UdeCataluña

Cree una copia de este Notebook en su Drive y desarrolle el taller sobre ese documento copia. Para crear la copia debe dar click en el botón **Copiar en Drive** ubicado en el panel superior, se abrirá un nuevo notebook que podrá manipular a su antojo y que se almacenará en su Drive dentro de la Carpeta *Colab Notebooks*.

Predicción del precio de hospedajes en Airbnb para la ciudad de Nueva York

Contexto: Airbnb es una empresa que ofrece una plataforma de software dedicada a la oferta de alojamientos particulares y turísticos mediante la cual los anfitriones pueden publicitar y contratar el arriendo de sus propiedades con sus huéspedes; anfitriones y huéspedes pueden valorarse mutuamente, como referencia para futuros usuarios. Muchos nuevos anfitriones no cuentan con información global de tendencias del mercado por lo que sus precios no son óptimos. Airbnb gana una comisión por cada arrendamiento, por lo tanto, está interesado en que sus anfitriones cobren una tarifa óptima de acuerdo a las características del hospedaje. Si los anfitriones ganan más... Airbnb también.

Problema de Negocio: La empresa Airbnb lo ha contratado para desarrollar un análisis descriptivo y exploratorio que permita responder la siguiente pregunta: ¿Cuál es la variable o característica más relevante para determinar el precio de un hospedaje en Airbnb?

Sistema de información: El conjunto de datos objetivo posee información acerca de 30.000 hospedajes de la plataforma Airbnb en la ciudad de Nueva York. Los datos a usar son datos públicos creados por Inside Airbnb, para más información puede consultar aquí.

Indicaciones para resolver el Taller

El objetivo de este taller es que usted pueda desarrollar algunas tareas de limpieza para asegurar la calidad del análisis. Para ello tendrá que programar, investigar y analizar todos los resultados que vaya obteniendo. Tenga en cuenta las siguientes indicaciones:

- Añada comentarios al código para que documente sus soluciones.
- Coloque su análisis en una celda de Texto luego de cada resultado.
- Para resolver un ejercicio puede usar tantas celdas de Código o Texto como requiera.

Si se le presenta un error de código o duda. Siga los siguientes pasos:

1. Lea y entienda el error, para ello puede buscar en la documentación de la librería o googlearlo
2. Intente resolverlo
3. Comuníquese con el expero temático usando el Foro, recuerde enviar un pantallazo del error y mencionar que Ejercicio está solucionando. **Abstengase de compartir el link de su Notebook en el Foro.**

A continuación, se listan algunos recursos que pueden ser valiosos para su análisis.

- En esta página puede encontrar las gráficas que se pueden construir dependiendo de las variables disponibles, una breve explicación de cada gráfica y código para construir cada visualización. https://www.data-to-viz.com/
- Documentación de Pandas. https://pandas.pydata.org/docs/index.html
- Documentación de Seaborn. https://seaborn.pydata.org/index.html

0)-Librerías necesarias para el proyecto

A continuación vamos a cargar las librerías necesarias para el desarrollo de este caso.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
```

In [2]:
```
!python --version
print('NumPy', np.__version__)
print('Pandas', pd.__version__)
print('Matplotlib', mpl.__version__)
print('Seaborn',sns.__version__)
```

```
Python 3.10.11
NumPy 1.22.4
Pandas 1.5.3
Matplotlib 3.7.1
Seaborn 0.12.2
```

Este caso fue creado con las siguientes versiones:

```
Python 3.7.13
NumPy 1.21.6
Pandas 1.3.5
Matplotlib 3.2.2
Seaborn 0.11.2
```

1)-Data set de AirBnb

In [3]:
```
pd.options.display.max_columns = 100 # Permite visualizar todas las columnas del dataf

airbnb = pd.read_csv('https://github.com/HarryVargas96/UdeCataluna/blob/main/data/airb
airbnb.head(3)
```

Out[3]:

	id	name	transit	host_id	host_since	host_response_time	host_response_rate	host_is_s
0	2539	Clean & quiet apt home by the park	Very close to F and G trains and Express bus i...	2787	39698.0	within an hour	1.0	
1	3647	THE VILLAGE OF HARLEM....NEW YORK !	NaN	4632	39777.0	within a day	1.0	
2	7750	Huge 2 BR Upper East Cental Park	NaN	17985	39953.0	within a day	1.0	

In [4]:
```
# Dimensiones del dataframe
airbnb.shape
```

Out[4]:
```
(30179, 76)
```

In [5]:
```
# Resumen de las variables del dataframe
airbnb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30179 entries, 0 to 30178
Data columns (total 76 columns):
 #   Column                                        Non-Null Count  Dtype  
---  ------                                        --------------  -----  
 0   id                                            30179 non-null  int64  
 1   name                                          30166 non-null  object 
 2   transit                                       18190 non-null  object 
 3   host_id                                       30179 non-null  int64  
 4   host_since                                    30170 non-null  float64
 5   host_response_time                            17082 non-null  object 
 6   host_response_rate                            17082 non-null  float64
 7   host_is_superhost                             30170 non-null  float64
 8   host_listings_count                           30170 non-null  float64
 9   host_identity_verified                        30170 non-null  float64
 10  street                                        30179 non-null  object 
 11  neighbourhood                                 30170 non-null  object 
 12  latitude                                      30179 non-null  float64
 13  longitude                                     30179 non-null  float64
 14  property_type                                 30179 non-null  object 
 15  room_type                                     30179 non-null  object 
 16  accommodates                                  30179 non-null  int64  
 17  bathrooms                                     30179 non-null  float64
 18  bedrooms                                      30179 non-null  int64  
 19  beds                                          30179 non-null  int64  
 20  bed_type                                      30179 non-null  object 
 21  amenities                                     30179 non-null  object 
 22  price                                         30179 non-null  int64  
 23  guests_included                               30179 non-null  int64  
 24  extra_people                                  30179 non-null  int64  
 25  minimum_nights                                30179 non-null  int64  
 26  calendar_updated                              30179 non-null  object 
 27  has_availability                              30179 non-null  int64  
 28  availability_30                               30179 non-null  int64  
 29  availability_60                               30179 non-null  int64  
 30  availability_90                               30179 non-null  int64  
 31  availability_365                              30179 non-null  int64  
 32  number_of_reviews                             30179 non-null  int64  
 33  number_of_reviews_ltm                         30179 non-null  int64  
 34  review_scores_rating                          21094 non-null  float64
 35  review_scores_accuracy                        21068 non-null  float64
 36  review_scores_cleanliness                     21078 non-null  float64
 37  review_scores_checkin                         21050 non-null  float64
 38  review_scores_communication                   21069 non-null  float64
 39  review_scores_location                        21047 non-null  float64
 40  review_scores_value                           21049 non-null  float64
 41  instant_bookable                              30179 non-null  int64  
 42  cancellation_policy                           30179 non-null  object 
 43  calculated_host_listings_count                30179 non-null  int64  
 44  calculated_host_listings_count_entire_homes   30179 non-null  int64  
 45  calculated_host_listings_count_private_rooms  30179 non-null  int64  
 46  calculated_host_listings_count_shared_rooms   30179 non-null  int64  
 47  reviews_per_month                             21919 non-null  float64
 48  check_in_24h                                  30179 non-null  int64  
 49  air_conditioning                              30179 non-null  int64  
 50  high_end_electronics                          30179 non-null  int64  
 51  bbq                                           30179 non-null  int64  
 52  balcony                                       30179 non-null  int64  
 53  nature_and_views                              30179 non-null  int64  
 54  bed_linen                                     30179 non-null  int64  
```

