

▼ Ciencia de Datos Aplicada MINE-4101 - PARCIAL 1

Universidad de los Andes

Realizado por: Yeimy A. Cano M.

“Al entregar la solución de este parcial, yo, Yeimy A Cano con código 202213304 me comprometo a no conversar durante el desarrollo de este examen con ninguna persona que no sea el profesor del curso, sobre aspectos relacionados con el parcial; tampoco utilizaré algún medio de comunicación por voz, texto o intercambio de archivos, para consultar o compartir con otros, información sobre el tema del parcial. Soy consciente y acepto las consecuencias que acarreará para mi desempeño académico cometer fraude en este parcial”.

▼ 0. Importación de librerías de trabajo

Primero se procede a hacer la importación de librerías para poder trabajar con los datos y poder realizar el análisis.

```
!pip install --upgrade pandas-profiling
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: pandas-profiling in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: statsmodels<0.14,>=0.13.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests<2.29,>=2.24.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: visions[type_image_path]==0.7.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: seaborn<0.12,>=0.10.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: matplotlib<3.6,>=3.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib~=1.1.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas!=1.4.0,<1.5,>1.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy<1.24,>=1.16.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy<1.10,>=1.4.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: multimethod<1.9,>=1.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: missingno<0.6,>=0.4.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tqdm<4.65,>=4.48.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tangled-up-in-unicode==0.2.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pydantic<1.10,>=1.8.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: imagehash in /usr/local/lib/python3.7/dist-packages (from vimagehash)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from vimagehash)
```

```
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from pylev)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.7/dist-packages (from pylev)
```



```
!pip install pylev
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Requirement already satisfied: pylev in /usr/local/lib/python3.7/dist-packages (1.4.0)
```



```
import numpy as np
import pandas as pd

from pandas_profiling import ProfileReport

import seaborn as sns
from scipy import stats
import scipy
import statsmodels.api as sm

#Entrenamiento del modelo
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures , OneHotEncoder, MinMaxScaler
from sklearn.linear_model import LinearRegression,Lasso, Ridge
from sklearn.base import BaseEstimator,TransformerMixin
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
%matplotlib inline
plt.rcParams['figure.dpi'] = 110
import pylev

import itertools
```

```
!python --version
print('NumPy', np.__version__)
print('pandas', pd.__version__)
print('SciPy', scipy.__version__)
print('statsmodels', sm.__version__)
print('Matplotlib', mpl.__version__)
print('Seaborn', sns.__version__)
```

Python 3.7.14

NumPy 1.21.6

pandas 1.3.5

SciPy 1.7.3

statsmodels 0.13.2

Matplotlib 3.2.2

Seaborn 0.11.2

▼ Entendimiento del Negocio

a. Objetivos del negocio y situación actual.

Inmobiliaria Los Alpes es una inmobiliaria con varios años de trayectoria la cual se especializa en el arrendamiento de inmuebles de corta estancia. La empresa ha ido creciendo poco a poco y en la actualidad enfrenta una serie de retos relacionados con la gran cantidad de competencia que hay en el sector, dificultando la obtención de nuevos inmuebles para su administración como arrendador, así como conseguir los arrendatarios idóneos para el uso de los mismos.

Como estrategia para mantener su competitividad en el mercado, la Inmobiliaria Los Alpes ha decidido enfocarse en determinar los inmuebles que podrían llegar a ser populares desde antes de su publicación. La popularidad de un inmueble está dada en términos de la cantidad de comentarios (reviews) que ha obtenido independientemente de su calificación. Esta actividad es importante dado que los inmuebles más populares son los que generan mayor rotación, mientras que para los de menor popularidad hay que hacer una mayor inversión en marketing para promocionarlos.

La inmobiliaria busca contratar con usted un proyecto de consultoría que tiene como objetivo validar el uso de técnicas de machine learning que apoyen la toma de decisiones del equipo de marketing respecto a los inmuebles para los que deben hacer inversiones en promoción dada su baja popularidad. Le han compartido dos conjuntos de datos: (1) el histórico de sus inmuebles con su respectiva popularidad y (2) los inmuebles que están próximos a publicarse. Adicionalmente, le han compartido el diccionario de datos correspondiente.

▼ Entendimiento de los datos

▼ a. Adquisición e integración de datos.

El archivo de datos está como archivo de CSV, se hace una exploración con una muestra de datos "losalpes_history.csv"

Para trabajar con los datos, éstos se ubican en una ruta de github.

```
# Loading data historico
url = 'https://raw.githubusercontent.com/yacanom/CDA_MINE-4101_repo/main/Parcial%201/losalpes_history.csv'
data = pd.read_csv(url, sep=',',)

# Loading data losalpes_new.csv
url2 = 'https://raw.githubusercontent.com/yacanom/CDA_MINE-4101_repo/main/Parcial%201/losalpes_new.csv'
dataP = pd.read_csv(url2, sep=',',)
```

b. Diccionario de datos.

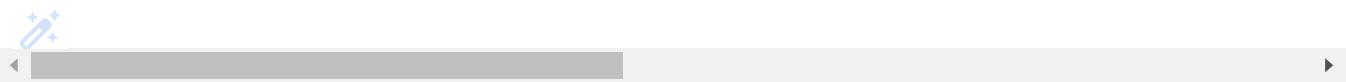
De acuerdo a la información entregada, el set de datos cuentan con los siguientes columnas con la respectiva descripción:

Field	Description
id	Identificador del inmueble
neighbourhood group	Localidad o distrito en el que se encuentra el inmueble
neighbourhood	Barrio en el que se encuentra el inmueble
lat, long	Geolocalización del inmueble
country	Pais en el que se encuentra el inmueble
instant bookable	Indicador de si es posible realizar reserva directamente en la plataforma
cancellation_policy	Política de cancelacion de la reserva
room type	Tipo de inmueble
construction year	Año de construcción del inmueble
price	Precio por noche del inmueble
service fee	Costo del servicio el cual debe ser cancelado al dejar el inmueble
minimum nights	Cantidad mínima de noches que el inmueble puede ser reservado
availability 365	Disponibilidad total en dias durante el ultimo año
number of reviews	Total de comentarios del inmueble
review rate number	Calificación promedio dada al inmueble

▼ c. Exploración de datos.

```
data.head()
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_bookable
0	48540006	Manhattan	Hell's Kitchen	40.76212	-73.98820	United States	True
1	35079903	Manhattan	Midtown	40.74623	-73.98499	United States	True
2	50681273	Manhattan	Upper West Side	40.78859	-73.97568	United States	False
3	13039267	Manhattan	Financial District	40.70817	-74.00511	United States	False
4	8998640	Manhattan	Lower East Side	40.72130	-73.98900	United States	True



Vamos a hacer una exploración sobre el set de datos históricos, con `data.info` se puede hacer una ojeada muy superficial para conocer tipos de datos, cantidad, sin tener mucha información detallada que entienda el modelo.

