

# final\_AA

February 16, 2021

**Autora:** Helena Antich Homar

**Asignatura:** Aprendizaje automático

**Ejercicio:** Práctica final: *Dataset* de Airbnb **Fecha de entrega:** 17 Febrero de 2021

**Descripción del programa:** Este programa analiza y limpia los datos del fichero de Airbnb

## 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

[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<sup>1</sup>

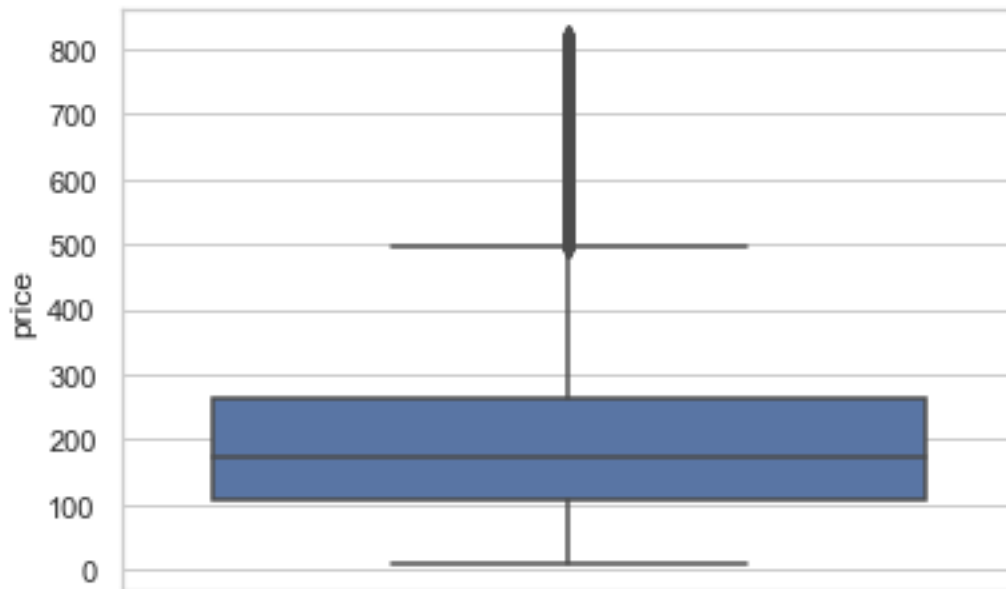
Teniendo en cuenta la desviación estándar, vamos a filtrar el dataframe para eliminar outliers analizando la distribución estadística del precio (boxplot) y la distribución por cuantiles (probplot).

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

---

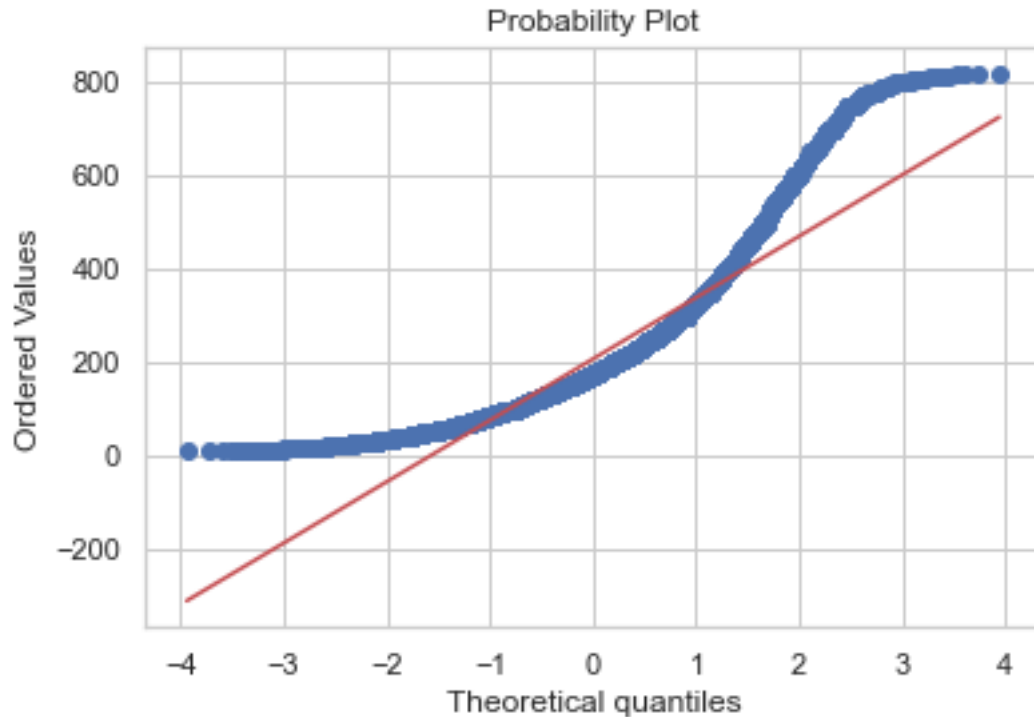
<sup>1</sup>A lo largo del procesado de las columnas se ha ido realizando este estudio a modo de monitorización del dataset.

```
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	<a href="https://www.airbnb.com/rooms/11547">https://www.airbnb.com/rooms/11547</a>	20200919153121	2020-09-21	
1	100831	<a href="https://www.airbnb.com/rooms/100831">https://www.airbnb.com/rooms/100831</a>	20200919153121	2020-09-21	
2	105891	<a href="https://www.airbnb.com/rooms/105891">https://www.airbnb.com/rooms/105891</a>	20200919153121	2020-09-20	
3	106833	<a href="https://www.airbnb.com/rooms/106833">https://www.airbnb.com/rooms/106833</a>	20200919153121	2020-09-20	
4	130669	<a href="https://www.airbnb.com/rooms/130669">https://www.airbnb.com/rooms/130669</a>	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:

```
[22]: 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:

```
[23]: 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:

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

```
[24]: 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)

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

```
[25]: 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”.

```
[26]: 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)

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

[27]: 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.

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

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

```
[29]:
```

	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.