```
 55  breakfast                                  30179 non-null  int64
 56  tv                                         30179 non-null  int64
 57  coffee_machine                             30179 non-null  int64
 58  cooking_basics                             30179 non-null  int64
 59  white_goods                                30179 non-null  int64
 60  elevator                                   30179 non-null  int64
 61  gym                                        30179 non-null  int64
 62  child_friendly                             30179 non-null  int64
 63  parking                                    30179 non-null  int64
 64  outdoor_space                              30179 non-null  int64
 65  host_greeting                              30179 non-null  int64
 66  hot_tub_sauna_or_pool                      30179 non-null  int64
 67  internet                                   30179 non-null  int64
 68  long_term_stays                            30179 non-null  int64
 69  pets_allowed                               30179 non-null  int64
 70  private_entrance                           30179 non-null  int64
 71  secure                                     30179 non-null  int64
 72  self_check_in                              30179 non-null  int64
 73  smoking_allowed                            30179 non-null  int64
 74  accessible                                 30179 non-null  int64
 75  event_suitable                             30179 non-null  int64
dtypes: float64(16), int64(49), object(11)
memory usage: 17.5+ MB
```

2)-Errores y datos atípicos

En la fase de exploración previa identificamos algunas incongruencias en las variables price y bathrooms. Para la variable price encontramos hospedajes con precios desde 0 dólares, y a su vez tenemos algunos atípicos muy lejos del cuartil 75 y el bígote superior. Vea el boxplot para el precio.

Por otro lado, la variable bathrooms presenta valores decimales, 0.5 y 7.5 que podemos ver en el resumen estadístico. Si bien es extraño pensar en cantidad de baños como un valor decimal es muy común encontrar la notación 0.5 en estos escenarios. 0.5 representa un baño que solo tiene cisterna y lavamanos, pero que no tiene ducha. 1.0 representa un baño con ducha, cisterna y lavamanos.

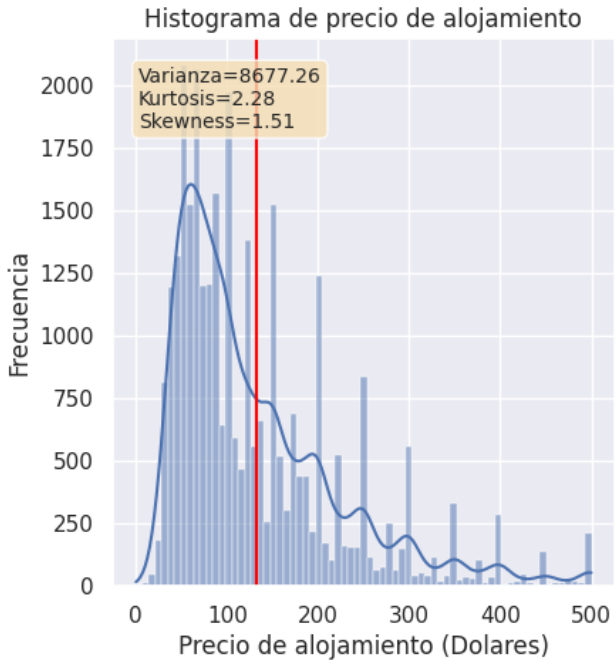
In [12]:
```
airbnb[['price','bathrooms']].describe()
```

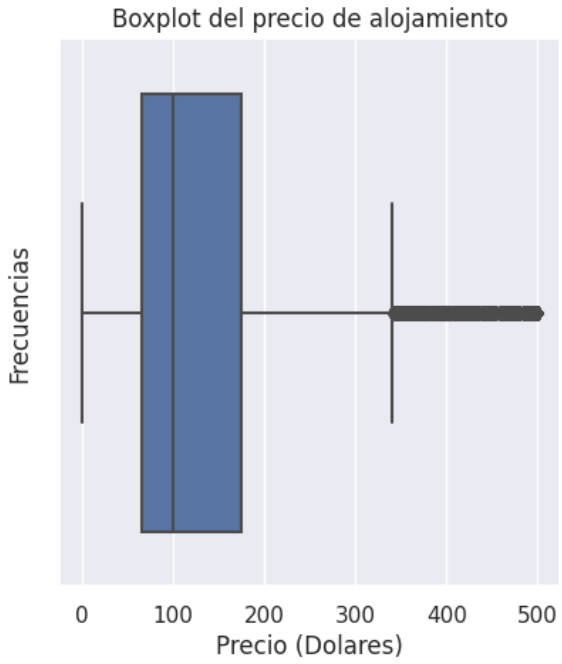
Out[12]:

	price	bathrooms
count	30179.000000	30179.000000
mean	132.949965	1.151595
std	93.151824	0.422225
min	0.000000	0.500000
25%	65.000000	1.000000
50%	100.000000	1.000000
75%	175.000000	1.000000
max	500.000000	7.500000

In [13]:

```
fig, ax = plt.subplots(1,2,figsize=(10,5))
# Plot para el histograma de los precios de alojamiento
sns.histplot(data=airbnb,x="price",kde=True, ax = ax[0])
sns.boxplot(data=airbnb,x="price")

# Plot vertical para indicar el promedio del precio
mean = airbnb['price'].mean()
ax[0].axvline(mean, color='red')

# Agregamos un pequeño box para mostrar el kurtosis, la varianza y el skewness
variance = round(airbnb['price'].var(),2)
kurtosis = round(airbnb['price'].kurt(),2)
skewness = round(airbnb['price'].skew(),2)

textstr = '\n'.join( (r'Varianza=%.2f' % (variance, ), r'Kurtosis=%.2f' % (kurtosis, )
props = dict(boxstyle='round', facecolor='wheat', alpha=0.8)
ax[0].text(0.05, 0.95, textstr, transform=ax[0].transAxes, fontsize=10,
        verticalalignment='top', bbox=props)

ax[0].set_title("Histograma de precio de alojamiento")
ax[0].set_xlabel("Precio de alojamiento (Dolares)")
ax[0].set_ylabel("Frecuencia")
ax[1].set_title("Boxplot del precio de alojamiento")
ax[1].set_xlabel("Precio (Dolares)")
ax[1].set_ylabel("Frecuencias")

plt.subplots_adjust(hspace=0.5)
plt.show()
```





2.1-Detección de valores atípicos

A continuación se genera un conjunto de datos llamado `atipicos_precio` que contiene las observaciones consideradas como valores atípicos severos de acuerdo al boxplot del precio. Para ello generamos la función `outliers` la cual detecta y filtra a partir del dataframe a todos aquellos valores que superan 1.5 veces el rango intercuartílico.

In [14]:
```
# Outliers de una única columna
def outliers(dataframe, colum_name):
  q1 = dataframe[colum_name].quantile(0.25)
  q3 = dataframe[colum_name].quantile(0.75)
  IQR =q3-q1
  condition = (dataframe[colum_name] < (q1-1.5*IQR)) | (dataframe[colum_name] > (q3+1.
  outliers_data = dataframe.loc[condition]
  return outliers_data

# Eliminar outliers de la data
def filtered_outliers(dataframe, column_name):
  outliers_index = list(outliers(dataframe, column_name).index)
  df_filtered = dataframe.drop(outliers_index)
  return df_filtered
```

In [15]:
```
price_outliers = outliers(airbnb,"price")
price_without_outliers = filtered_outliers(airbnb,"price")

print("the shape of the outlierdata is=", price_outliers.shape)
```

```
the shape of the outlierdata is= (1418, 76)
```

En las líneas de código anteriores, hemos hecho dos nuevos dataframes donde 'price_outliers' contiene el dataframe de los precios considerados como atípicos mientras que 'price_without_outliers' contiene el dataframe original de AirBnb pero una vez los valores atípicos han sido removidos.

2.2-Análisis de valores atípicos

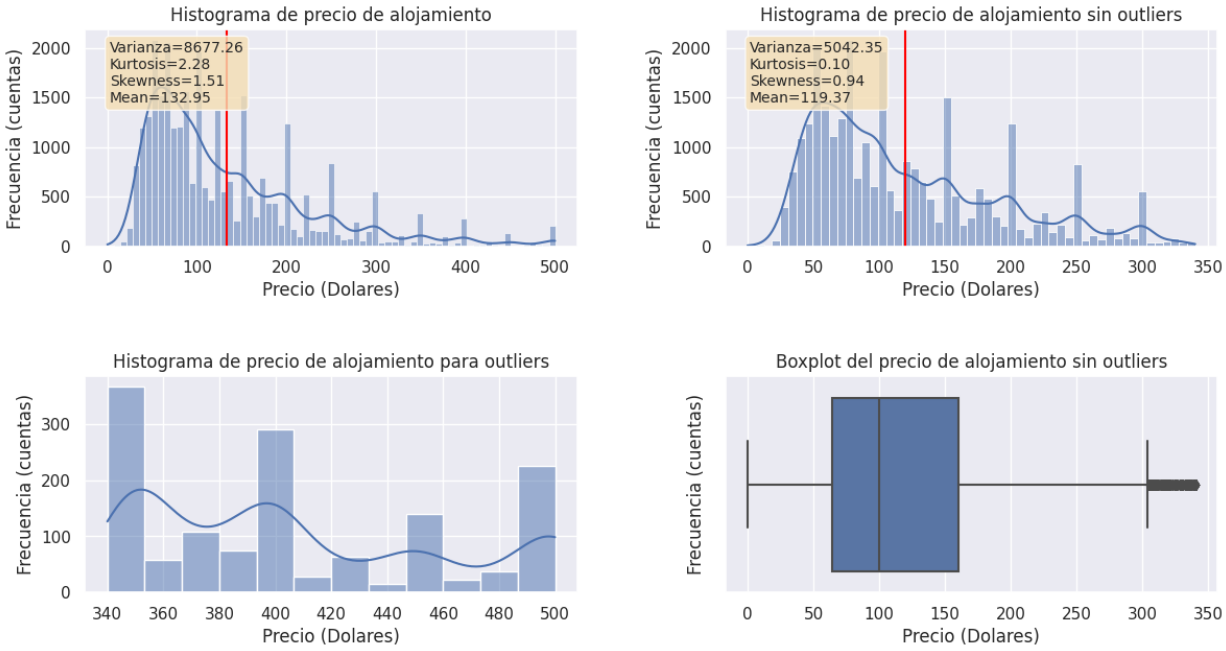
En esta subsección analizaremos los valores atípicos del precio respondiendo a preguntas tales como la cantidad de valores atípicos presentes en la data, que diferencias posee la data original con respecto a la data filtrada (sin outliers) y darémos una breve descripción del patrón que el precio de los hospedajes atípicos arroja.

In [16]:
```
atípicos_precio = pd.read_csv('https://github.com/HarryVargas96/UdeCataluna/blob/main/
price_outliers = atípicos_precio
print('Las dimensiones de atipicos_precio son: {}'.format(atípicos_precio.shape))
```

