, -5.46209626e-03,
         2.13811093e-03,  0.00000000e+00, -1.21021814e-04,
        -1.62269507e-02, -3.98065029e-03, -1.21021814e-04,
```

```

-1.21021814e-04, -9.40061694e-03, 6.00508436e-04,
-1.21739325e-02, 1.92726138e-02, -3.56937874e-05,
2.14072124e-03, -3.85696994e-03, 1.34144185e-02,
-8.59299340e-04, -3.40531636e-03, 0.00000000e+00,
0.00000000e+00, -1.21021814e-04, 0.00000000e+00,
9.38420765e-03, 0.00000000e+00, -1.84123492e-02,
1.74557420e-03, 1.01190581e-02, 4.99845207e-03,
-3.27494534e-03, -5.65392320e-03, -1.67568560e-02,
-1.21021814e-04, -1.21021814e-04, -1.65679493e-02,
-1.21021814e-04, 5.59448948e-03, -6.35541350e-03,
-5.63950821e-03, 0.00000000e+00, -1.88774349e-03,
-1.13816922e-02, 4.20086102e-03, 9.40886014e-03,
-1.21021814e-04, -1.21021814e-04, -1.21021814e-04,
1.69797331e-03, 6.87967123e-02, -2.75233263e-02,
-9.01609702e-03, 3.02502251e-03, -1.27187353e-03,
5.14520749e-03, 2.48975670e-02, -2.39801923e-02,
1.36085036e-02, -5.17556396e-03, 2.94030674e-03,
1.10251651e-02, -9.80091209e-03, 2.82202940e-02,
-6.52063752e-03, -1.16366755e-02, 3.93384060e-03,
-3.00012813e-03, 2.09051217e-03, 4.85780592e-02,
-9.49353833e-03, 1.08499489e-02, -2.80145365e-02,
-1.81138255e-02, 2.28351791e-02, -6.32122214e-03,
1.32085899e-02, -1.21806268e-02, 6.85840412e-03,
-1.37966297e-02, -6.04534859e-03, -1.14005054e-02,
-3.16776343e-02, 1.80784058e-02, -1.51072443e-02,
1.94261092e-02, 6.68748979e-03, -2.03334327e-02,
9.06315266e-03, 8.87081723e-03, -1.09815657e-03,
-9.73137968e-03, -6.23310411e-03, 1.87198130e-02,
6.50920335e-04, -3.81091588e-02, 2.34372107e-03,
-7.35742316e-03, 1.68011308e-02, -5.57925170e-03,
-1.90409442e-02, -3.34549703e-02, -9.25264281e-03,
3.09935002e-02, 1.70337014e-02, -4.67396039e-03]])

```

```

[69]: df_error = df_error.append(function_error_save('Ridge', y_test,
    ↪y_predict_ridge))
df_error

```