```
data.shape
```

```
(102083, 16)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 102083 entries, 0 to 102082
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id               102083 non-null   int64  
 1   neighbourhood group  101463 non-null   object  
 2   neighbourhood      101476 non-null   object  
 3   lat                101484 non-null   float64 
 4   long               101484 non-null   float64 
 5   country             100967 non-null   object  
 6   instant_bookable    101387 non-null   object  
 7   cancellation_policy 101416 non-null   object 
```

```

8   room type          101492 non-null  object
9   construction year 101279 non-null  float64
10  price              101245 non-null  object
11  service fee         101220 non-null  object
12  minimum nights     101089 non-null  float64
13  availability 365   101049 non-null  float64
14  number of reviews   101313 non-null  float64
15  review rate number 101174 non-null  float64
dtypes: float64(7), int64(1), object(8)
memory usage: 12.5+ MB

```

De esta revisión se puede observar que hay datos faltantes, de acuerdo al tamaño del dataframe deberían haber 102083 registros pero solo el id, presenta este número completo, los demás presentan datos pendientes.

```
data.describe()
```

	id	lat	long	construction year	minimum nights	availabi
count	1.020830e+05	101484.000000	101484.000000	101279.000000	101089.000000	101049.00
mean	2.914263e+07	40.728070	-73.94981	2012.459454	8.151698	141.07
std	1.625930e+07	0.055829	0.04936	7.895321	30.665643	135.46
min	1.001254e+06	40.499790	-74.24984	1020.000000	-1223.000000	-10.00
25%	1.507753e+07	40.688750	-73.98261	2007.000000	2.000000	3.00
50%	2.913163e+07	40.722270	-73.95456	2012.000000	3.000000	96.00
75%	4.320009e+07	40.762730	-73.93260	2018.000000	5.000000	268.00

Ahora usamos ProfileReport para poder hacer la exploración sobre el data set de entrenamiento (train), para no dar la información del dataset completo.

```
ProfileReport(data)
```

Summarize dataset:	80/80 [00:22<00:00, 3.83it/s,
100%	Completed]
Generate report structure: 100%	1/1 [00:06<00:00, 6.69s/it]
Render HTML: 100%	1/1 [00:02<00:00, 2.36s/it]
1006307	1 < 0.1%

Value	Count	Frequency (%)
57367417	1	< 0.1%
57366865	1	< 0.1%
57366313	1	< 0.1%
57365760	1	< 0.1%
57365208	1	< 0.1%
57364656	1	< 0.1%
57364103	1	< 0.1%
57363551	1	< 0.1%
57362999	1	< 0.1%
57362446	1	< 0.1%

De acuerdo a la exploración realizada con el reporte, se puede ver que:

- En la columna ***id*** se observan que no todos los datos son únicos lo que sugiere se tienen datos duplicados
- ***neighbourhood group***: falta el 0.6% de los datos. Hay 3 categorías que predominan los datos.
- ***neighbourhood***: falta el 0.6% de los datos
- ***lat***: Falta el 0.6% de los datos, así como en ***long***
- ***country***: Esta de dos formas diferente el mismo país United States y United States of America
- ***instant_bookable***: falta el 0.7% de los datos
- ***cancellation_policy***: falta el 0.7% de los datos

- **room type**: falta el 0.6% de los datos. Dos categorías predominan: "Entire home/apt" y "Private room"
- **construction year**: Se observa que hay valores atípicos (3), años 1020 y 1022
- **price**: como tiene el carácter "\$" no se tiene cómo número, es mejor quitarlo y pasarlo a numérico. Hay signos de - con lo que se tendrían valores negativos y falta el 0.8% de los datos
- **service fee**: como tiene el carácter "\$" no se tiene cómo número, es mejor quitarlo y pasarlo a numérico, falta el 0.8% de los datos
- **minimum nights**: hay 13 registros como valores negativos, falta el 1% de los datos, aunque el 95percentil está 30, se evidencian valores máximos de 5645.
- **availability 365**: falta el 1% de los datos, hay 428 registros negativos (0.4%), es un tipo de dato sobre los días disponibles en el último año luego el valor máximo debería ser 365, pero se encuentra un máximo de 3677. El 95 percentil es de 365.
- **number of reviews**: falta el 0.8% de los datos. Se observa una desviación estándar del 49.47, así como que el 95 percentil es de 125, pero los 10 valores más altos son superiores a 678.
- **review rate number**: falta el 0.9% de los datos, la mayoría de los datos se distribuye similar del 2.0-5.0 pero son menores los valores de 1.0

Revisión de los datos duplicados:

```
data[data.duplicated()]
```

		id	neighbourhood group	neighbourhood	lat	long	country	instant_bc
4084		35568136	Brooklyn	Bedford-Stuyvesant	40.69492	-73.93883	United States	
6140		20378229	Manhattan	Harlem	40.80624	-73.95622	United States	
8017		35520638	Brooklyn	Prospect Heights	40.68246	-73.97602	United States	
10177		6079182	Brooklyn	Park Slope	40.67792	-73.98035	United States	

Se identificaron 536 datos duplicados.

Ahora al querer revisar el listado de distritos mas populares:

```
df_top10 = data[{"neighbourhood group", "neighbourhood"}].groupby("neighbourhood group").cour
df_top10
```

	neighbourhood group	Total canciones	
0	Manhattan	43376	
1	Brooklyn	41429	
2	Queens	13015	
3	Bronx	2666	
4	Staten Island	943	
5	Broolkyn	7	
6	Manhatan	4	
7	Quens	4	
8	Manattan	1	
9	brookln	1	

```
data["neighbourhood group"].unique()
```

```
array(['Manhattan', 'Brooklyn', 'Queens', 'Bronx', 'Staten Island', nan,
       'Quens', 'Manattan', 'Broolkyn', 'Manhatan', 'brookln', 'manhattan'],
      dtype=object)
```

Se evidencia que hay casos de registros en donde quedó mal registrado el distrito como en el

```
#Se revisan las coordenadas de los distritos para ver si de esta forma se podrían imputar los  
data[data['neighbourhood group'] == 'Manhattan']['lat'].min()
```

```
print(f"Manhattan Latitud min y max:%s", data[data['neighbourhood group'] == 'Manhattan']['lat'].min())  
print(f"Manhattan Longitud min y max:%s", data[data['neighbourhood group'] == 'Manhattan']['long'].min())
```

```
print(f"Queens Latitud min y max:%s", data[data['neighbourhood group'] == 'Queens']['lat'].min())  
print(f"Queens Longitud min y max:%s", data[data['neighbourhood group'] == 'Queens']['long'].min())
```

```
print(f"Bronx Latitud min y max:%s", data[data['neighbourhood group'] == 'Bronx']['lat'].min())  
print(f"Bronx Longitud min y max:%s", data[data['neighbourhood group'] == 'Bronx']['long'].min())
```