```
Las dimensiones de atipicos_precio son: (1430, 76)
```

In [17]:
```
# Plot del histograma de los precios original, atípicos y para la distribución sin los
fig, ax = plt.subplots(2,2, figsize=(14,7))

sns.histplot(data=airbnb, x="price", kde=True, ax=ax[0,0])
sns.histplot(data=price_without_outliers, x="price", kde=True, ax=ax[0,1])
sns.histplot(data=price_outliers, x="price", kde=True, ax=ax[1,0])
sns.boxplot(data=price_without_outliers, x="price")

# Plot vertical para indicar el promedio del precio
mean_1 = airbnb["price"].mean()
mean_2 = price_without_outliers["price"].mean()
ax[0,0].axvline(mean_1, color="red")
ax[0,1].axvline(mean_2, color="red")

# Agregamos un pequeño box para mostrar el kurtosis, la varianza y el skewness
variance_1 = round(airbnb["price"].var(),2)
variance_2 = round(price_without_outliers["price"].var(),2)
kurtosis_1 = round(airbnb['price'].kurt(),2)
kurtosis_2 = round(price_without_outliers['price'].kurt(),2)
skewness_1 = round(airbnb['price'].skew(),2)
skewness_2 = round(price_without_outliers['price'].skew(),2)

textstr = '\n'.join( (r'Varianza=%.2f' % (variance_1, ), r'Kurtosis=%.2f' % (kurtosis_
props = dict(boxstyle='round', facecolor='wheat', alpha=0.8)
ax[0,0].text(0.05, 0.95, textstr, transform=ax[0,0].transAxes, fontsize=10,
        verticalalignment='top', bbox=props)

textstr = '\n'.join( (r'Varianza=%.2f' % (variance_2, ), r'Kurtosis=%.2f' % (kurtosis_
props = dict(boxstyle='round', facecolor='wheat', alpha=0.8)
ax[0,1].text(0.05, 0.95, textstr, transform=ax[0,1].transAxes, fontsize=10,
        verticalalignment='top', bbox=props)

# Agregamos leyendas para mayor claridad del plot
ax[0,0].set_title("Histograma de precio de alojamiento")
ax[0,1].set_title("Histograma de precio de alojamiento sin outliers")
ax[1,0].set_title("Histograma de precio de alojamiento para outliers")
ax[1,1].set_title("Boxplot del precio de alojamiento sin outliers")
ax[0,0].set_xlabel("Precio de alojamiento (Dolares)")
ax[0,1].set_xlabel("Precio de alojamiento (Dolares)")
ax[0,0].set_ylabel("Frecuencia (cuentas)")
ax[0,1].set_ylabel("Frecuencia (cuentas)")
ax[1,0].set_ylabel("Frecuencia (cuentas)")
ax[1,1].set_ylabel("Frecuencia (cuentas)")
ax[0,0].set_xlabel("Precio (Dolares)")
```

```
ax[0,1].set_xlabel("Precio (Dolares)")
ax[1,0].set_xlabel("Precio (Dolares)")
ax[1,1].set_xlabel("Precio (Dolares)")

# Ajustamos el espacio entre subplots
plt.subplots_adjust(hspace=0.6, wspace=0.3)

plt.show()
```



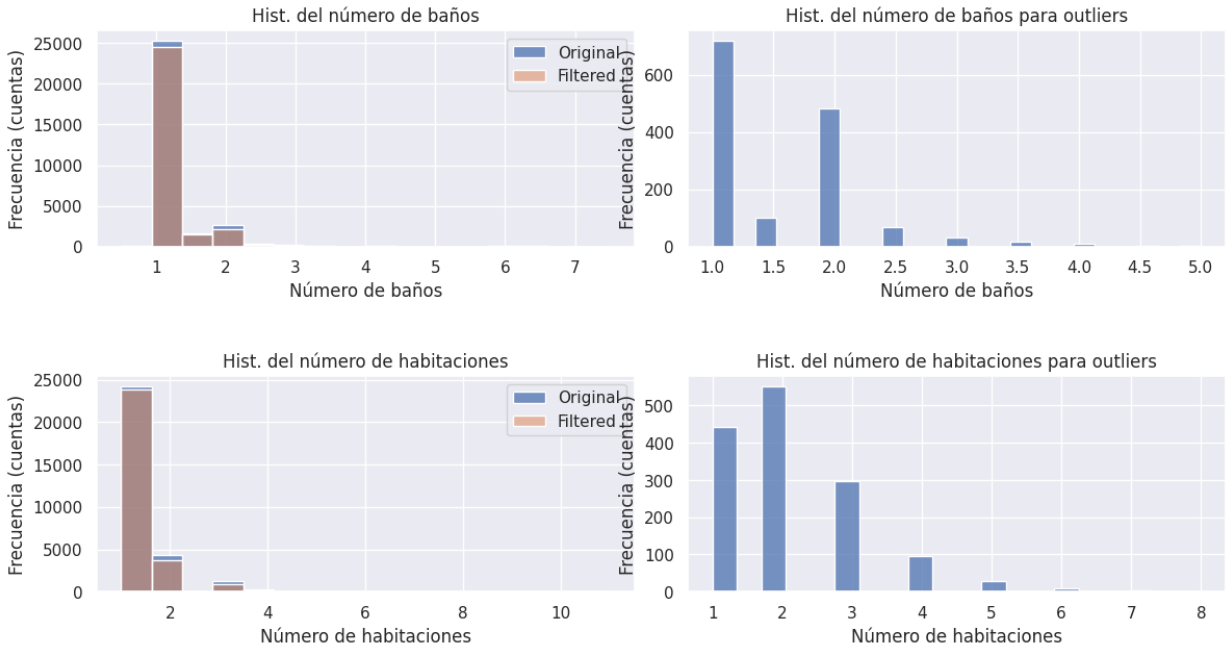
Se puede observar en el anterior conjunto de plots, comolos valores atípicos para el precio dentro de la distribución original ayudaban no solo a que el promedio de los precios fuese mayor sino a demás contribuyen notoriamente la varianza, la concentración y el sesgo de los valores. De hecho, una vez removidos éstos valores atípicos desde la data original, el sesgo de la distribución se reduce consireablemente haciendo que los valores esten más ceracanos al promedio. Más aún, la influencia de los valores atípicos es tal que el rango de la distribución se reduce casi a la mitad.

Por otra parte, la distribución de los precios atípicos no parece seguir un patrón fuertemente definido, podría decirse ingenuamente que es una distribución multimodal de cuatro picos pero realmente la data no es lo suficientemente precisa ni uniforme en este sector de la data original como para ofrecer tal conclución.

No obstante, es de resaltar que a pesar que un primer filtro de valores atípicos fue efectuado, al realizar el boxplot de la data filtrada si bien el sesgo hacia la izquierda se reduce considerablemente y la varianza de los precios disminuye notoriamente, aún exísten valores atípicos para esta nueva data que entre el rango de rpecios de 300 y 350 dólares.

In [18]:
```
# Plot del histograma de los precios original, atípicos y para la distribución sin los
fig, ax = plt.subplots(2,2, figsize=(14,7))


sns.histplot(data=airbnb, x="bathrooms", label="Original",ax=ax[0,0])
sns.histplot(data=price_without_outliers, x="bathrooms", label="Filtered",ax=ax[0,0],
```

```
sns.histplot(data=price_outliers, x="bathrooms", ax=ax[0,1])

sns.histplot(data=airbnb, x="bedrooms", label="Original",ax=ax[1,0])
sns.histplot(data=price_without_outliers, x="bedrooms", label="Filtered",ax=ax[1,0], a
sns.histplot(data=price_outliers, x="bedrooms", ax=ax[1,1])

# Agregamos leyendas para mayor claridad del plot
ax[0,0].set_title("Hist. del número de baños")
ax[0,1].set_title("Hist. del número de baños para outliers")
ax[1,0].set_title("Hist. del número de habitaciones")
ax[1,1].set_title("Hist. del número de habitaciones para outliers")
ax[0,0].set_xlabel("Número de baños")
ax[0,1].set_xlabel("Número de baños")
ax[1,0].set_xlabel("Número de habitaciones")
ax[1,1].set_xlabel("Número de habitaciones")
ax[0,0].set_ylabel("Frecuencia (cuentas)")
ax[0,1].set_ylabel("Frecuencia (cuentas)")
ax[1,0].set_ylabel("Frecuencia (cuentas)")
ax[1,1].set_ylabel("Frecuencia (cuentas)")