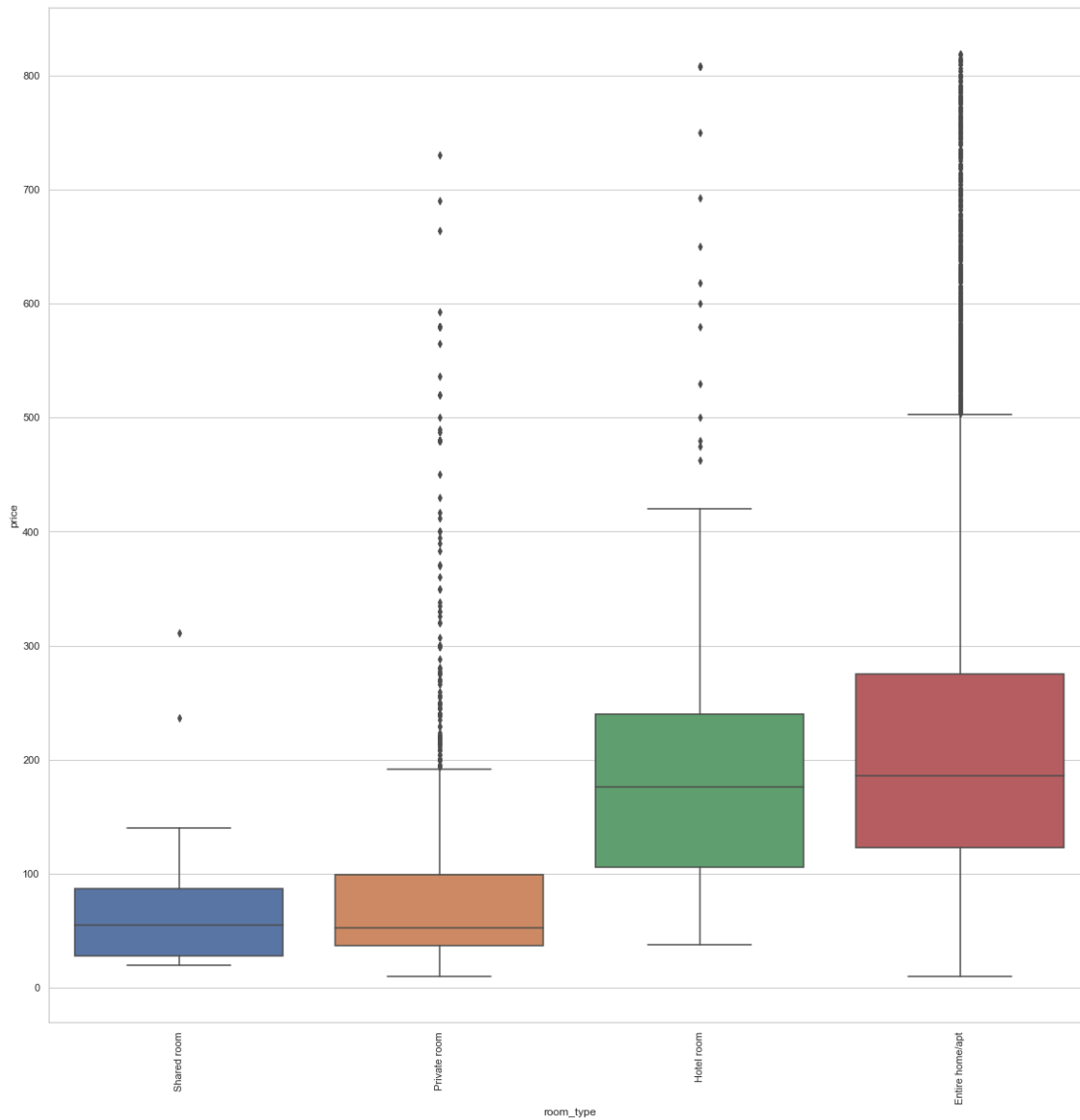
En primer lugar analizamos `room_type`:

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

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

```
[31]: 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)
```

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



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

```
[32]:
```

	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.

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

```
[33]:      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`:

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

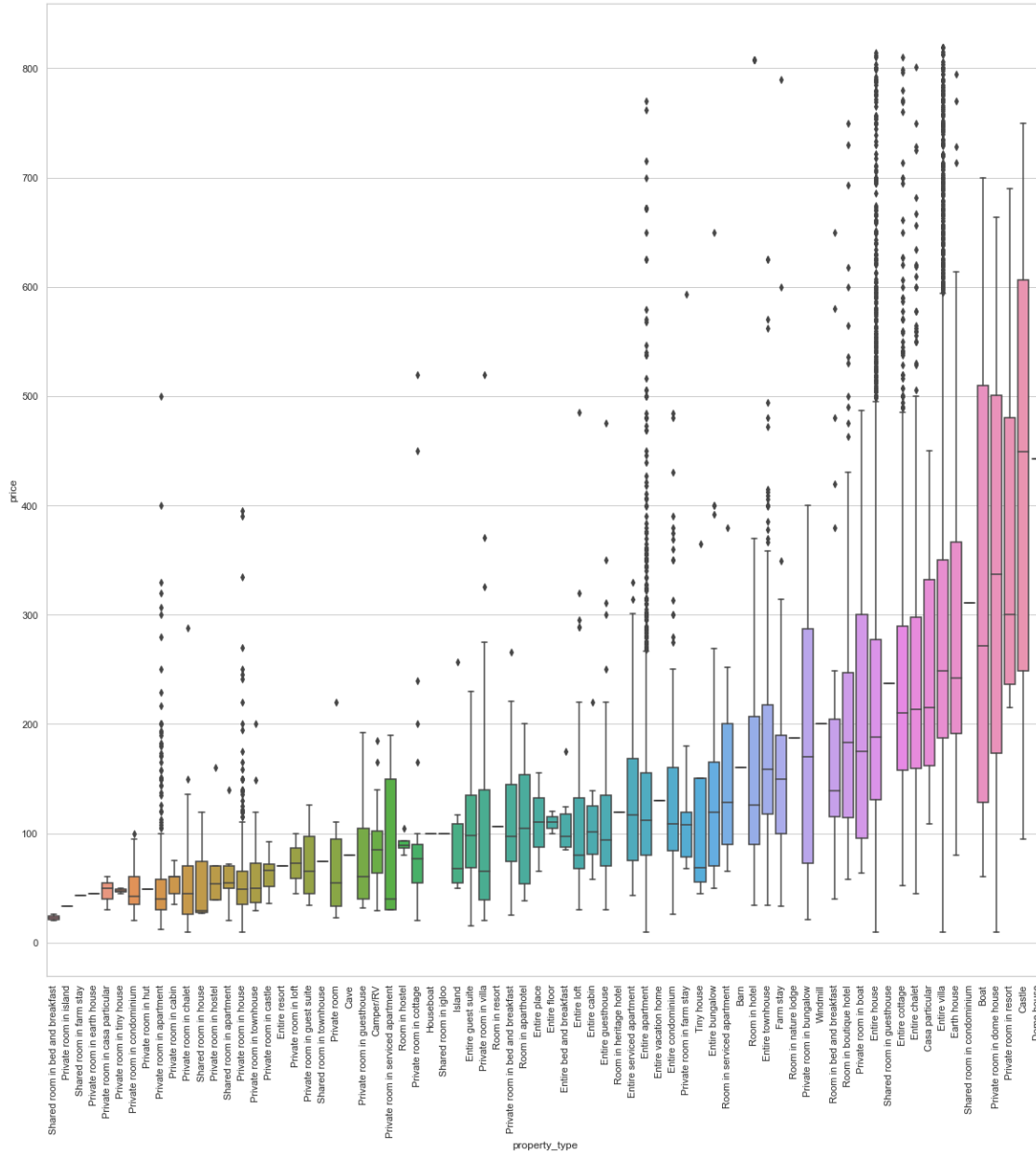
```
[34]: array(['Entire apartment', 'Entire house', 'Entire townhouse',  
        'Entire villa', 'Private room in apartment', 'Entire guesthouse',  
        'Private room in guest suite', 'Private room in villa',  
        'Entire loft', 'Entire cottage', 'Castle', 'Entire condominium',  
        'Room 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)
```