```

[69]:
      modelo      MAE      MSE      R2
0  LinearRegression  1.913400e+11  7.982741e+25 -0.000077
0           Ridge    5.490676e-01  5.604845e-01 -0.020668

```

En este caso el error cometido es mucho menor y aunque R^2 esté dando valores extremadamente raros. Veamos otros modelos.

3.4.2 Regresión lineal generalizada

Este modelo es una extensión del modelo de regresión lineal. Los valores previstos se calculan a partir de una combinación lineal de las variables de entrada, mediante una función de enlace inversa llamada h :

$$\hat{y}(w, X) = h(Xw)$$

Además, la función a minimizar depende de una función de distribución de la familia exponencial (Normal, Poisson, Gamma, etc.). Esta dependencia queda explicitada por la función de desviación unitaria d . Por tanto el valor a minimizar vendrá dado por:

$$\min_w \frac{1}{2n_{\text{samples}}} \sum_i d(y_i, \hat{y}_i) + \frac{\alpha}{2} \|w\|_2,$$

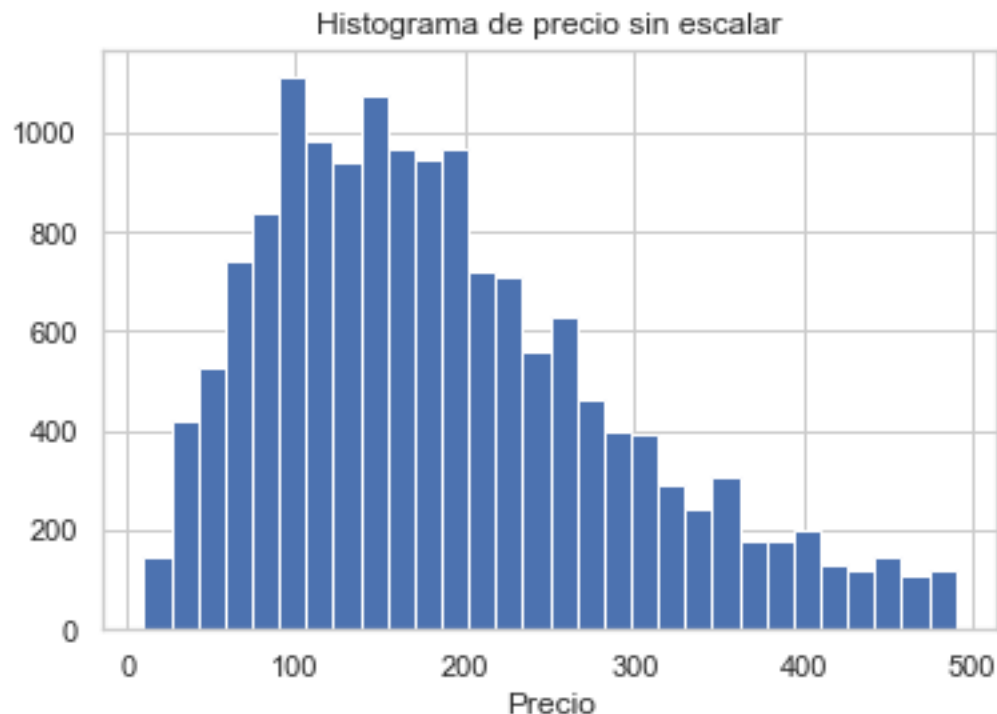
Esta función a minimizar incluye la penalización de la función anterior sobre los pesos. De esta manera estamos aplicando un modelo más complejo que el anterior, pero manteniendo sus ventajas.

Para la implementación de este modelo se utiliza la función [TweedieRegressor](#).

Para realizar una selección de la función de densidad coherente deberemos ver cómo es histograma de los datos según el precio:

```
[70]: plt.hist(dfB.price, bins=30)
plt.title('Histograma de precio sin escalar')
plt.xlabel('Número de datos')
plt.xlabel('Precio')
```

```
[70]: Text(0.5, 0, 'Precio')
```



La función que más se aproxima a la forma de la densidad de datos es la de una normal, por tanto seleccionamos el parámetro “power” igual a 1.

```
[71]: normal = TweedieRegressor(power=0)
normal.fit(X_train, y_train)
y_predict_normal = normal.predict(X_test)
df_error = df_error.append(function_error_save('Tweedie - Normal', y_test,
↪y_predict_normal))
df_error
```