```
print(f"Staten Island Latitud min y max:%s", data[data['neighbourhood group'] == 'Staten Island']['lat'].min())  
print(f"Staten Island Longitud min y max:%s", data[data['neighbourhood group'] == 'Staten Island']['long'].min())
```

```
Manhattan Latitud min y max:%s 40.70234 40.87821  
Manhattan Longitud min y max:%s -74.01851 -73.90855  
Queens Latitud min y max:%s 40.56546 40.79721  
Queens Longitud min y max:%s -73.95953 -73.70522  
Bronx Latitud min y max:%s 40.80011 40.91697  
Bronx Longitud min y max:%s -73.93296 -73.78158  
Staten Island Latitud min y max:%s 40.49979 40.64816  
Staten Island Longitud min y max:%s -74.24984 -74.06092
```

```
data[data['neighbourhood group'].isnull() & data['lat'].isnull() ]
```

```
      id neighbourhood
      group neighbourhood lat long country instant_bookable
```

Hay 591 casos donde el neighbourhood group no tiene datos, pero tampoco se tienen los valores en las demás columnas, de los cuales no se justifica imputarles valores.

```
    id neighbourhood
    group neighbourhood lat long country instant_bookable
```

Ahora el top 10 entre los 224 barrios:

```
449 28979197      NaN      NaN      NaN      NaN      NaN      NaN
```

```
df_top10 = data[{"neighbourhood group", "neighbourhood"}].groupby("neighbourhood").count().sc
df_top10
```

	neighbourhood	Total sitios	edit
0	Bedford-Stuyvesant	7856	
1	Williamsburg	7715	
2	Harlem	5397	
3	Bushwick	4927	
4	Hell's Kitchen	3927	
5	Upper West Side	3830	
6	Upper East Side	3637	
7	East Village	3461	
8	Midtown	3369	
9	Crown Heights	3223	

```
data.loc[:, 'neighbourhood'].sort_values().unique()
```

```
array(['Allerton', 'Arden Heights', 'Arrochar', 'Arverne', 'Astoria',
       'Bath Beach', 'Battery Park City', 'Bay Ridge', 'Bay Terrace',
       'Bay Terrace, Staten Island', 'Baychester', 'Bayside', 'Bayswater',
       'Bedford-Stuyvesant', 'Belle Harbor', 'Bellerose', 'Belmont',
       'Bensonhurst', 'Bergen Beach', 'Boerum Hill', 'Borough Park',
       'Breezy Point', 'Briarwood', 'Brighton Beach', 'Bronxdale',
       'Brooklyn Heights', 'Brownsville', "Bull's Head", 'Bushwick',
       'Cambria Heights', 'Canarsie', 'Carroll Gardens', 'Castle Hill',
       'Castleton Corners', 'Chelsea', 'Chelsea, Staten Island',
       'Chinatown', 'City Island', 'Civic Center', 'Clarendon Village',
       'Clason Point', 'Clifton', 'Clinton Hill', 'Co-op City',
       'Cobble Hill', 'College Point', 'Columbia St', 'Concord',
       'Concourse', 'Concourse Village', 'Coney Island', 'Corona',
       'Crown Heights', 'Cypress Hills', 'DUMBO', 'Ditmars Steinway',
       'Dongan Hills', 'Douglaslaston', 'Downtown Brooklyn', 'Dyker Heights',
       'East Elmhurst', 'East Flatbush', 'East Harlem', 'East Morrisania',
       'East New York', 'East Village', 'Eastchester', 'Edenwald',
```

```
'Edgemere', 'Elmhurst', 'Eltingville', 'Emerson Hill',
'Far Rockaway', 'Fieldston', 'Financial District', 'Flatbush',
'Flatiron District', 'Flatlands', 'Flushing', 'Fordham',
'Forest Hills', 'Fort Greene', 'Fort Hamilton', 'Fort Wadsworth',
'Fresh Meadows', 'Gerritsen Beach', 'Glen Oaks', 'Glendale',
'Gowanus', 'Gramercy', 'Graniteville', 'Grant City', 'Gravesend',
'Great Kills', 'Greenpoint', 'Greenwich Village', 'Grymes Hill',
'Harlem', "Hell's Kitchen", 'Highbridge', 'Hollis', 'Holliswood',
'Howard Beach', 'Howland Hook', 'Huguenot', 'Hunts Point',
'Inwood', 'Jackson Heights', 'Jamaica', 'Jamaica Estates',
'Jamaica Hills', 'Kensington', 'Kew Gardens', 'Kew Gardens Hills',
'Kingsbridge', 'Kips Bay', 'Laurelton', 'Lighthouse Hill',
'Little Italy', 'Little Neck', 'Long Island City', 'Longwood',
'Lower East Side', 'Manhattan Beach', 'Marble Hill',
'Mariners Harbor', 'Maspeth', 'Melrose', 'Middle Village',
'Midland Beach', 'Midtown', 'Midwood', 'Mill Basin',
'Morningside Heights', 'Morris Heights', 'Morris Park',
'Morrisania', 'Mott Haven', 'Mount Eden', 'Mount Hope',
'Murray Hill', 'Navy Yard', 'Neponsit', 'New Brighton', 'New Dorp',
'New Dorp Beach', 'New Springville', 'NoHo', 'Nolita',
'North Riverdale', 'Norwood', 'Oakwood', 'Olinville', 'Ozone Park',
'Park Slope', 'Parkchester', 'Pelham Bay', 'Pelham Gardens',
'Port Morris', 'Port Richmond', "Prince's Bay", 'Prospect Heights',
'Prospect-Lefferts Gardens', 'Queens Village', 'Randall Manor',
'Red Hook', 'Rego Park', 'Richmond Hill', 'Richmondtown',
'Ridgewood', 'Riverdale', 'Rockaway Beach', 'Roosevelt Island',
'Rosebank', 'Rosedale', 'Rossville', 'Schuylerville', 'Sea Gate',
'Sheepshead Bay', 'Shore Acres', 'Silver Lake', 'SoHo',
'Soundview', 'South Beach', 'South Ozone Park', 'South Slope',
'Springfield Gardens', 'Spuyten Duyvil', 'St. Albans',
'St. George', 'Stapleton', 'Stuyvesant Town', 'Sunnyside',
'Sunset Park', 'Theater District', 'Throgs Neck', 'Todt Hill',
'Tompkinsville', 'Tottenville', 'Tremont', 'Tribeca',
'Two Bridges', 'Unionport', 'University Heights',
'Upper East Side', 'Upper West Side', 'Van Nest', 'Vinegar Hill',
'Wakefield', 'Washington Heights', 'West Brighton', 'West Farms',
'West Village', 'Westchester Square', 'Westerleigh', 'Whitestone',
'Williamsbridge', 'Williamsburg', 'Willowbrook', 'Windsor Terrace',
'Woodhaven', 'Woodlawn', 'Woodrow', 'Woodside', nan], dtype=object)
```

Se ven algunos problemas de datos como: "Bay Terrace, Staten Island" donde se podría interpretar que el barrio es "Bay Terrace" del distrito de Staten Island, sucede lo mismo con "Chelsea, Staten Island".