```
[35]: 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)  
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
```

```
[35]: [Text(0, 0, 'Shared room in bed and breakfast'),  
      Text(1, 0, 'Private room in island'),
```

Text(2, 0, 'Shared room in farm stay'),  
Text(3, 0, 'Private room in earth house'),  
Text(4, 0, 'Private room in casa particular'),  
Text(5, 0, 'Private room in tiny house'),  
Text(6, 0, 'Private room in condominium'),  
Text(7, 0, 'Private room in hut'),  
Text(8, 0, 'Private room in apartment'),  
Text(9, 0, 'Private room in cabin'),  
Text(10, 0, 'Private room in chalet'),  
Text(11, 0, 'Shared room in house'),  
Text(12, 0, 'Private room in hostel'),  
Text(13, 0, 'Shared room in apartment'),  
Text(14, 0, 'Private room in house'),  
Text(15, 0, 'Private room in townhouse'),  
Text(16, 0, 'Private room in castle'),  
Text(17, 0, 'Entire resort'),  
Text(18, 0, 'Private room in loft'),  
Text(19, 0, 'Private room in guest suite'),  
Text(20, 0, 'Shared room in townhouse'),  
Text(21, 0, 'Private room'),  
Text(22, 0, 'Cave'),  
Text(23, 0, 'Private room in guesthouse'),  
Text(24, 0, 'Camper/RV'),  
Text(25, 0, 'Private room in serviced apartment'),  
Text(26, 0, 'Room in hostel'),  
Text(27, 0, 'Private room in cottage'),  
Text(28, 0, 'Houseboat'),  
Text(29, 0, 'Shared room in igloo'),  
Text(30, 0, 'Island'),  
Text(31, 0, 'Entire guest suite'),  
Text(32, 0, 'Private room in villa'),  
Text(33, 0, 'Room in resort'),  
Text(34, 0, 'Private room in bed and breakfast'),  
Text(35, 0, 'Room in aparthotel'),  
Text(36, 0, 'Entire place'),  
Text(37, 0, 'Entire floor'),  
Text(38, 0, 'Entire bed and breakfast'),  
Text(39, 0, 'Entire loft'),  
Text(40, 0, 'Entire cabin'),  
Text(41, 0, 'Entire guesthouse'),  
Text(42, 0, 'Room in heritage hotel'),  
Text(43, 0, 'Entire serviced apartment'),  
Text(44, 0, 'Entire apartment'),  
Text(45, 0, 'Entire vacation home'),  
Text(46, 0, 'Entire condominium'),  
Text(47, 0, 'Private room in farm stay'),  
Text(48, 0, 'Tiny house'),

Text(49, 0, 'Entire bungalow'),  
Text(50, 0, 'Room in serviced apartment'),  
Text(51, 0, 'Barn'),  
Text(52, 0, 'Room in hotel'),  
Text(53, 0, 'Entire townhouse'),  
Text(54, 0, 'Farm stay'),  
Text(55, 0, 'Room in nature lodge'),  
Text(56, 0, 'Private room in bungalow'),  
Text(57, 0, 'Windmill'),  
Text(58, 0, 'Room in bed and breakfast'),  
Text(59, 0, 'Room in boutique hotel'),  
Text(60, 0, 'Private room in boat'),  
Text(61, 0, 'Entire house'),  
Text(62, 0, 'Shared room in guesthouse'),  
Text(63, 0, 'Entire cottage'),  
Text(64, 0, 'Entire chalet'),  
Text(65, 0, 'Casa particular'),  
Text(66, 0, 'Entire villa'),  
Text(67, 0, 'Earth house'),  
Text(68, 0, 'Shared room in condominium'),  
Text(69, 0, 'Boat'),  
Text(70, 0, 'Private room in dome house'),  
Text(71, 0, 'Private room in resort'),  
Text(72, 0, 'Castle'),  
Text(73, 0, 'Dome house')]



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.

```
[36]: 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.