# Ajustamos el espacio entre subplots
plt.subplots_adjust(hspace=0.6, wspace=0.1)
ax[0, 0].legend()
ax[1, 0].legend()
plt.show()
```



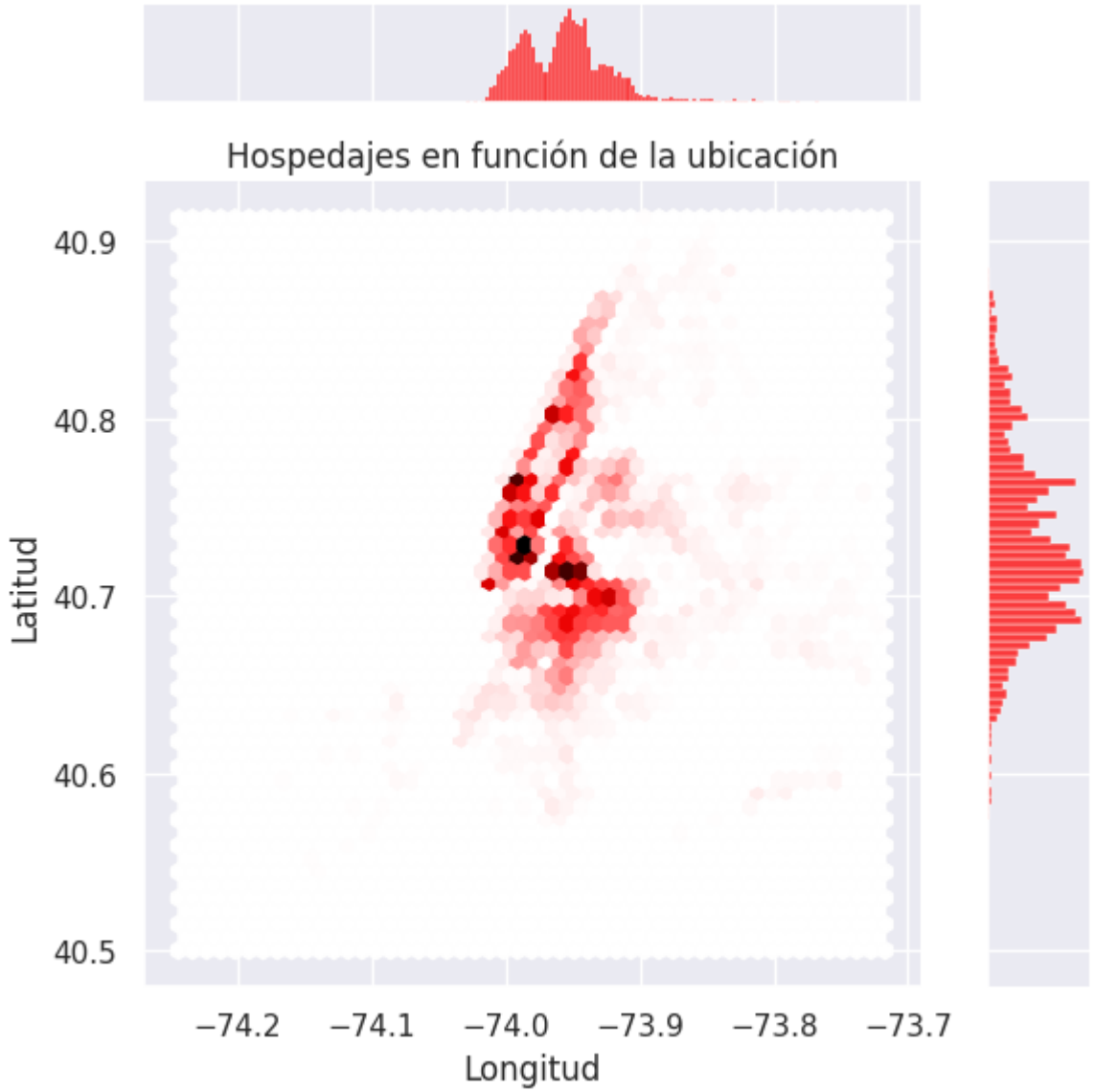
Al comparar las distribuciónes para las variables numero de baños y número de habitaciones para el conjunto completo de información con el subconjunto de outliers, podemos darnos cuenta que en ambas variables los outliers contribuyen poco a la distribución de precios más comunes de la distribución como se puede ver en los histogramas de la izquierda donde en cada caso, ambas distribuciones son casi idénticas.

No obstante, la parte derecha de los histogramas de la columna izquierda la cual no es tan visible para la data original, puede ser evidenciada en la distribución para los precios outliers. Los histogramas para los precios outliers en el caso de ambas variables presentan una mayor variedad, lo que nos puede indicar levemente que el precio atípico puede verse influenciado a

parte de múchos otros factores como vimos en la fase 1 del proyecto, por un alto número de baños y habitaciones. Sinembargo ésta conclusión es sólo una hipótesis pues la información que proveen las distribuciones no es lo suficientemente fuerte para asegurarla del todo, en especial en el caso del número de baños.

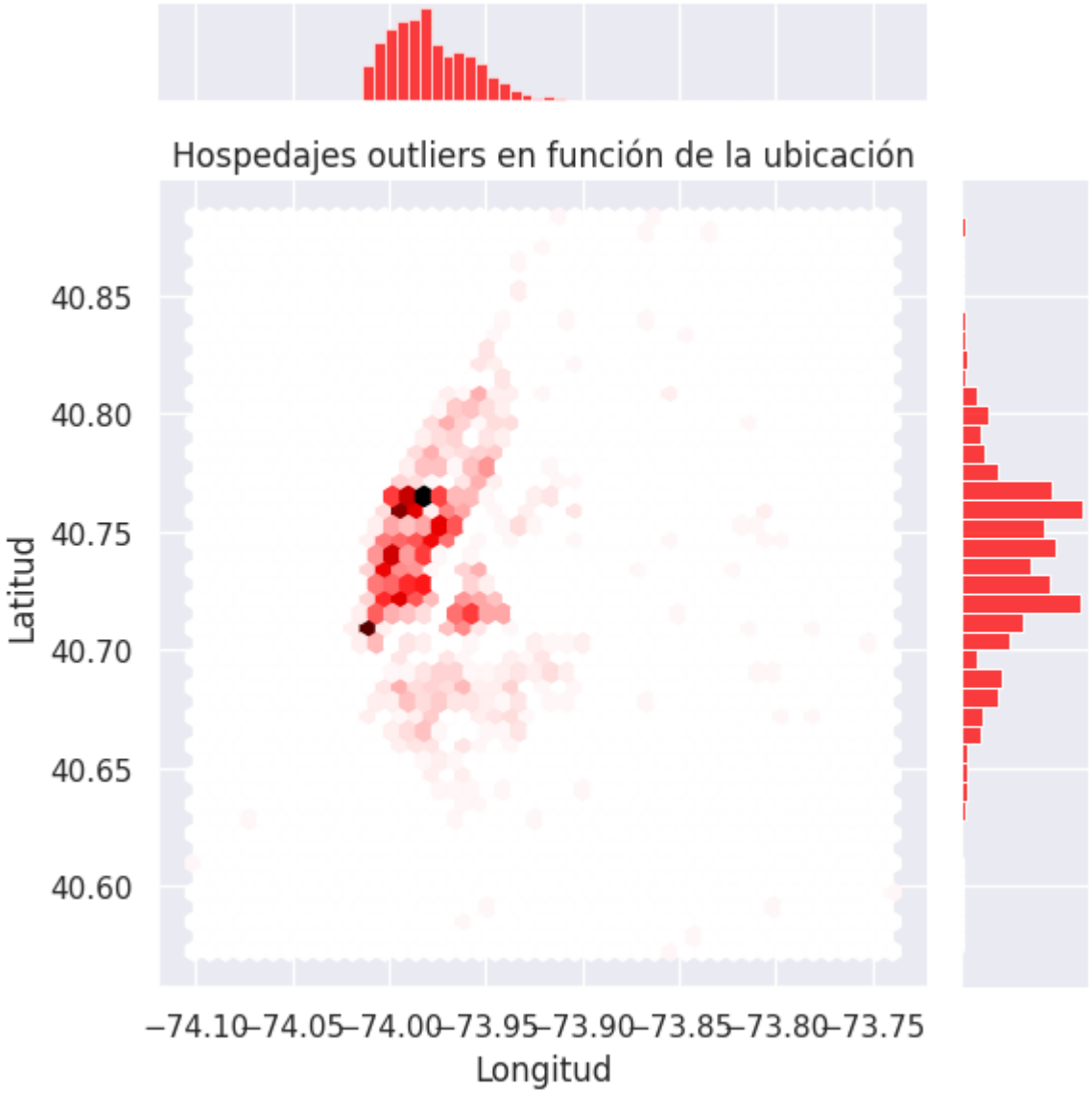
Procedamos ahora observar cual es el comportamiento del precion en términos de la ubicación del hospedaje para cada caso.

In [19]:
```
sns.jointplot(data = airbnb,x='longitude', y='latitude', kind="hex",color = 'red')
plt.xlabel('Longitud')
plt.ylabel('Latitud')
plt.title('Hospedajes en función de la ubicación')
plt.tight_layout()
plt.show()
```



In [20]:
```
sns.jointplot(data = price_outliers,x='longitude', y='latitude', kind="hex",color = 'r
plt.xlabel('Longitud')
plt.ylabel('Latitud')
plt.title('Hospedajes outliers en función de la ubicación')
```