```
data[(data['neighbourhood'] == "Bay Terrace, Staten Island") | (data['neighbourhood'] == "C
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_boc
3121	24674571	Staten Island	Bay Terrace, Staten Island	40.55105	-74.13660	United States	
15141	30348901	Staten Island	Chelsea, Staten Island	40.59116	-74.18580	United States	
35374	46461701	Staten Island	Bay Terrace, Staten Island	40.55182	-74.14439	United States	
64075	42683417	Staten Island	Bay Terrace, Staten Island	40.55105	-74.13660	United States	

Se encuentran 4 registros que presentan esa novedad, como son casos puntuales podrían corregirse puntualmente.

Ahora se revisa, si los datos nulos de barrio son diferentes a los vistos con nulos en el distrito:

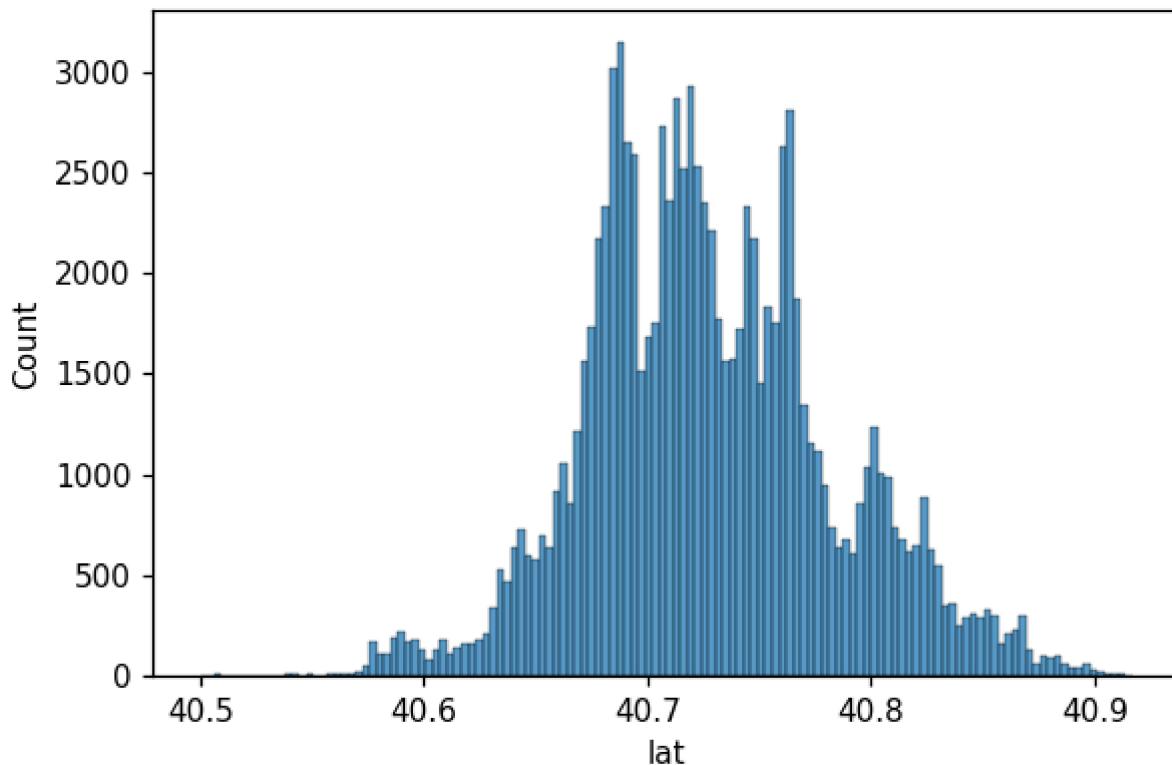
```
data[data['neighbourhood'].isnull() & ~data['neighbourhood group'].isnull() ]
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_book
1481	1318909	Manhattan		NaN	40.79816	-73.96190	United States
9839	1345971	Manhattan		NaN	40.75348	-73.97065	United States
16964	1377452	Brooklyn		NaN	40.73783	-73.95259	United States
27804	1356465	Brooklyn		NaN	40.68016	-73.94878	United States
44256	1402306	Manhattan		NaN	40.76217	-73.98411	United States
53213	1339806	Manhattan		NaN	40.72709	-73.98274	United

Luego los datos nulos de barrio estarían dentro dos 591 registros completamente sin datos identificados anteriormente. Y quedarían 16 sin datos de barrio.

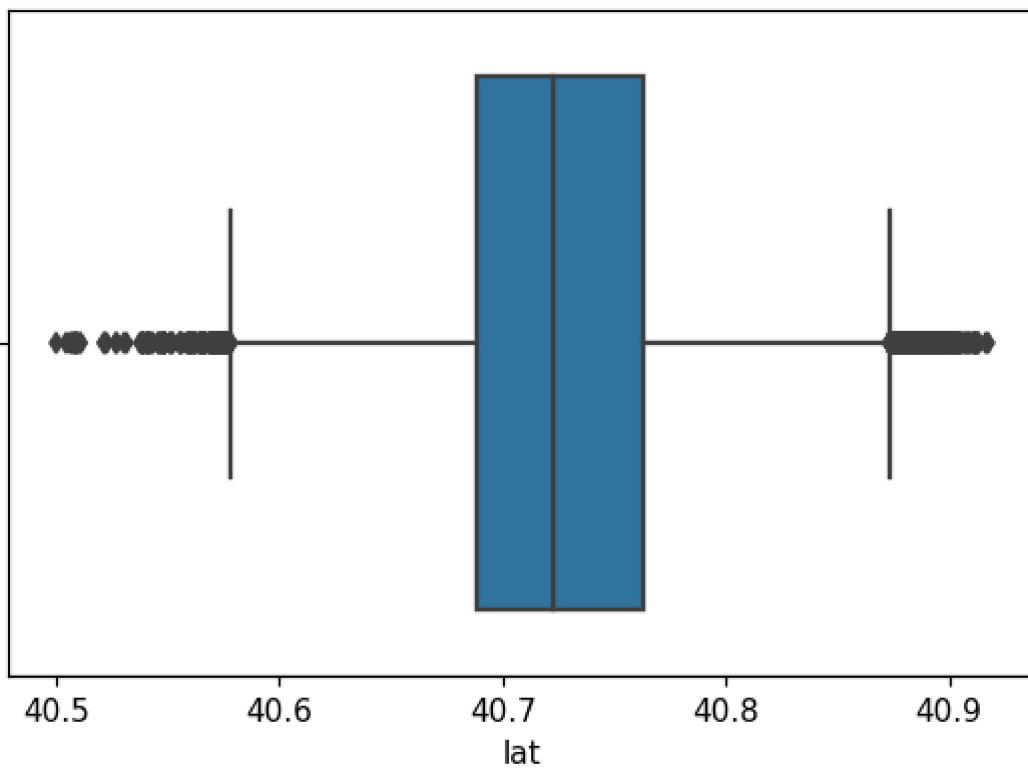
```
sns.histplot(data=data, x="lat")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9336ad8090>
```