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

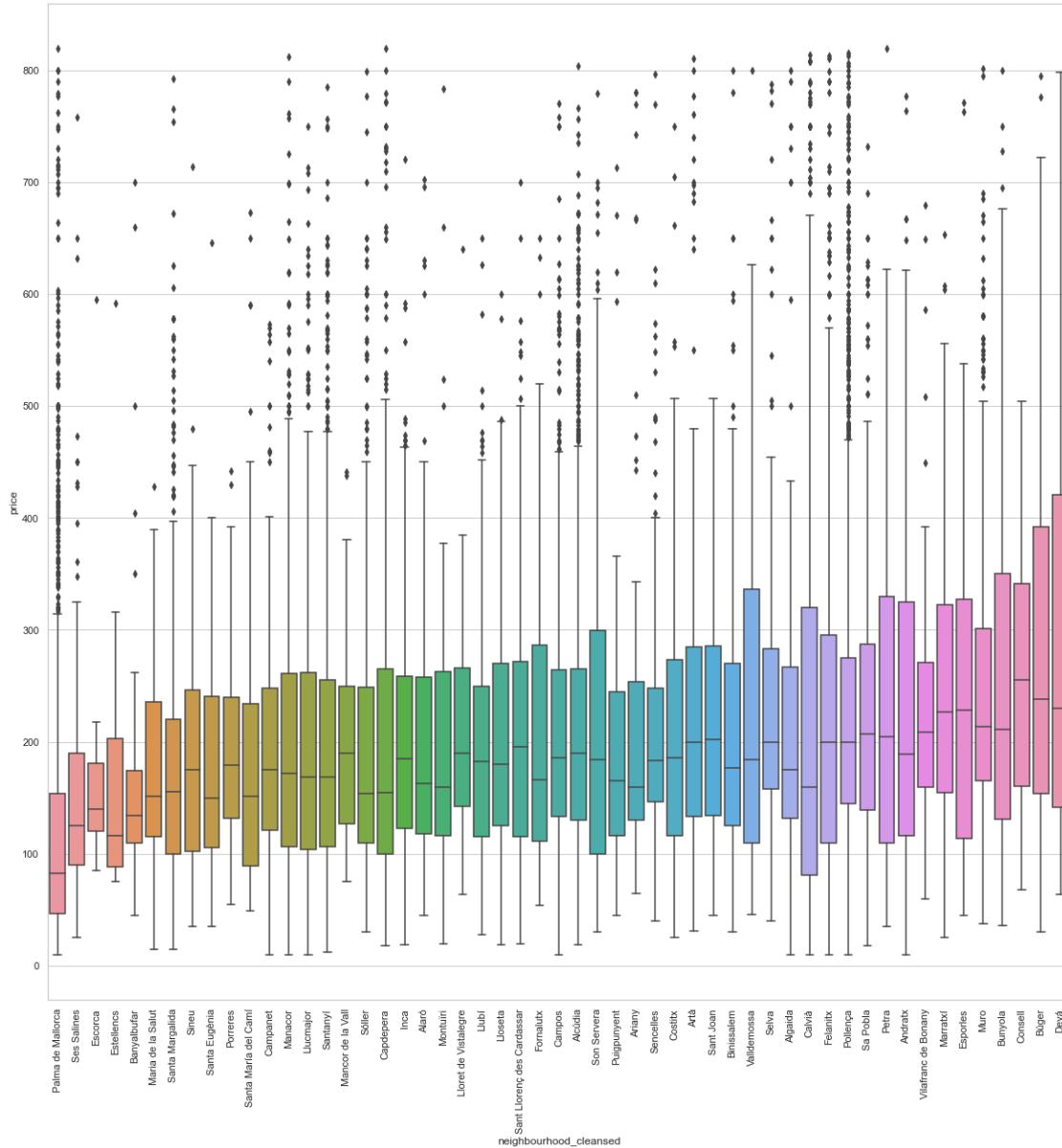
```
[37]: array(['Calvià', 'Santa Margalida', 'Maria de la Salut',  
        'Sant Llorenç des Cardassar', 'Palma de Mallorca', 'Selva',  
        'Esporles', 'Banyalbufar', 'Manacor', 'Sa Pobla', 'Alcúdia',  
        'Pollença', 'Artà', 'Santanyí', 'Fornalutx', 'Campos', 'Llucmajor',  
        'Sineu', '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)
```

```
[38]: 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)
```

```
[38]: [Text(0, 0, 'Palma de Mallorca'),  
      Text(1, 0, 'Ses Salines'),  
      Text(2, 0, 'Escorca'),  
      Text(3, 0, 'Estellencs'),  
      Text(4, 0, 'Banyalbufar'),  
      Text(5, 0, 'Maria de la Salut'),  
      Text(6, 0, 'Santa Margalida'),  
      Text(7, 0, 'Sineu'),  
      Text(8, 0, 'Santa Eugènia'),  
      Text(9, 0, 'Porreres'),  
      Text(10, 0, 'Santa María del Camí'),  
      Text(11, 0, 'Campanet'),  
      Text(12, 0, 'Manacor'),  
      Text(13, 0, 'Llucmajor'),  
      Text(14, 0, 'Santanyí'),  
      Text(15, 0, 'Mancor de la Vall'),  
      Text(16, 0, 'Sóller'),  
      Text(17, 0, 'Capdepera'),  
      Text(18, 0, 'Inca'),  
      Text(19, 0, 'Alaró'),
```



Text(20, 0, 'Montuïri'),  
Text(21, 0, 'Lloret de Vistalegre'),  
Text(22, 0, 'Llubí'),  
Text(23, 0, 'Lloseta'),  
Text(24, 0, 'Sant Llorenç des Cardassar'),  
Text(25, 0, 'Fornalutx'),  
Text(26, 0, 'Campos'),  
Text(27, 0, 'Alcúdia'),  
Text(28, 0, 'Son Servera'),  
Text(29, 0, 'Puigpunyent'),  
Text(30, 0, 'Ariany'),  
Text(31, 0, 'Sencelles'),  
Text(32, 0, 'Costitx'),  
Text(33, 0, 'Artà'),  
Text(34, 0, 'Sant Joan'),  
Text(35, 0, 'Binissalem'),  
Text(36, 0, 'Valldemossa'),  
Text(37, 0, 'Selva'),  
Text(38, 0, 'Algaida'),  
Text(39, 0, 'Calvià'),  
Text(40, 0, 'Felanitx'),  
Text(41, 0, 'Pollença'),  
Text(42, 0, 'Sa Pobla'),  
Text(43, 0, 'Petra'),  
Text(44, 0, 'Andratx'),  
Text(45, 0, 'Vilafranc de Bonany'),  
Text(46, 0, 'Marratxí'),  
Text(47, 0, 'Esporles'),  
Text(48, 0, 'Muro'),  
Text(49, 0, 'Bunyola'),  
Text(50, 0, 'Consell'),  
Text(51, 0, 'Búger'),  
Text(52, 0, 'Deyá']



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*.

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

```
[39]:
```

	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]

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

```
[40]:      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]

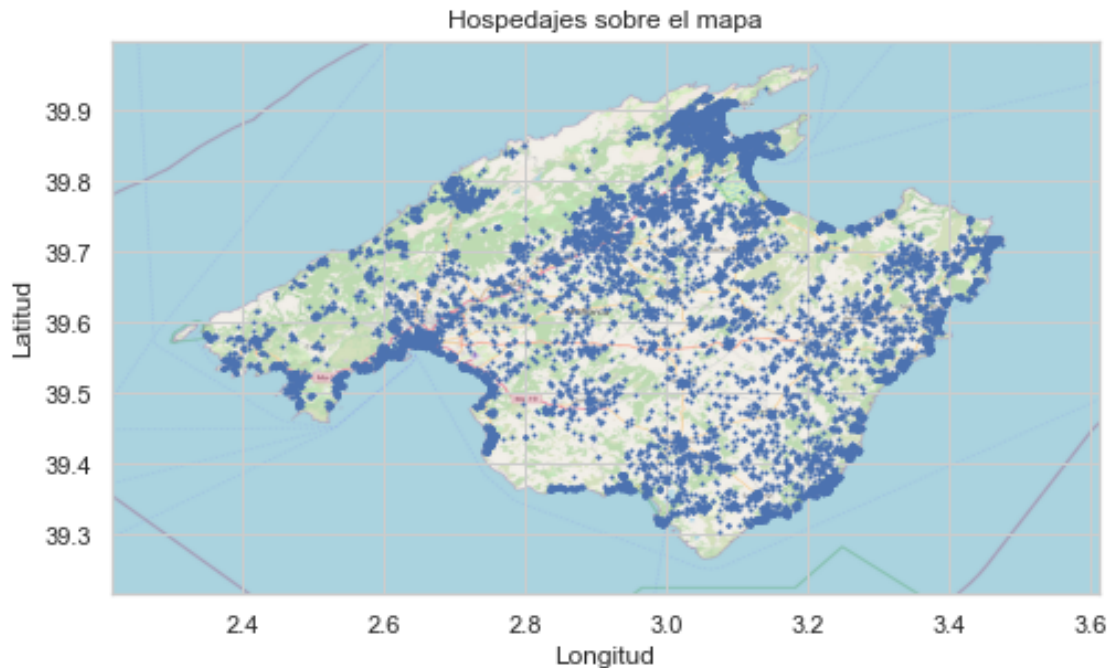
## 2.5 Columnas de latitud y longitud

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:

```
[41]: 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')
```

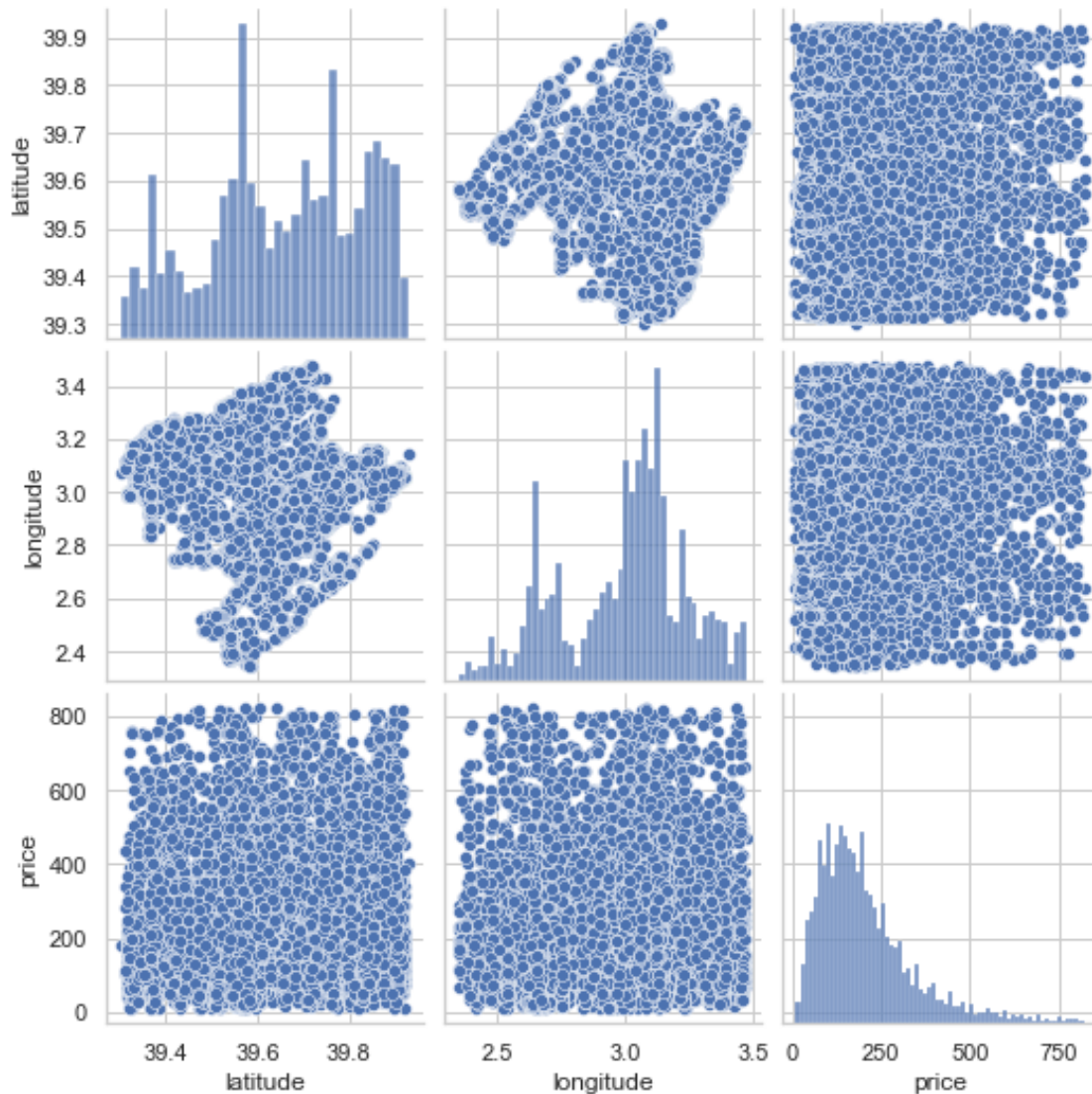
[41]: <matplotlib.image.AxesImage at 0x7fb780adc450>



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*;

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

[42]: <seaborn.axisgrid.PairGrid at 0x7fb780a2dc50>



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.

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

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

```
[44]: 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](#).

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

```
[45]:
```

	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.