```
plt.tight_layout()
plt.show()
```



Comparando los jointplots, podemos observar que los precios atípicos se ubican en una zona específica de la ciudad, más específicamente, los precios atípicos poseen un fuerte sesgo hacia la derecha en términos de longitud e incrementa su sesgo hacia la izquierda en términos de latitud. Ya que ésta data corresponde a información geoespacial, podémos decir que los alojamientos atípicos se concentran hacia la parte nor-occidente de Central Park contribuyendo al modo izuqierdo de la distribución de longitud.

2.3-Analisis geoespacial de valores atípicos

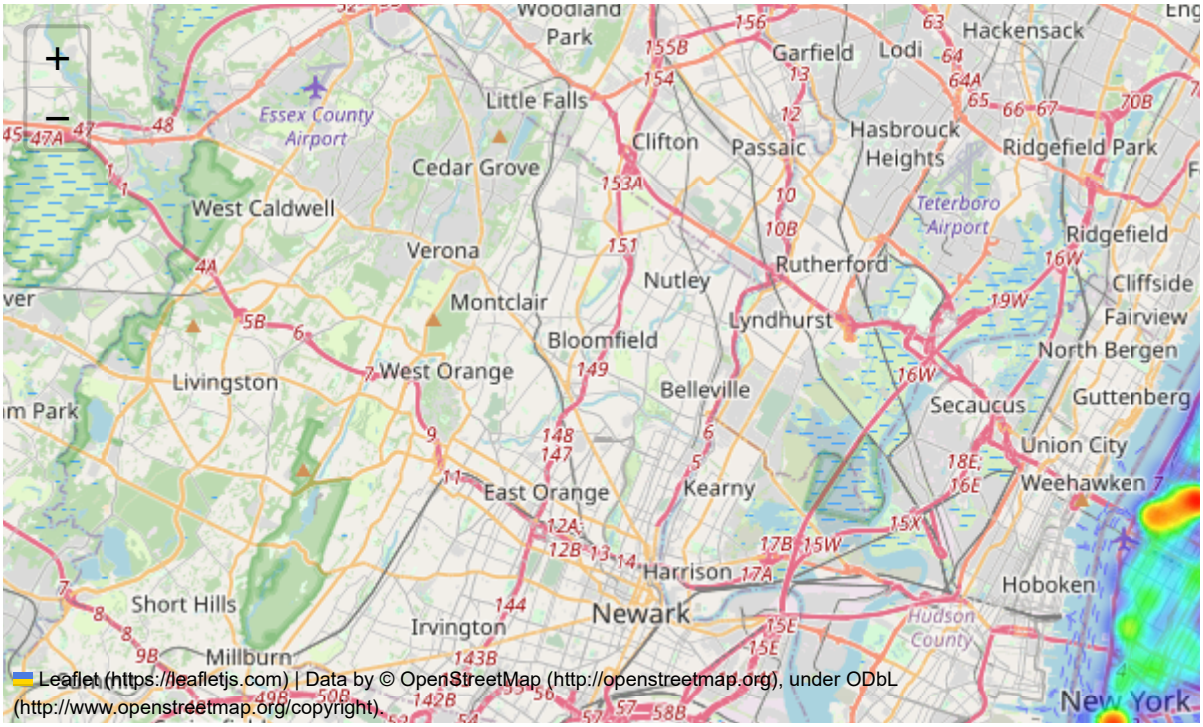
A continuación realizaremos dos HeatMaps de la ciudad de NewYork mostrando como los précios de la data original y la data para los outliers se distribuye en las diferentes zonas urbanas, así, con esto se pretende discernir si el alto valor de los outliers ha de deberse a un facor económico de negocio o puede por el contrario argumentarse que se trata de data errónea.

In [21]:
```
import folium
from folium.plugins import HeatMap

nueva_york = [40.781027,-73.965726]
map = folium.Map(location = nueva_york, zoom_start= 10,tiles="OpenStreetMap")
data = list(zip(airbnb['latitude'],
                airbnb['longitude'],
                airbnb['price'])) # Note que pasamos una tercera columna que es el pre

hm_loc = HeatMap(data,
              min_opacity = 0.2,
              radius = 8,
              blur = 6)

# Ahora añadimos la capa al mapa que ya habíamos creado
map.add_child(hm_loc)
map
```

Out[21]:

In [22]:
```
nueva_york = [40.781027,-73.965726]
map = folium.Map(location = nueva_york, zoom_start= 10,tiles="OpenStreetMap")
data = list(zip(price_outliers['latitude'],
                price_outliers['longitude'],
                price_outliers['price'])) # Note que pasamos una tercera columna que e

hm_loc = HeatMap(data,
              min_opacity = 0.2,
              radius = 8,
              blur = 6)

# Ahora añadimos la capa al mapa que ya habíamos creado
map.add_child(hm_loc)
map
```

Out[22]:



A pesar de que hasta éste momento no poseemos herramientas más potentes para discriminar los valores atípicos como data errónea o con ruido, al comparar el heatmap de la ciudad de NewYork para la data original con el de la data filtrada, puede verse que efectivamente los valores atípicos se ubican directamente en zonas altamente privilegiadas de Brooklin y Manhatan presentes en el distrito económico de la ciudad, lo cual nos indica que el alto valor de éstos hospedajes puede ser debido a su central posición en la zona urbana. Por ende, se considera que ésta información NO puede ser desechada por el momento debido a que la intuición ha de decirnos que la data es acorde a las necesidades y estrategias de negocio.

2.4-Análisis de valores atípicos inferiores

Debido a que el rango intercuartílico no es capaz de reportar outliers en la parte inferior de la distribución de precios y que un precio de cero dólares es altamente sospechoso, procedemos a observar cuántos hospedajes encontramos en la ciudad de NewYork por debajo de los 10 dolares.

In [23]:
```
lower_price = airbnb.loc[airbnb["price"]<11]
print("*********"*4)
print("Dimensiones del subconjunto de precios bajos= ",lower_price.shape)
print("cantidad de precios nulos=", (lower_price["price"]==0).sum())
print("cantidad de precios no nulos=", (lower_price["price"]>0).sum())
print("*********"*4)
```

```
**********************************
Dimensiones del subconjunto de precios bajos=  (16, 76)
cantidad de precios nulos= 6
cantidad de precios no nulos= 10
**********************************
```

A pesar de que la cantidad de hospedajes por igual o menos de 10 dolares en NewYork es de 16, exísten otros precios comparables que pueden ser opciones alternas (y tal vez más creibles

en términos de negocio) para estipular el precio mínimo de hospedaje, entre los cuales encontramos 15, 18 y 20 dolares.

In [24]:
```
lower_price2 = airbnb.loc[(airbnb['price'] < 21) & (airbnb['price'] >11)]
print("*********"*4)
print("Dimensiones del subconjunto de precios bajos= ",lower_price2.shape)