```
sns.boxplot(x=data["lat"])
```

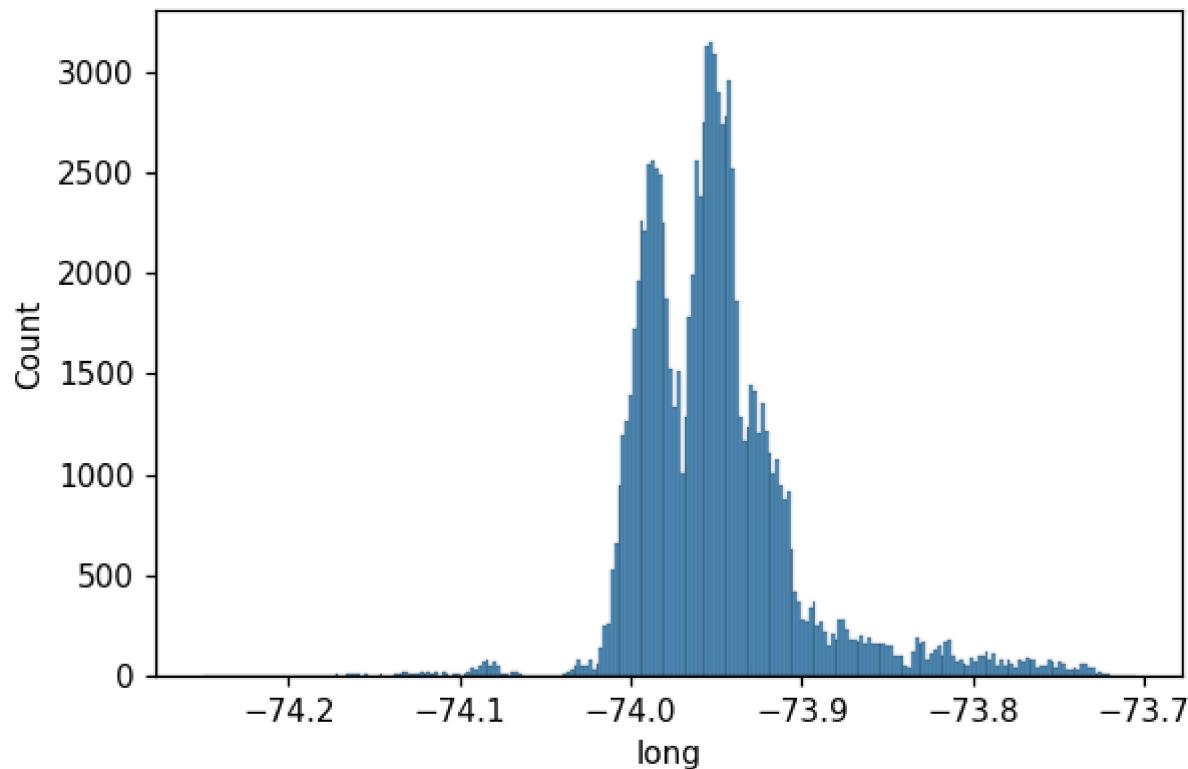
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9334da4090>
```



Del histograma y boxplot se puede ver que hay un amplio número de datos de latitud que se concentran en el medio, con algunos valores que se escapan del Q1 y Q3.

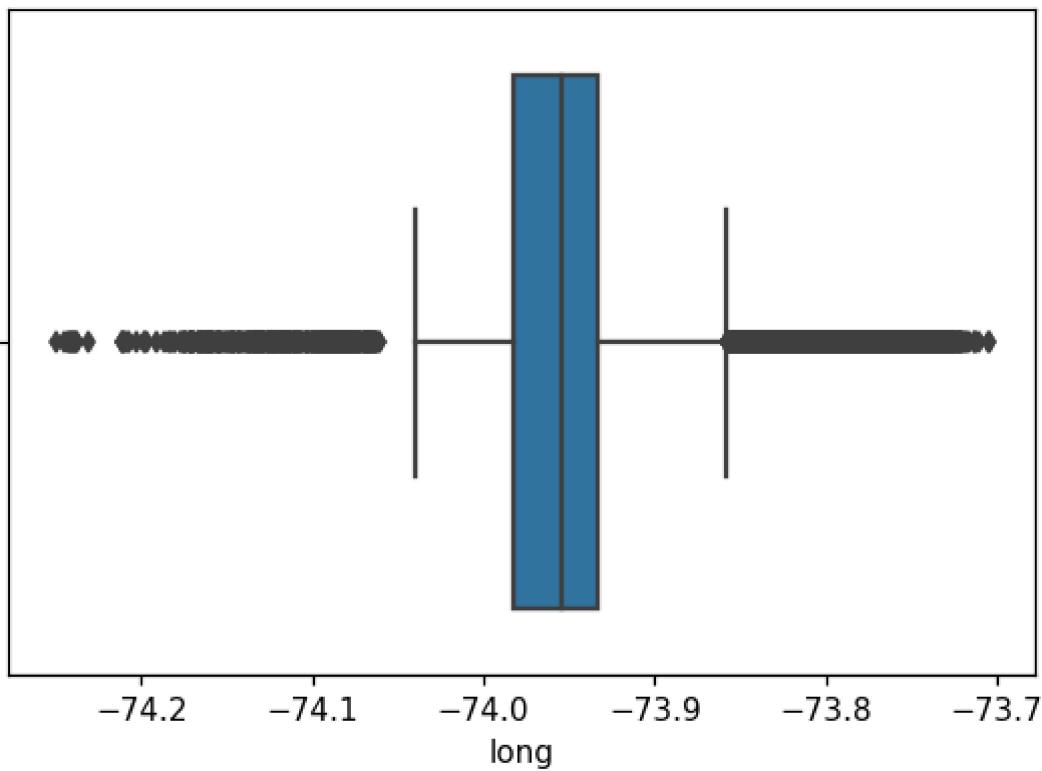
```
sns.histplot(data=data, x="long")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9336f6ffd0>
```



```
sns.boxplot(x=data["long"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9336fa9790>
```



```
data.loc[:, {'lat', 'long'}].describe()
```

	long	lat	geometry
count	101484.000000	101484.000000	
mean	-73.94981	40.728070	
std	0.04936	0.055829	
min	-74.24984	40.499790	
25%	-73.98261	40.688750	
50%	-73.95456	40.722270	
75%	-73.93260	40.762730	
max	-73.70522	40.916970	

```
#revisando los valores vacíos de lat y long, que no esten en los antes evaluados:
```

```
data[ data['lat'].isnull() & data['long'].isnull() & ~data['neighbourhood'].isnull() ]
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_bookable	c
15698	1545904	Manhattan	Upper West Side	NaN	NaN	United States	True	
20771	1512766	Manhattan	Flatiron District	NaN	NaN	United States	False	
29646	1490122	Brooklyn	Greenpoint	NaN	NaN	United States	True	
31995	1431578	Brooklyn	Crown Heights	NaN	NaN	United States	False	
47763	1466925	Queens	Elmhurst	NaN	NaN	United States	True	
73732	1434892	Brooklyn	Greenpoint	NaN	NaN	United States	False	
90504	1442624	Manhattan	East Village	NaN	NaN	United States	False	
100878	1450908	Manhattan	West Village	NaN	NaN	United States	True	