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

```
[46]:
```

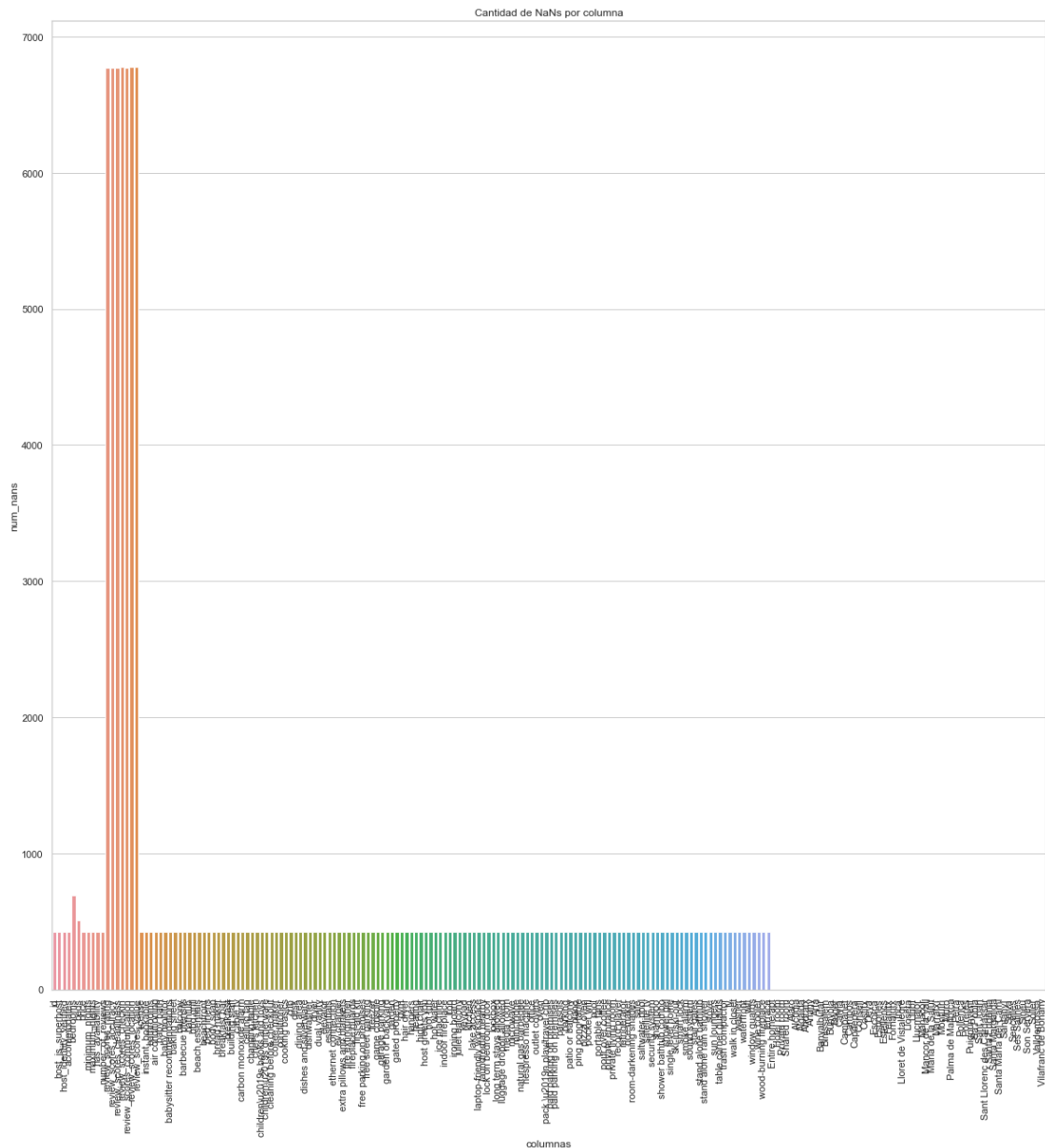
	columnas	num_nans
0	id	422
1	host_is_superhost	424
2	host_identity_verified	424
3	accommodates	422
4	bedrooms	692
..	...	...
201	Sineu	0
202	Son Servera	0
203	Sóller	0
204	Valldemossa	0
205	Vilafranc de Bonany	0



[206 rows x 2 columns]

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

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

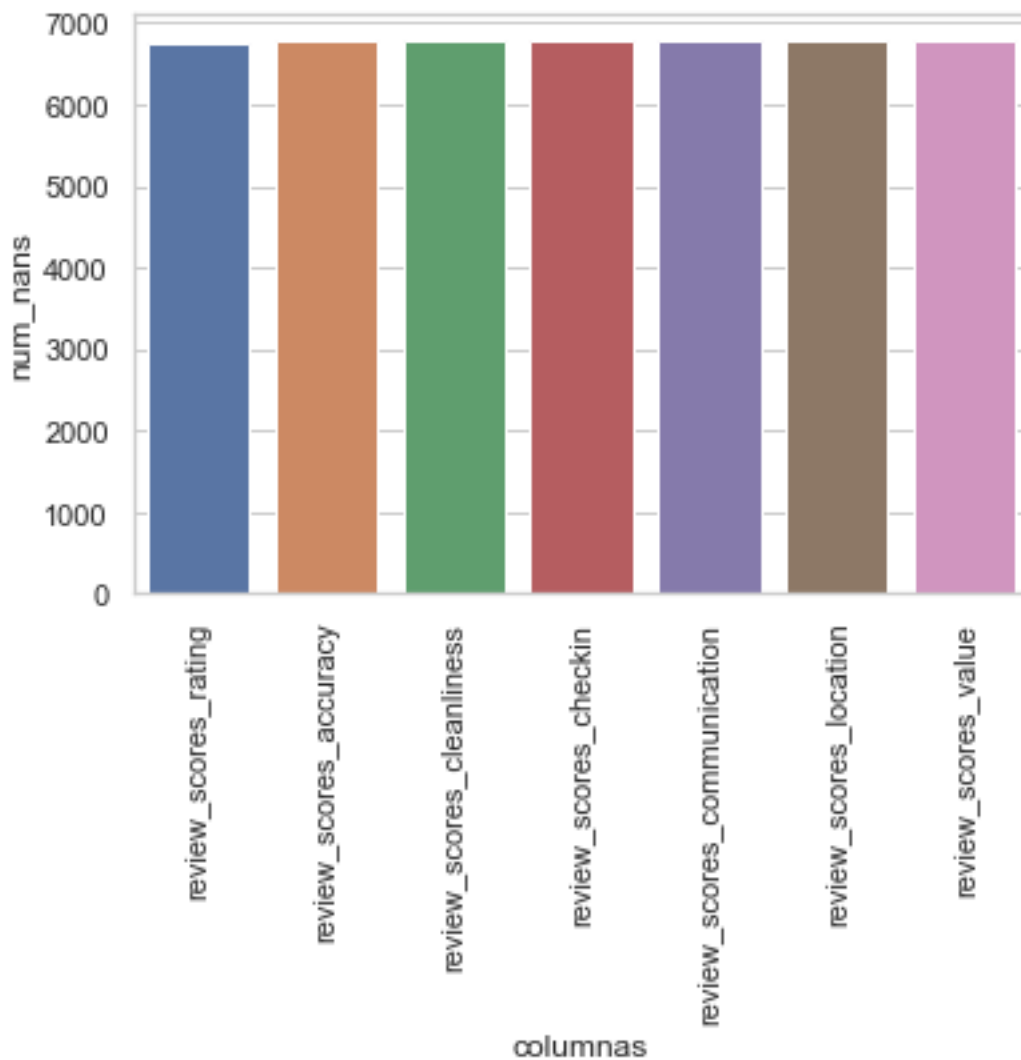


Teniendo en cuenta la longitud del dataset, se puede identificar una serie de columnas que tienen

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

```
[48]: 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]: [Text(0, 0, 'review_scores_rating'),
      Text(1, 0, 'review_scores_accuracy'),
      Text(2, 0, 'review_scores_cleanliness'),
      Text(3, 0, 'review_scores_checkin'),
      Text(4, 0, 'review_scores_communication'),
      Text(5, 0, 'review_scores_location'),
      Text(6, 0, 'review_scores_value')]
```



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

```
[49]: 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.

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

```
[50]:
```

	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	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 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.