```
[71]:
```

	modelo	MAE	MSE	R2
0	LinearRegression	1.913400e+11	7.982741e+25	-0.000077
0	Ridge	5.490676e-01	5.604845e-01	-0.020668
0	Tweedie - Normal	5.729425e-01	5.840309e-01	-0.852748

4 Análisis de errores

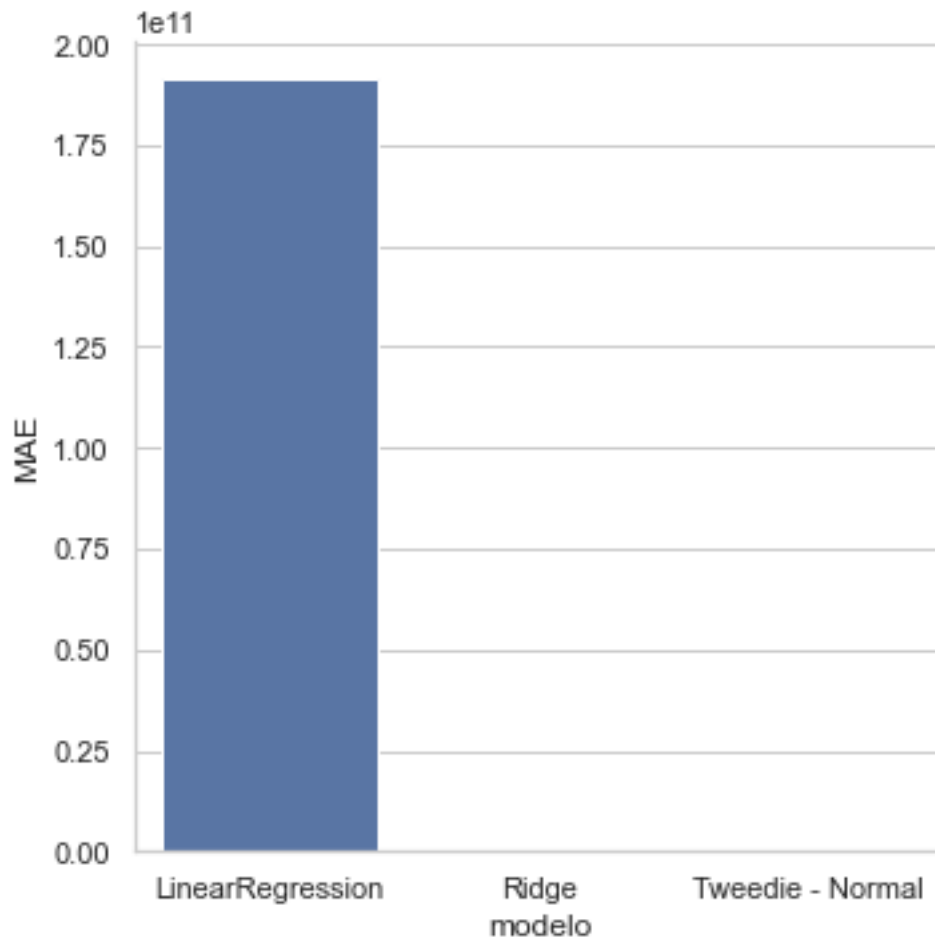
Gracias a la visualización del dataframe de los errores hemos podido ver cómo la métrica R^2 no ha sido útil en este caso, llegando a dar valores negativos, mientras que el MAE y el MSE se han ido modificando. A continuación visualizaremos los errores eliminando los valores de R^2 ya que no son útiles.

```
[72]: df_error = df_error[['modelo', 'MAE', 'MSE']]
```

Se empieza por el MAE:

```
[73]: sns.catplot(data=df_error, kind="bar", x='modelo', y='MAE')
```

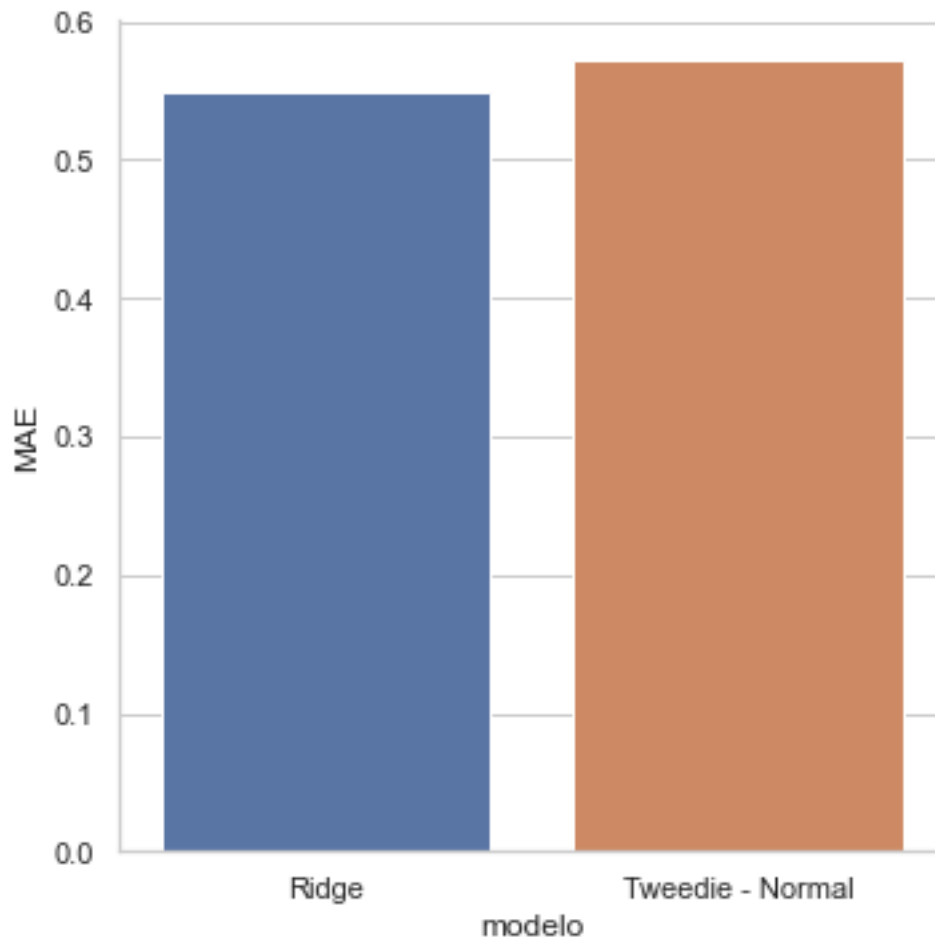
```
[73]: <seaborn.axisgrid.FacetGrid at 0x7f8d0e4bba50>
```



Como podemos ver, la regresion inicial comete errores muy grandes, así que vamos a descartarla a partir de este momento.

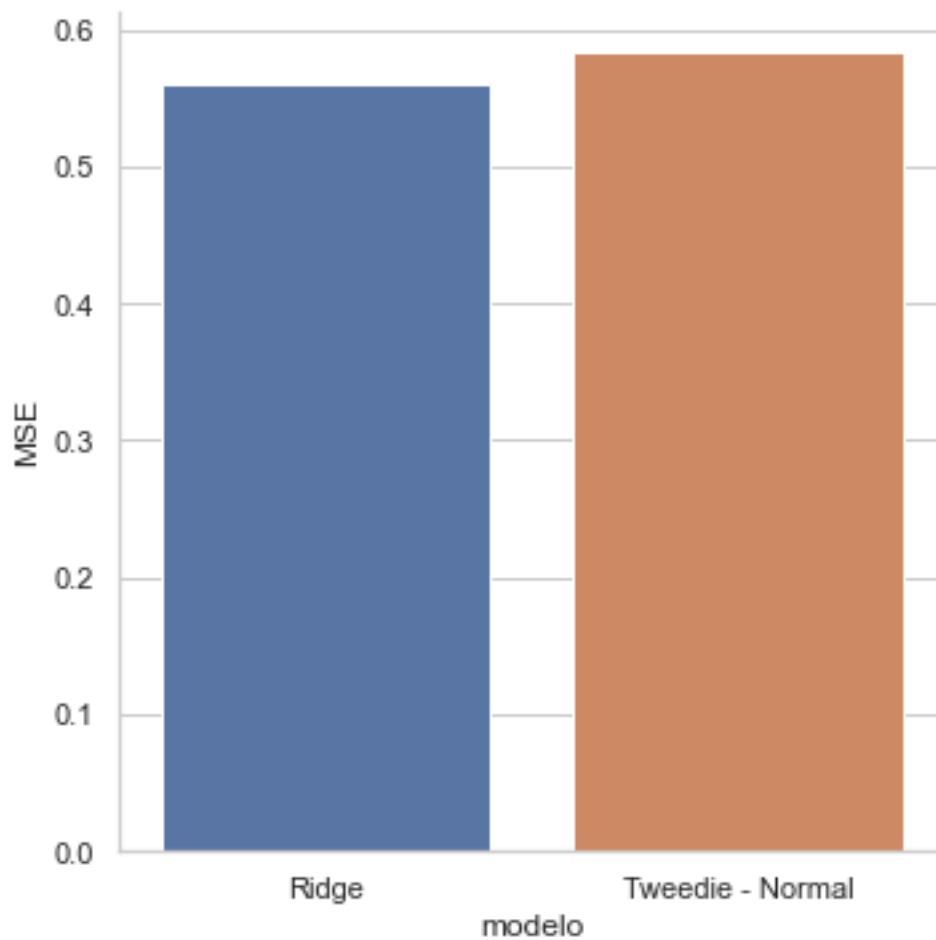
```
[74]: df_error = df_error.loc[df_error.modelo != 'LinearRegression',:]  
sns.catplot(data=df_error, kind="bar", x='modelo', y='MAE')
```

```
[74]: <seaborn.axisgrid.FacetGrid at 0x7f8d0e65a390>
```



```
[75]: sns.catplot(data=df_error, kind="bar", x='modelo', y='MSE')
```

```
[75]: <seaborn.axisgrid.FacetGrid at 0x7f8d0e65f710>
```



5 Resultado

Como se puede observar el modelo *Ridge* es el que da mejores resultados con ambas métricas. Por tanto es el modelo elegido para hacer la regresión del dataset de AirBNB.