```
data[ data['lat'].isnull() & data['long'].isnull() & ~data['neighbourhood'].isnull() ].count
```

id	8
neighbourhood group	8
neighbourhood	8
lat	0
long	0
country	8
instant_bookable	8
cancellation_policy	8
room type	8
construction year	7
price	8
service fee	8
minimum nights	8
availability 365	8
number of reviews	8
review rate number	8
dtype: int64	

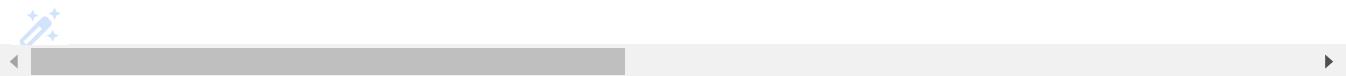
Al revisar los datos nulos tanto para long y lat, que contengan datos de las otras variables, solo se encuentran 8 casos. Es decir que los otros datos nulos estarían dentro de los 591 registros que no tienen valores en las diferentes columnas. Aun sin modelar, se podría pensar como hipótesis que las coordenadas geográficas tendrían más efecto que el barrio o el distrito, por lo que para estos

casos dado que lat y long no tienen una varianza muy alta pueden reemplazarse los valores vacíos

```
data['lat'] = data['lat'].fillna(data['lat'].median())
```

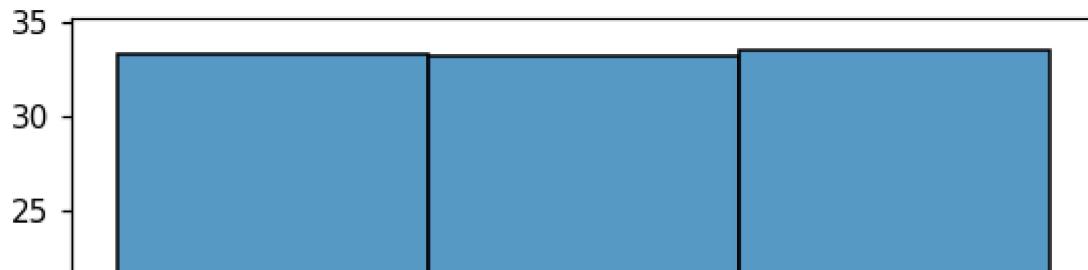
```
data[ data['long'].isnull() & ~data['neighbourhood'].isnull() ]
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_bookable
15698	1545904	Manhattan	Upper West Side	40.72227	NaN	United States	True
20771	1512766	Manhattan	Flatiron District	40.72227	NaN	United States	False
29646	1490122	Brooklyn	Greenpoint	40.72227	NaN	United States	True
31995	1431578	Brooklyn	Crown Heights	40.72227	NaN	United States	False
47763	1466925	Queens	Elmhurst	40.72227	NaN	United States	True
73732	1434892	Brooklyn	Greenpoint	40.72227	NaN	United States	False
90504	1442624	Manhattan	East Village	40.72227	NaN	United States	False
100878	1450908	Manhattan	West Village	40.72227	NaN	United States	True



```
sns.histplot(data=data, x="cancellation_policy", stat="percent")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93369d6850>
```



Respecto a la política de cancelación, se observa que los tres valores están bien distribuidos la data.



```
#revisando los valores vacíos de room type, que no estén en los anteriores evaluados:  
data[ data['room type'].isnull() & ~data['neighbourhood'].isnull() ]
```

id	neighbourhood group	neighbourhood	lat	long	country	instant_bookable	cancellation_
591	Brasília	Brasília	-15.7800	-47.9400	BRAZIL	True	0

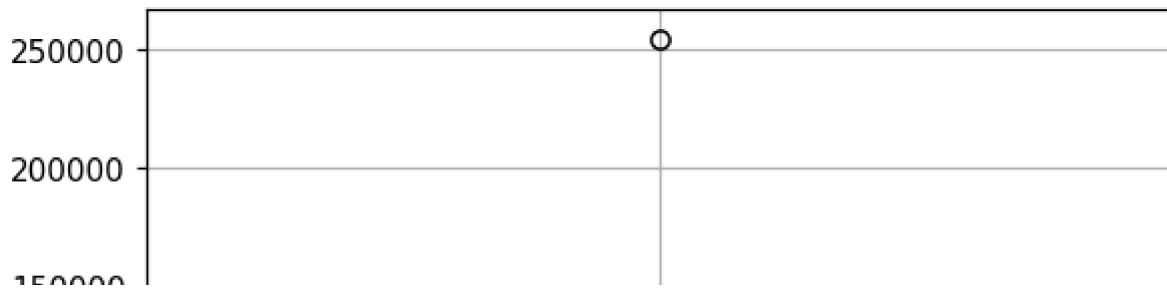
No se evidencian datos vacíos para "room type" diferente a los 591 registros reportados anteriormente.

Ahora se visualiza un boxplot, de la columna precio:

```
dfprice = data.copy()  
  
dfprice['price'] = dfprice['price'].str.replace(r'$','')  
dfprice['price'] = dfprice['price'].str.replace(',','.')
```

```
dfprice['price'] = dfprice['price'].str.replace(r'\t','')  
dfprice['price'] = dfprice['price'].astype(float)  
boxplot = dfprice.boxplot(column=['price'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: The defa
    """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: The defa
    This is separate from the ipykernel package so we can avoid doing imports until
```



```
dfprice['service fee'] = dfprice['service fee'].str.replace(r'$','')
dfprice['service fee'] = dfprice['service fee'].str.replace(r',','.')
dfprice['service fee'] = dfprice['service fee'].str.replace(r'\t','')
dfprice['service fee'] = dfprice['service fee'].astype(float)
boxplot = dfprice.boxplot(column=['service fee'])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: The defa
    """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: The defa
    This is separate from the ipykernel package so we can avoid doing imports until
```



Se observa que si se manejara *price* y *serve fee* con números hay valores outliers para ambos, viendo un poco en detalle:

```
dfprice.loc[:, ['price', 'service fee']].describe()
```

	price	service fee	edit
count	101245.000000	101220.000000	
mean	435.631734	126.190130	
std	858.065480	388.772016	
min	-611.000000	-193.000000	
25%	137.000000	68.000000	
50%	427.000000	125.000000	
75%	710.000000	182.000000	
max	254000.000000	122000.000000	

```
dfprice.loc[:, ['price', 'service fee']].quantile(.95)
```

```
price      941.8
service fee    229.0
Name: 0.95, dtype: float64
```