```
[51]: df.dropna()
```

```

[51]:
      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
...
17175  44663146.0         0.0                    1.0          2.0
17176  44664664.0         0.0                    1.0          1.0
17177  44672575.0         0.0                    1.0          4.0
17179  44673578.0         0.0                    1.0          8.0
17180  44673714.0         0.0                    1.0          6.0

      bedrooms beds price minimum_nights maximum_nights \
0           1.0  1.0  89.0                5.0            60.0
1           4.0  7.0 175.0                7.0           365.0
2           3.0  4.0 140.0                6.0           365.0
3           2.0  4.0 200.0                5.0           365.0
4           1.0  2.0 110.0                2.0           365.0
...
17175           1.0  0.0  88.0                1.0          1125.0
17176           2.0  4.0  40.0                1.0           365.0
17177           2.0  2.0 120.0                5.0            14.0
17179           4.0  4.0 548.0                1.0           365.0
17180           3.0  3.0 180.0                5.0            14.0

      has_availability ... Santa María del Camí Santanyí Selva Sencelles \
0                1.0 ...                0            0            0            0
1                1.0 ...                0            0            0            0
2                1.0 ...                0            0            0            0
3                1.0 ...                0            0            0            0
4                1.0 ...                0            0            0            0
...
17175           ... 1.0 ...                0            0            0            0
17176           ... 1.0 ...                0            0            0            0
17177           ... 1.0 ...                0            0            0            0
17179           ... 1.0 ...                0            0            0            1
17180           ... 1.0 ...                0            0            0            0

      Ses Salines Sineu Son Servera Sóller Valldemossa \
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
...
17175           ... 0      0            0      0            0

```

17176	0	0	0	0	0
17177	0	0	0	0	0
17179	0	0	0	0	0
17180	0	0	0	0	0

	Vilafranc de Bonany
0	0
1	0
2	0
3	0
4	0
...	...
17175	0
17176	0
17177	0
17179	0
17180	0

[16419 rows x 199 columns]

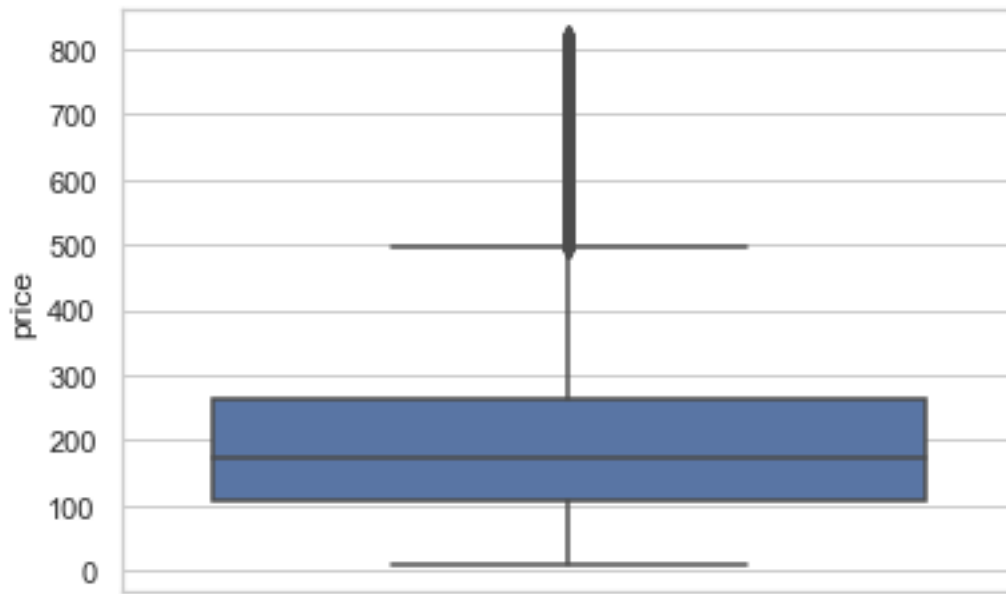
Para poder analizar la influencia del preprocesado en el *dataset*, se vuelven a realizar las comprobaciones iniciales de distribución y de outliers.

En primer lugar analizamos la estadística del precio:

```
[52]: df.price.describe()
```

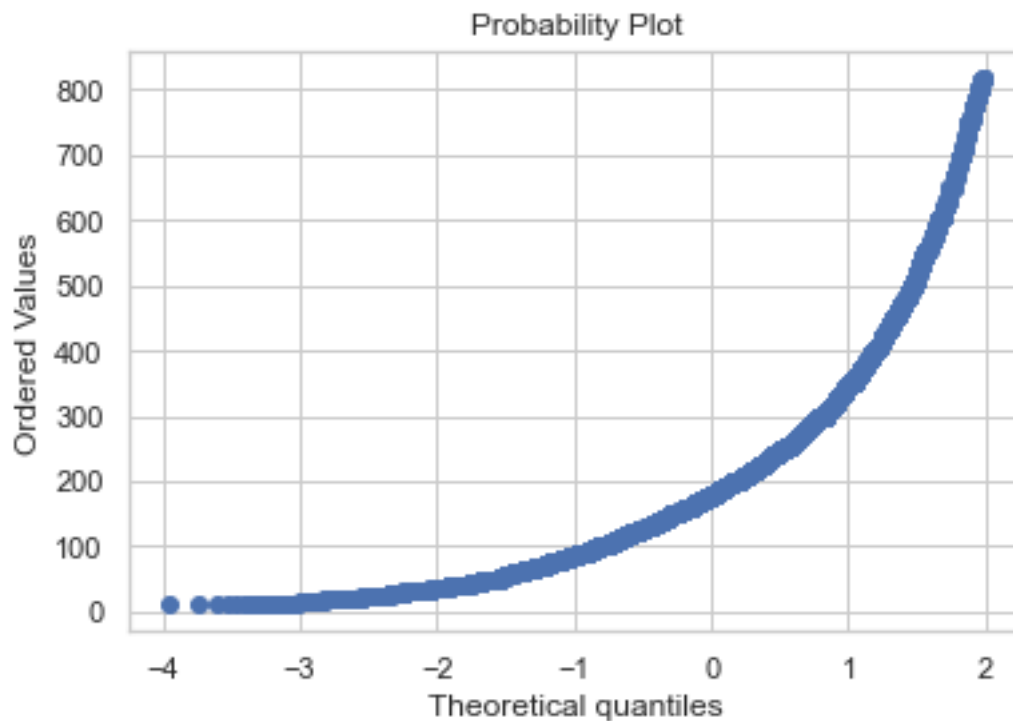
```
[52]: count    17181.000000
      mean      207.043386
      std       139.862822
      min       10.000000
      25%       109.000000
      50%       175.000000
      75%       264.000000
      max       819.000000
      Name: price, dtype: float64
```

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



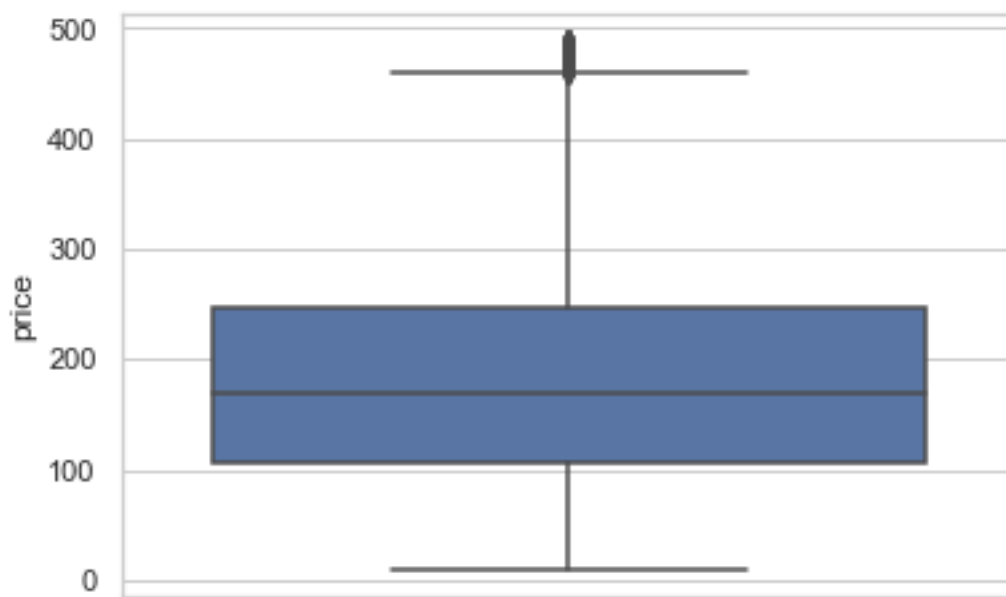
```
[54]: stats.probplot(df.price, plot=sns.mpl.pyplot)
```