print("*********"*4)
lower_price2["price"]
```

```
**********************************
Dimensiones del subconjunto de precios bajos=  (44, 76)
**********************************
```

Out[24]:
```
1392     18
2194     16
3379     20
3603     16
5911     20
8294     20
10616    20
11429    20
11529    20
11666    20
11994    15
15080    20
15292    12
15425    12
15774    20
15822    15
16418    20
16555    20
17110    20
18486    15
18643    20
20972    20
21789    16
22088    19
22665    20
22754    20
22785    20
22969    20
22981    20
23158    19
23395    20
23422    20
23623    20
25885    20
26338    18
26341    20
26982    15
27010    20
27069    20
27658    20
28652    20
29386    15
29815    16
30126    20
Name: price, dtype: int64
```

De nuevo, carecemos en éste momento más de que herrmientas, de información concreta sobre el caso de estudio para poder discernir si estos precios altamente bajos de 10, 12 y 15 dolares o nulos, corresponden a situaciones reales donde el hospejade es extremadamente económico o en donde un hospedaje gratuito responde a necesidades sociales cubiertas por establecimientos. Por ejemplo, podría tratarse más de un hospedaje tipo hostal en donde el hospejade por noche no es cobrado monetariamente pero si con trabajo de mano de obra por parte del usuario. Así pues, no podemos simplemente eliminar ni siquiera los valores nulos de precio pues carecemos dela información suficiente para dar tal aseveración.

2.5-Data filtrada por valores inferiores

A pesar de que sería beneficioso poseer más información de caso sobre los valores atípicos inferiores, generaremos una nueva data sin éstos valores ya que representan una parte minuscula de la data original.

In [25]:
```
index_lower = list(airbnb[airbnb["price"] < 0].index)
airbnb2 = airbnb.drop(index_lower)
airbnb2.shape
```

Out[25]:
```
(30179, 76)
```

3)-Valores nulos

Una de las tareas obligatorias en esta fase de limpieza es detectar y explorar los valores nulos. Haga esa inspección con el dataframe `airbnb2` que acabamos de crear .

3.1-Tratamiento de valores nulos

En la presente subsección analizaremos el porcentaje de nulidad de cada una de nuestras variables, darémos una perspectiva a cerca de que está ocurriendo con cada una de ellas y propondrémos unos métodos muy sencillos para lidiar con los valores faltantes en nuéstra información deacuerdo a los porcentajes de nulidad de la variable y la relevancia de la mísma para el negocio en cuestión.

Procederemos a aumentar la calidad de la data en términos de los valores faltanes, para ello primero revisaremos los porcentajes de nulidad en cada variable catalogándolas de mayor a menor según éste criterio. Creamos una pequeña funcion que catalogue a las variables de la data deacuerdo a su porcentaje de nulidad.

In [26]:
```
def null_classifier(dataframe):
  series = dataframe.isnull().sum() * 100 / dataframe.shape[0]
  return series.sort_values(ascending=False)
```

In [27]:
```
null_classifier(airbnb2)[0:20]
```

Out[27]:
```
host_response_time             43.397727
host_response_rate             43.397727
transit                        39.726300
review_scores_location         30.259452
review_scores_value            30.252825
review_scores_checkin          30.249511
review_scores_accuracy         30.189867
review_scores_communication    30.186554
review_scores_cleanliness      30.156732
review_scores_rating           30.103715
reviews_per_month              27.370026
name                            0.043076
neighbourhood                   0.029822
host_since                      0.029822
host_identity_verified          0.029822
host_listings_count             0.029822
host_is_superhost               0.029822
accessible                      0.000000
high_end_electronics            0.000000
self_check_in                   0.000000
dtype: float64
```

In [28]:
```
null_var = null_classifier(airbnb2)[0:20]
cols = list(null_var.index)
airbnb2[cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30179 entries, 0 to 30178
Data columns (total 20 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   host_response_time           17082 non-null  object 
 1   host_response_rate           17082 non-null  float64
 2   transit                      18190 non-null  object 
 3   review_scores_location       21047 non-null  float64
 4   review_scores_value          21049 non-null  float64
 5   review_scores_checkin        21050 non-null  float64
 6   review_scores_accuracy       21068 non-null  float64
 7   review_scores_communication  21069 non-null  float64
 8   review_scores_cleanliness    21078 non-null  float64
 9   review_scores_rating         21094 non-null  float64
 10  reviews_per_month            21919 non-null  float64
 11  name                         30166 non-null  object 
 12  neighbourhood                30170 non-null  object 
 13  host_since                   30170 non-null  float64
 14  host_identity_verified       30170 non-null  float64
 15  host_listings_count          30170 non-null  float64
 16  host_is_superhost            30170 non-null  float64
 17  accessible                   30179 non-null  int64  
 18  high_end_electronics         30179 non-null  int64  
 19  self_check_in                30179 non-null  int64  
dtypes: float64(13), int64(3), object(4)
memory usage: 4.6+ MB
```

Nuestra data posee una enorme cantidad de valores nulos, exísten incluso variables con más de un 30% de nulidad,lo cual hace a éstas variables como candidades a eliminar de nuestro análisis. No obstante exísten variables que si pueden ser sensibles para la reputación del negocio como " review_scores_cleanliness" y "review_scores_communication", asi pues, procederemos a eliminar las variables con mas de 30% de nulidad pero que no son sensibles

para la reputación del hospedaje. Por otra parte, todas las variables categóricas que presenten una nulidad menor al 30% serán arregladas mediante la mediana de los datos. No obstante establecemos las siguientes estrategias deacuerdo al caso de negocio:

1. Aquellas variables que posean al menos 30% de nulidad serán eliminadas del dataset siempre y cuando representen variables sensibles que puedan influir en la percepcion de un cliente, como por ejemplo el puntaje atribuido a la locación del hospedaje, el puntaje dado a la comunicación ofrecida por el host, el puntaje de limpieza del hospedaje entre otras.

2. Una vez detectadas las variables con mayor nulidad y menos indispensables para la percepcion del negocio, detectamos las variables que si bien son sensibles para el negocio, no son lo suficientemente sensibles como para no tomar una postura más específica en cuanto a la reparación de la data. Esto quiere decir que variables enlistadas a continuación como "to_median" son variables que arreglaremos rellenando los espacios nulos con la mediana de los datos siendo estas, variables que reflejan una propiedad generalizada del hospedaje, como por ejemplo el puntaje en la comunicación, la limpieza, el rating, el número de reviews por mes entre otras.

3. Arreglaremos las demás variables mediante un llenado de ultimo valor válido para las variables más sensibles mediante las cuales queremos dar una perspectiva lo menos sesgada posible a un posible, donde variables como "review_scores_value", "neighbourhood" y "host_is_superhost" al ser respectivamente el puntaje dado por el cliente por medio de su review al la experiencia de hospedaje, el vecindario y la clasificación como superhost, son variables altamente sensibles para el negocio, por ende, si rellenamos con la mediana podríamos o bien falsa-positivamente beneficiar al hospedaje o bien falta-negativamente al negocio.

Para esta labor, nos alludaremos de las siguiente funcione:

In [29]:
```
def fill_null_values(dataframe, column_name, replace_by):
    if replace_by == "median":
        fill_value = dataframe[column_name].median()
    elif replace_by == "ffill":
        fill_value = None
    elif replace_by == "bfill":
        fill_value = None
    elif replace_by == "mean":
        fill_value = dataframe[column_name].mean()
    else:
        raise ValueError("Invalid value for replace_by. Please choose 'median', 'ffill

    if replace_by in ["ffill", "bfill"]:
        dataframe[column_name].fillna(method=replace_by, inplace=True)
    else:
        dataframe[column_name].fillna(fill_value, inplace=True)
```

Enlistamos las variables con nulidad deacuerdo a nuéstra estrategia de ingeniería de datos.

Antes de proceder, hacermos una copia de nuestros datos.

In [30]:
```
airbnb3 = airbnb2
```

In [31]:
```
to_eliminate = ["host_response_time","host_response_rate","review_scores_checkin","rev
to_median = ["review_scores_location","review_scores_communication","review_scores_cle
to_ffill = ["review_scores_value","neighbourhood","host_is_superhost"]
```

In [32]:
```
# Eliminamos las variables no relevantes
airbnb2 = airbnb2.drop(to_eliminate , axis=1)

# Rellenamos con la mediana
for variable in to_median:
      fill_null_values(airbnb2, variable, "median")

# Rellenamos de manera fordward
for variable in to_ffill:
      fill_null_values(airbnb2, variable, "ffill")
```

Finalmente, nos aseguramos que el procedimiento funcionara observando de nuevo el procentaje de nulidad.

In [33]:
```
null_classifier(airbnb2)
```

Out[33]:
```
name                          0.043076
instant_bookable              0.000000
tv                            0.000000
breakfast                     0.000000
bed_linen                     0.000000
                                ...
number_of_reviews             0.000000
number_of_reviews_ltm         0.000000
review_scores_rating          0.000000
review_scores_cleanliness     0.000000
event_suitable                0.000000
Length: 71, dtype: float64
```

4)-Transformación de variables

Ya hemos lidiado con datos atípicos y valores faltantes. Ahora procedamos a transformar nuestras características según las necesidades de cada variable. Vamos a seleccionar 11 variables para crear nuestro modelo basados en los análisis exploratorios realizados previamente. Recuerde, la variable objetivo es el precio. En la fase 3 construiremos un modelo que nos permita predecir el precio a partir de 10 variables predictoras. Vamos a preparar nuestros datos con ese propósito.

Para garantizar que todos trabajemos con los mismos datos, a continuación se entrega un dataset filtrado sin datos nulos y sin atípicos severos.

In [74]:
```
clean = pd.read_csv('https://github.com/HarryVargas96/UdeCataluna/blob/main/data/clear
clean.head()
```

Out[74]:

	price	neighbourhood	latitude	longitude	property_type	room_type	bathrooms	bedrooms	bed
0	149	Brooklyn	40.64749	-73.97237	Apartment	Private room	1.0	1	
1	150	Harlem	40.80902	-73.94190	Apartment	Private room	1.0	1	
2	190	Harlem	40.79685	-73.94872	Apartment	Entire home/apt	1.0	2	
3	60	Brooklyn	40.65599	-73.97519	Condominium	Private room	1.0	1	
4	80	Manhattan	40.86754	-73.92639	Apartment	Private room	1.0	1	

In [75]:
```
clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30173 entries, 0 to 30172
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   price              30173 non-null  int64  
 1   neighbourhood      30173 non-null  object 
 2   latitude           30173 non-null  float64
 3   longitude          30173 non-null  float64
 4   property_type      30173 non-null  object 
 5   room_type          30173 non-null  object 
 6   bathrooms          30173 non-null  float64
 7   bedrooms           30173 non-null  int64  
 8   beds               30173 non-null  int64  
 9   host_is_superhost  30173 non-null  float64
 10  parking            30173 non-null  int64  
dtypes: float64(4), int64(4), object(3)
memory usage: 2.5+ MB
```

4.1-Transformación de variables según su tipo

En la presente subsección analizarémos la tipología computacional (int, floeat, object) y la tipología estadística (ordinal, nominal, continúa, discreta) de las variables de la data limpia para seleccionar la transformación más adecuada para cada variable.

In [76]:
```
# Variables disponibles
sin_transformacion = ["price","neighbourhood","latitude","property_type","room_type",'

# Cualitativas - Categóricas
dummy_var = ["neighbourhood","host_is_superhost","parking"]
ordinal_encoding = ["property_type","room_type","bathrooms","bedrooms","beds"]

# Cuantitativas
normalizacion = ["latitude"]
estandarizacion = ["price"]
```

4.2-Variables categóricas

Filtramos las variables de la data "clean" deacuerdo asu tipología "object".

In [77]:
```
# Filtremos las variables tipo object
clean_object = clean.select_dtypes(include = 'object')
clean_object.head()
```

Out[77]:

	neighbourhood	property_type	room_type
0	Brooklyn	Apartment	Private room
1	Harlem	Apartment	Private room
2	Harlem	Apartment	Entire home/apt
3	Brooklyn	Condominium	Private room
4	Manhattan	Apartment	Private room

Consideremos ahora los niveles que pueden tomar cada una de las variables.

In [78]:
```
for variable in clean_object.columns:
  niveles = clean_object[variable].nunique()
  print('La variable {}, tiene {} niveles.'.format(variable,niveles))
```