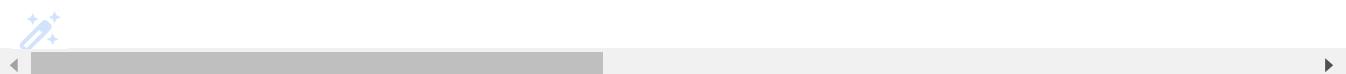
Ambos presentan valores atípicos como es el caso de los negativos, y de acuerdo al 95 percentil de los datos vemos valores extremos de acuerdo al valor máximo.

```
data[ data['number of reviews'].isnull() & ~data['review rate number'].isnull() ]
```

		id	neighbourhood group	neighbourhood	lat	long	country	instant_bc
18		26674449	Queens	Woodside	40.74434	-73.90347	United States	
646		9478036	Manhattan	Upper West Side	40.77992	-73.98046	United States	
934		46073434	Manhattan	Hell's Kitchen	40.76479	-73.98842	United States	
---	---	---	---	---	---	---	United	
data[~data['number of reviews'].isnull() & data['review rate number'].isnull()]								

		id	neighbourhood group	neighbourhood	lat	long	country	instant_
321		1104064	Brooklyn	Bedford-Stuyvesant	40.679920	-73.947500	United States	
999		29622627	Queens	Jamaica	40.682037	-73.796814	United States	
1104		1150457	Brooklyn	Bedford-Stuyvesant	40.683170	-73.947010	United States	
1670		26902549	Brooklyn	Flatbush	40.649460	-73.960410	United States	
1711		41652273	Manhattan	Harlem	40.821650	-73.955930	United States	
...	United	...
101039		1065955	Brooklyn	Williamsburg	40.719420	-73.957480	United States	
101130		33808508	Bronx	Wakefield	40.894020	-73.844370	United States	
101466		26937344	Brooklyn	Prospect Heights	40.679770	-73.968820	United States	
101543		51266159	Queens	Jamaica	40.679490	-73.798410	United States	
102081		1476314	Manhattan	Upper West Side	40.780120	-73.984390	United States	

317 rows × 16 columns



```
dfprice['review rate number'].describe()
```

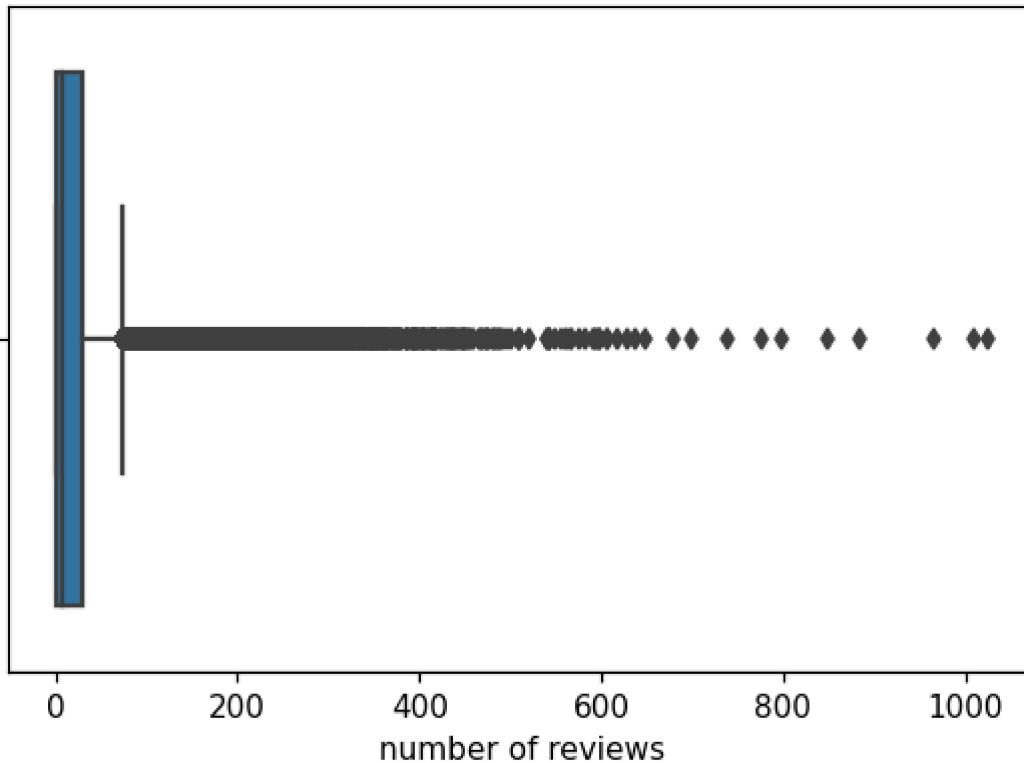
```
count    101174.000000
mean      3.279222
std       1.284902
min      1.000000
25%     2.000000
50%     3.000000
75%     4.000000
max      5.000000
Name: review rate number, dtype: float64
```

```
dfprice['review rate number'].median()
```

```
3.0
```

```
sns.boxplot(x=dfprice["number of reviews"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f933a5266d0>
```



```
dfprice["number of reviews"].describe()
```

```
count    101313.000000
mean      27.420479
std       49.473778
min      0.000000
25%     1.000000
50%     7.000000
75%    30.000000
```

```
max      1024.000000
Name: number of reviews, dtype: float64
```

▼ d. Preparación de los datos

De acuerdo al entendimiento de los datos realizados se crea la función la función **preprocess_quality** que enmarca las diferentes acciones a realizar al dataframe de datos.