```
[54]: ((array([-3.94816734, -3.73042554, -3.6112104 , ...,  3.6112104 ,
              3.73042554,  3.94816734])),
      array([10., 10., 10., ..., nan, nan, nan])),
      (nan, nan, nan))
```



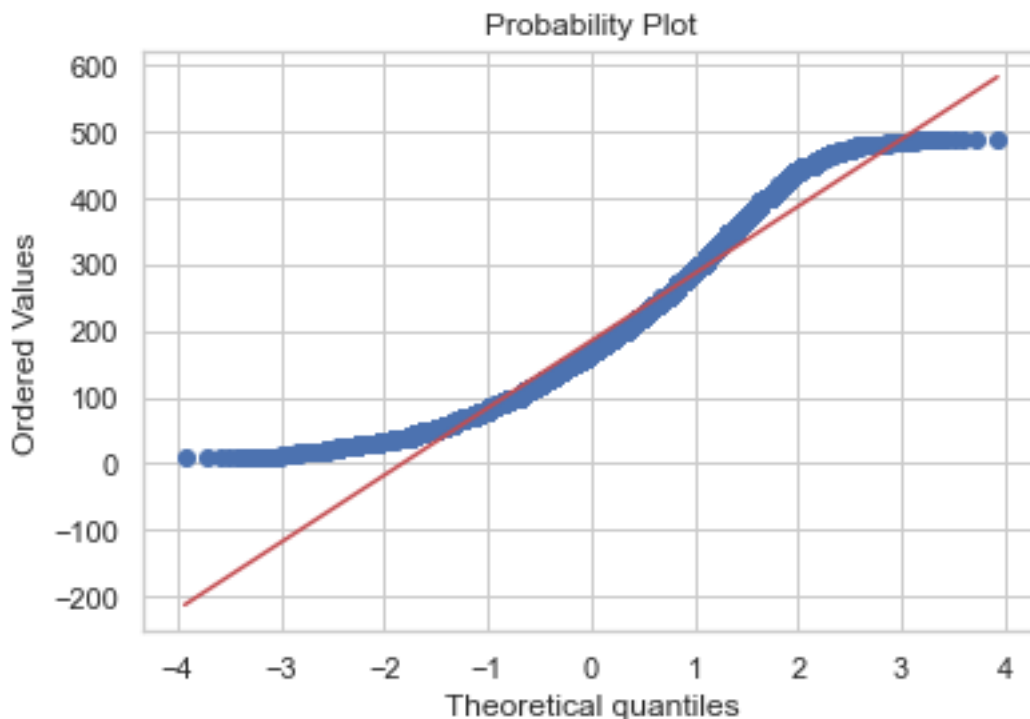
La distribución de precios continua en la misma línea, se pueden apreciar cambios poco significativos, como por ejemplo un descenso de la media (de 244 a 207). De ello se deduce que a pesar del tratamiento, el dataset conserva la estructura inicial, no así la estructura con respecto a la distribución por cuantiles, 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([-3.92968458, -3.71097508, -3.59118796, ...,  3.59118796,
              3.71097508,  3.92968458])),
      array([ 10.,  10.,  10., ..., 489., 489., 489.])),
      (101.26007504719549, 185.04747208246408, 0.9757998504636809))
```



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, propio de la librería *sklearn* y lo aplicaremos a ambos.

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

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

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-64-de2bb674ef84> in <module>
      1 df_train, df_test = train_test_split(dfB, test_size= 0.33)
----> 2 df_train.asDataFrame()

/Applications/anaconda3/envs/helena2/lib/python3.7/site-packages/pandas/core/
↳ generic.py in __getattr__(self, name)
    5458         if self._info_axis.
↳ _can_hold_identifiers_and_holds_name(name):
    5459             return self[name]
```



```

-> 5460             return object.__getattr__(self, name)
    5461
    5462     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'asDataFrame'

```

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

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

```

[58]: count      10919.000000
      mean        184.815242
      std         103.773949
      min          10.000000
      25%         104.000000
      50%         167.000000
      75%         247.000000
      max         489.000000
      Name: price, dtype: float64

```

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

```

[59]: count      5379.000000
      mean        185.518883
      std         103.711458
      min          10.000000
      25%         105.500000
      50%         169.000000
      75%         248.000000
      max         489.000000
      Name: price, dtype: float64

```

Una vez realizada esta comprobación, vamos a escalar los datos:

```

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

df_train = funcion_escalado.transform(df_train)
df_test = funcion_escalado.transform(df_test)

```

```

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

```

```

[66]: count      10919.000000
      mean        184.517802
      std         103.375700

```

```
min          10.000000
25%          105.000000
50%          167.000000
75%          246.000000
max          489.000000
Name: price, dtype: float64
```

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

```
[68]: count      5379.000000
mean       186.122666
std        104.509099
min         10.000000
25%        104.000000
50%        169.000000
75%        250.000000
max         489.000000
Name: price, dtype: float64
```

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 modelo seleccionado es un modelo de regresión. Se empezará por