```
La variable neighbourhood, tiene 186 niveles.
La variable property_type, tiene 33 niveles.
La variable room_type, tiene 3 niveles.
```

Las 3 variables son cualitativas ordinales, por lo que una transformación tipo dummy sería lo más apropiado, sin embargo, sí hacemos esto crearíamos un dataframe con 222 variables adicionales, esto puede entorpecer el trabajo con el dataframe. Por lo cual haremos lo siguiente:

1. Aplicar la transformación dummy a la variable room_type.
2. Aplicar ordinal encoding a las variables neighbourhood y property_type.

4.3-Transformación Dummy

Aplicamos la transformación dummy sobre el dataframe `clean` solamente para la variable room_type.

In [79]:
```
# Transformamos la variable room_type a dummy
clean_dummies = pd.get_dummies(clean, columns = ["room_type"])
clean_dummies
```

Out[79]:

	price	neighbourhood	latitude	longitude	property_type	bathrooms	bedrooms	beds	host_
0	149	Brooklyn	40.64749	-73.97237	Apartment	1.0	1	1	
1	150	Harlem	40.80902	-73.94190	Apartment	1.0	1	1	
2	190	Harlem	40.79685	-73.94872	Apartment	1.0	2	2	
3	60	Brooklyn	40.65599	-73.97519	Condominium	1.0	1	1	
4	80	Manhattan	40.86754	-73.92639	Apartment	1.0	1	1	
...	...	...	...	...	...	...	...	...	
30168	65	Jamaica	40.69137	-73.80844	House	2.0	4	3	
30169	70	Bedford-Stuyvesant	40.67853	-73.94995	Townhouse	1.0	1	2	
30170	40	Brooklyn	40.70184	-73.93317	Apartment	1.0	1	1	
30171	55	Hell's Kitchen	40.75751	-73.99112	Apartment	1.0	1	1	
30172	90	Midtown	40.76404	-73.98933	Apartment	2.0	1	1	

30173 rows × 13 columns

4.4-Transformación ordinal

Aplicamos la transformación ordinal sobre las variables neighbourhood y property_type. Nombramos el tranformador ordinal como `encoder` y guardamos el resultado en un objeto llamado `ord_vars`.

Haciendo uso de la librería sklearn, generamos un codificador ordinal para un dataframe de pandas en la siguiente forma:

In [80]:
```
from sklearn.preprocessing import OrdinalEncoder

# Seleccionamos las variables a codificar
var_to_encode = ["neighbourhood","property_type"]
data_to_encode = clean_dummies[var_to_encode]

# Creamos el codificador y aplicamos la ransformacion a la data
encoder = OrdinalEncoder()
encoded_data = encoder.fit_transform(data_to_encode)
encoded_data
```

Out[80]:
```
array([[ 18.,   1.],
       [ 80.,   1.],
       [ 80.,   1.],
       ...,
       [ 18.,   1.],
       [ 81.,   1.],
       [111.,   1.]])
```

Por otra parte, podemos usar category_encoders desde el modulo de OrdinalEncoders para hacer una forma más facilmente personalizada del codificador. Instalamos el módulo.

In [81]:
```
!pip install category_encoders
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: category_encoders in /usr/local/lib/python3.10/dist-packages (2.6.1)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.22.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.10.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.13.5)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2022.7.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (3.1.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encoders) (23.1)
```

In [82]:
```
from category_encoders.ordinal import OrdinalEncoder
```

In [83]:
```
# Extraemos los valores categóricos unicos
neighbourhood_2 = list(clean_dummies["neighbourhood"].unique())
prop_type_2 = list(clean_dummies["property_type"].unique())

# Generamos un dictionario para hacer un par llave-valor que respectivamente simbolice
mapping_neighbourhood_2 = {value: index for index, value in enumerate(neighbourhood_2)}
mapping_prop_type_2 = {value: index for index, value in enumerate(prop_type_2)}

# Planteamos el mapeo para el encoder en forma de una lista de disccionarios
mapping_var = [
    {
        "col": "neighbourhood",
        "mapping": mapping_neighbourhood_2
    },
    {
        "col": "property_type",
        "mapping": mapping_prop_type_2
    }]

# Insertamos el mapeo en el codificador y transformamos la data

encoder_2 = OrdinalEncoder(mapping = mapping_var)
```

```
encoded_data_2 = encoder.fit_transform(data_to_encode)
encoded_data_2
```

Out[83]:
```
array([[ 18.,   1.],
       [ 80.,   1.],
       [ 80.,   1.],
       ...,
       [ 18.,   1.],
       [ 81.,   1.],
       [111.,   1.]])
```

Ejemplo 1

Más información acerca de los codificadores Ordinales, solo ejecute el resto de este código cuando haya obtenido la respuesta de arriba.

El método Ordinal Encoder de scikitlearn organiza las categorías en orden alfabético. A continuación se muestra el objeto que contiene la información sobre la codificación.

In [84]:
```
encoder.categories_

# Retorna dos listas con dos arreglos de numpy
# Cada arreglo contiene las categorías de cada variable transformada
```

Out[84]:
```
[array(['Allerton', 'Alphabet City', 'Annadale', 'Astoria', 'Bath Beach',
       'Battery Park City', 'Bay Ridge', 'Baychester', 'Bayside',
       'Bedford Park', 'Bedford-Stuyvesant', 'Belmont', 'Bensonhurst',
       'Bergen Beach', 'Boerum Hill', 'Borough Park', 'Brighton Beach',
       'Bronxdale', 'Brooklyn', 'Brooklyn Heights', 'Brooklyn Navy Yard',
       'Brownsville', 'Bushwick', 'Canarsie', 'Carroll Gardens',
       'Castle Hill ', 'Castleton Corners', 'Chelsea', 'Chinatown',
       'City Island', 'Civic Center', 'Claremont', 'Clinton Hill',
       'Cobble Hill', 'College Point', 'Columbia Street Waterfront',
       'Concord', 'Concourse', 'Concourse Village', 'Coney Island',
       'Corona', 'Crotona', 'Crown Heights', 'DUMBO',
       'Ditmars / Steinway', 'Dongan Hills', 'Downtown Brooklyn',
       'Dyker Heights', 'East Elmhurst', 'East Flatbush', 'East Harlem',
       'East New York', 'East Village', 'Eastchester', 'Edenwald',
       'Elm Park', 'Elmhurst', 'Financial District', 'Flatbush',
       'Flatiron District', 'Flatlands', 'Flushing', 'Fordham',
       'Forest Hills', 'Fort Greene', 'Fort Hamilton', 'Fresh Meadows',
       'Glendale', 'Gowanus', 'Gramercy Park', 'Graniteville',
       'Grant City', 'Grasmere', 'Gravesend', 'Great Kills', 'Greenpoint',
       'Greenwich Village', 'Greenwood Heights', 'Grymes Hill',
       'Hamilton Heights', 'Harlem', "Hell's Kitchen", 'Highbridge',
       'Hillcrest', 'Howard Beach', 'Hudson Square', 'Hunts Point',
       'Inwood', 'Jackson Heights', 'Jamaica', 'Kensington',
       'Kew Garden Hills', 'Kingsbridge', 'Kingsbridge Heights',
       'Kips Bay', 'Lefferts Garden', 'Lindenwood', 'Little Italy',
       'Long Island City', 'Longwood', 'Lower East Side', 'Manhattan',
       'Manhattan Beach', 'Marble Hill', 'Mariners Harbor', 'Maspeth',
       'Meatpacking District', 'Meiers Corners', 'Melrose',
       'Middle Village', 'Midland Beach', 'Midtown', 'Midtown East',
       'Midwood', 'Morningside Heights', 'Morris Heights', 'Morris Park',
       'Morrisania', 'Mott Haven', 'Murray Hill', 'New Brighton',
       'New Dorp Beach', 'New Springville', 'Noho', 'Nolita', 'Norwood',
       'Oakwood', 'Ozone Park', 'Park Slope', 'Park Versailles',
       'Parkchester', 'Pelham Bay', 'Port Morris', 'Port Richmond',
       "Prince's Bay", 'Prospect Heights', 'Queens', 'Red Hook',
       'Rego Park', 'Richmond Hill', 'Ridgewood', 'Riverdale',
       'Roosevelt Island', 'Rosebank', 'Sea Gate', 'Sheepshead Bay',
       'Soho', 'Soundview', 'South Beach', 'South Ozone Park',
       'South Street Seaport', 'Spuyten Duyvil', 'St. George',
       'Stapleton', 'Staten Island', 'Sunnyside', 'Sunset Park',
       'The Bronx', 'The Rockaways', 'Throgs Neck',
       'Times Square/Theatre District', 'Tompkinsville', 'Tottenville',
       'Tremont', 'Tribeca', 'Union Square', 'University Heights',
       'Upper East Side', 'Upper West Side', 'Utopia', 'Van Nest',
       'Vinegar Hill', 'Wakefield', 'Washington Heights', 'West Brighton',
       'West Farms', 'West Village', 'Westchester Village', 'Westerleigh',
       'Whitestone', 'Williamsbridge', 'Williamsburg', 'Windsor Terrace',
       'Woodhaven', 'Woodlawn', 'Woodside'], dtype=object),
 array(['Aparthotel', 'Apartment', 'Barn', 'Bed and breakfast', 'Boat',
       'Boutique hotel', 'Bungalow', 'Cabin', 'Camper/RV',
       'Casa particular (Cuba)', 'Castle', 'Cave', 'Condominium',
       'Cottage', 'Dome house', 'Earth house', 'Farm stay', 'Guest suite',
       'Guesthouse', 'Hostel', 'Hotel', 'House', 'Houseboat', 'Loft',
       'Other', 'Resort', 'Serviced apartment', 'Tent', 'Tiny house',
       'Townhouse', 'Treehouse', 'Villa', 'Yurt'], dtype=object)]
```

Creemos diccionarios en formato csv con la codificación de ambas variables. El proceso de documentación es clave en el proyecto.

In [85]:
```
# Extraemos el arreglo relacionado con los barrios

categories_neigh = encoder.categories_[0]

# Creamos una lista con números igual a la longitud del arreglo de los barrios

neigh_codes = [i for i in range(len(categories_neigh))]

# Creamos un dataframe con los barrios y códigos

barrios = pd.DataFrame({'codigo': neigh_codes,
                        'barrio': categories_neigh})

# Creamos un archivo csv para la documentación del proceso

barrios.to_csv('Diccionario barrios airbnb.csv')

barrios.head()
```

Out[85]:

	codigo	barrio
0	0	Allerton
1	1	Alphabet City
2	2	Annadale
3	3	Astoria
4	4	Bath Beach

In [86]:
```
# Extraemos el arreglo relacionado con el tipo de propiedad

categories_property = encoder.categories_[1]

# Creamos una lista con números igual a la longitud del arreglo de los tipos de propie

property_codes = [i for i in range(len(categories_property))]

# Creamos un dataframe con los tipos de propiedad y códigos

propiedades = pd.DataFrame({'codigo': property_codes,
                        'barrio': categories_property})

# Creamos un archivo csv para la documentación del proceso

propiedades.to_csv('Diccionario tipos de propiedad airbnb.csv')
propiedades.head()
```

Out[86]:

	codigo	barrio
0	0	Aparthotel
1	1	Apartment
2	2	Barn
3	3	Bed and breakfast
4	4	Boat

Para finalizar, reemplazamos las variables transformadas en nuestro dataframe original. Y damos un vistazo a nuestro dataframe con todas las variables codificadas como númericas.

In [94]:
```
# Convertimos el arreglo de numpy en un dataframe
ord_vars = pd.DataFrame(encoded_data,columns = ['neighbourhood','property_type'])
ord_vars.head()
```

Out[94]:

	neighbourhood	property_type
0	18.0	1.0
1	80.0	1.0
2	80.0	1.0
3	18.0	12.0
4	101.0	1.0

In [96]:
```
# Reemplazamos en nuestro dataset original

for variable in ord_vars.columns:
  clean[variable] = ord_vars[variable]

clean.head()
```

Out[96]:

	price	neighbourhood	latitude	longitude	property_type	room_type	bathrooms	bedrooms	bed
0	149	18.0	40.64749	-73.97237	1.0	Private room	1.0	1	
1	150	80.0	40.80902	-73.94190	1.0	Private room	1.0	1	
2	190	80.0	40.79685	-73.94872	1.0	Entire home/apt	1.0	2	
3	60	18.0	40.65599	-73.97519	12.0	Private room	1.0	1	
4	80	101.0	40.86754	-73.92639	1.0	Private room	1.0	1	

4.5-Unicidad de formato

Observe la variables parking y host_is_superhost son binarias. Solo toman valores de Falso (0) y Verdadero (1). Sin embargo, la variable parking tiene una codificación extraña con -1 como código de Falso, lo cual que puede ocasionar algunos inconvenientes. Encuentre todos los campos de la variable parking que contengan -1.0 y reemplacelos por 0.0.

In [97]:
```
clean_copy = clean
clean_copy.head()
```

Out[97]:

	price	neighbourhood	latitude	longitude	property_type	room_type	bathrooms	bedrooms	bed
0	149	18.0	40.64749	-73.97237	1.0	Private room	1.0	1	
1	150	80.0	40.80902	-73.94190	1.0	Private room	1.0	1	
2	190	80.0	40.79685	-73.94872	1.0	Entire home/apt	1.0	2	
3	60	18.0	40.65599	-73.97519	12.0	Private room	1.0	1	
4	80	101.0	40.86754	-73.92639	1.0	Private room	1.0	1	

In [98]:
```
clean["parking"] = clean["parking"].map(lambda x : 0 if x < 0 else x)
clean.head()
```

Out[98]:

	price	neighbourhood	latitude	longitude	property_type	room_type	bathrooms	bedrooms	bed
0	149	18.0	40.64749	-73.97237	1.0	Private room	1.0	1	
1	150	80.0	40.80902	-73.94190	1.0	Private room	1.0	1	
2	190	80.0	40.79685	-73.94872	1.0	Entire home/apt	1.0	2	
3	60	18.0	40.65599	-73.97519	12.0	Private room	1.0	1	
4	80	101.0	40.86754	-73.92639	1.0	Private room	1.0	1	

4.6-Conclusiones finales

En esta fase de limpieza y transformación de los datos, empleamos tanto técnicas vístas en clase como conocimiento de negocio para ajustar la data en un formato más conciso y correcto para una posterior implementación de modelo de Mchine Learning. Con éste propósito, ejecutamos dos fases principales para este ejercicio de ingeniería de datos: limpieza de los datos y transformación de los datos.

En la fase de limpieza de datos ejecutamos en orden los siguientes procedimientos:

1. En términos de corrección de errores y manejo de valores atípicos, eliminamos los valores atípicos que superaran 1.5 veces el rango intercuartílico, por otra parte eliminamos los valores de hospedaje nulo al no tener sentido en el contexto del negocio. Pudimos deducir que los valores del precio del hospedaje atípicos correspondían a hospedajes cuya localización era altamente privilegiada.
2. En términos de corrección de la data según su nulidad (manejo de valores nulos), procedimos a ejecutar tres procedimientos: eliminación de variable, relleno por media y fordwad fill. En este apartado analizamos el porcentaje de nulidad en conjunto con la importancia de la variable para el negocio para poder seleccionar la estrategia más adecuada para cada variable.
3. En términos de transformación de variables, en la presente práctica nos enfocamos únicamente en la enumeración dummy para variables categóricas con pocas clases y en la transformación ordinal para variables categóricas con un gran número de clases.
4. Finalmente, se detectó que una de las variables del dataset a pesar de estar enumerada y poseer una cantidad binaria de clases, presentaba un formato extraño en la presentación de su información, así pues, el paso final efectuado fue la corrección del formato anómalo.

No obstante, ya que el curso del presente proyecto se centra en ejecutar algoritmos de Machine Learning a la deducción (regresion) del precio de los hospedajes, se considera apropiado que dos transformaciónes más fuera efectuada: la estandarización del precio y la numerización ordinal del tipo de habitación.

In [112…

```
from sklearn.preprocessing import OrdinalEncoder

# Seleccionamos las variables a codificar
var_to_encode = ["room_type"]
data_to_encode = clean[var_to_encode]

# Creamos el codificador y aplicamos la ransformacion a la data
encoder = OrdinalEncoder()
encoded_data = encoder.fit_transform(data_to_encode)

# Convertimos el arreglo de numpy en un dataframe
ord_var = pd.DataFrame(encoded_data,columns = ["room_type"])

clean["room_type"] = ord_var["room_type"]
```

In [113…

```
from sklearn import preprocessing

# Creamos un objeto tipo escaler
scaler = preprocessing.StandardScaler(with_mean=True, with_std=True)
# Ejecutamos la transformación
price_data = clean["price"].values.reshape(-1,1)
scaled_price = scaler.fit_transform(price_data)
price_data = pd.DataFrame(scaled_price, columns =["price"])

# Remplazamos la data estandarizada dentro de nuestro dataframe
clean["price"] = price_data
clean.head()
```

Out[113]:

	price	neighbourhood	latitude	longitude	property_type	room_type	bathrooms	bedrooms
0	0.172037	18.0	40.64749	-73.97237	1.0	1.0	1.0	1
1	0.182773	80.0	40.80902	-73.94190	1.0	1.0	1.0	1
2	0.612231	80.0	40.79685	-73.94872	1.0	0.0	1.0	2
3	-0.783507	18.0	40.65599	-73.97519	12.0	1.0	1.0	1
4	-0.568778	101.0	40.86754	-73.92639	1.0	1.0	1.0	1

Aquí termina la segunda fase del proyecto, ya conocemos bastante nuestros datos y tenemos un conjunto limpio y preparado para la fase de modelamiento. En la siguiente fase vamos a utilizar dos algoritmos de Machine Learning para construir 2 modelos, haremos el proceso de experimentación para optimizar los errores y extraeremos información valiosa de nuestros modelos.

Créditos

Profesor: Harry Vargas Rodríguez

Corporación Universitaria de Cataluña - *Diplomado en Big Data y Data Science*