- 1) como se encontraron 536 de 102083 registros como duplicados, serán borrados éstos registros
- 2) Corregir el nombre de *neighbourhood group*, con la función `fix_neighbourhood_group` usando el método levenshtein. Para el caso de los valores faltantes, aplicar buscando por las coordenadas a qué distrito pertenece o si tiene el nombre del barrio, asignar el distrito que le corresponde. Si hay registros que no cumplan alguna de las anteriores el registro se borra.
- 3) En el caso de *neighbourhood* como la mayoría de registros nulos se borra en el punto 1, los pocos que quedan se les asigna como barrio la moda para esta columna (Bedford-Stuyvesant).
- 4) Como se evidenció que la mayoría de *nan* de *lat* y *long* estan en los registros mencionados en el punto 1, para los que si tienen datos en otras columnasse propone imputar el valor con la mediana de cada uno, dado que son variables con poca desviación y que no son más de 20 registros.
- 5) Para el caso de Country, todos los países quedaran como "United States".
- 6) En el caso de *instant_bookable* que es de tipo booleano False tiene el 50.1% mientras que True el 49.8% (cuando se quitan los registros indicados en 1), dado que los faltantes son el 0.1% se decide colocar los valores faltantes el false.
- 7) La columna *cancellation_policy* tiene un 0.1% de valores faltantes, es variable categórica de 3 posibles valores para los cuales se tienen: moderate 33.5%, strict 33.2% y flexible 33.2%. Se decide usar la moda para reemplazar los valores faltantes.
- 8) La columna *construction year* presenta valores negativos, solo 3, y pareciera fue un error de digitación: 1020 y 1022, se actualiza los valores a 2020 y 2022. (sumando 1000). Para los valores faltantes se usará la mediana, ya que la desviación esde 7.
- 9) Remover el signo \$ de las columnas *price* y *service fee*, y que sean valores numéricos, reemplazar los numeros negativos y valores faltantes con la mediana. En ambos casos se identificaron outliers muy superiores al percentil 95, por lo que se ajustan los valores superiores a este un valor ligeramente superior.
- 10) Para el caso de *minimum nights* que tiene valores negativos, se observa que la mediana es 3 y el Q3 es 5, por lo que se dejará el valor de la mediana.
- 11) En el caso de *availability 365* tiene valroes superiores a 365 que se espera que sea el máximo, los valores superiores a 365 se igualan a éste valor. Por otra parte para los valores negativos y ausentes se usará la media para imputarlos.

- 12) En la columna *number of reviews*, dado que es la variable objetivo se prefiere borrar en vez de imputar por algun valor que realmente pueda no reflejar la popularidad del sitio.
- 13) En el caso *review rate number* se sugiere reemplazar los valores faltantes con la mediana, que de acuerdo a lo visto es un valor de 3.

```
def fix_neighbourhood_group(neighbourhood_group):
    if pylev.levenshtein('Manhattan', neighbourhood_group) <= 2:
        return 'Manhattan'
    elif pylev.levenshtein('Brooklyn', neighbourhood_group) <= 2:
        return 'Brooklyn'
    elif pylev.levenshtein('Queens', neighbourhood_group) <= 2:
        return 'Queens'
    elif pylev.levenshtein('Bronx', neighbourhood_group) <= 2:
        return 'Bronx'
    elif pylev.levenshtein('Staten Island', neighbourhood_group) <= 2:
        return 'Staten Island'

def preprocess(df):
    #Eliminar los datos duplicados:
    df = df.drop_duplicates()

#Dado los 591 que evidenciaron que solo tenia valores para el índice, se borran aquellos q
df= df.drop(df[df['neighbourhood group'].isnull() & df['lat'].isnull() & df['long'].isnull()]

#Asignar distrito a partir de los datos de Geolocalización en la columna neighbourhood gro
df.loc[ (df['lat'] >= 40.70234) & (df['lat'] <= 40.87821) & (df['long'] >= -74.01851) & (c
df.loc[ (df['lat'] >= 40.56546) & (df['lat'] <= 40.79721) & (df['long'] >= -73.95953) & (c
df.loc[ (df['lat'] >= 40.80011) & (df['lat'] <= 40.91697) & (df['long'] >= -73.93296) & (c
df.loc[ (df['lat'] >= 40.49979) & (df['lat'] <= 40.64816) & (df['long'] >= -74.24984) & (c

#Generar un diccionario con clave barrio y de valor distrito, como otra forma para colocar
distritos = df.loc[~df['neighbourhood group'].isnull(), ['neighbourhood', 'neighbourhood gr
d = dict([(i,a) for i,a in zip(distritos['neighbourhood'], distritos['neighbourhood group']
df.loc[ (df['neighbourhood group'].isnull() ) , ['neighbourhood group']] = df["neighbourho

#Eliminar los registros filas con datos faltantes de distrito (si no quedo corregido en alg
df = df.dropna(subset=['neighbourhood group'])

#Corregir el nombre de neighbourhood group con Levenshtein método.
df['neighbourhood group'] = df['neighbourhood group'].apply(fix_neighbourhood_group)

#Corrección de los dos barrios "Bay Terrace, Staten Island" "Chelsea, Staten Island"
df['neighbourhood'] = df['neighbourhood'].apply(lambda x: "Bay Terrace" if (x == "Bay Terra
df['neighbourhood'] = df['neighbourhood'].apply(lambda x: "Chelsea" if (x == "Chelsea, Stat

#Imputar los valores pendientes de barrio con la moda:
df['neighbourhood'] = df['neighbourhood'].fillna(pd.Series(df["neighbourhood"].values.flatt
```

```
#Asignar los valores de la media en lat y long para los valores vacíos:  
df['lat'] = df['lat'].fillna(df['lat'].median())  
df['long'] = df['long'].fillna(df['long'].median())  
  
#Ajustar la columna country:  
df['country'] = "United States"  
  
#Asignar False en instant_bookable para los valores faltantes  
df['instant_bookable'] = df['instant_bookable'].fillna(pd.Series(df["instant_bookable"].va)  
  
#Asignar False en instant_bookable para los valores faltantes  
df['cancellation_policy'] = df['cancellation_policy'].fillna(pd.Series(df["cancellation_po]  
  
#construction year: tiene valores de los años 1022 y 1021  
df['construction year'] = df['construction year'].apply(lambda x: x+1000 if (x<=1022) else  
df['construction year'] = df['construction year'].fillna(df['construction year'].median())  
  
#remover el signo $ de las columnas price y service fee:  
df['price'] = df['price'].str.replace(r'$','')  
df['price'] = df['price'].str.replace(',','.')  
df['price'] = df['price'].str.replace(r'\t','')  
df['price'] = df['price'].astype(float)  
  
df['service fee'] = df['service fee'].str.replace(r'$','')  
df['service fee'] = df['service fee'].str.replace(',','.')  
df['service fee'] = df['service fee'].str.replace(r'\t','')  
df['service fee'] = df['service fee'].astype(float)  
  
#Reemplazar los valores faltantes de: price, service fee con el valor medio  
df['price'] = df['price'].fillna(df['price'].median())  
df['service fee'] = df['service fee'].fillna(df['service fee'].median())  
  
#valores negativos de price y service fee  
df['price'] = df['price'].apply(lambda x: df['price'].median() if (x<0) else x)  
df['service fee'] = df['service fee'].apply(lambda x: df['service fee'].median() if (x<0) e  
  
#Modificar valores outliers de price: 941 es el 95 percentil, service fee 229.0 es el p95  
df['price'] = df['price'].apply(lambda x: 950 if (x>950) else x)  
df['service fee'] = df['service fee'].apply(lambda x: 250 if (x>250) else x)  
  
#valores negativos de minimum nights usa 3 que es el de la mediana, y lo mismo para los valo  
df['minimum nights'] = df['minimum nights'].apply(lambda x: df['minimum nights'].median() i  
df['minimum nights'] = df['minimum nights'].fillna(df['minimum nights'].median())  
  
#availability 365 valores superiores a 365 se deja con el valor máximo: 365  
df['availability 365'] = df['availability 365'].apply(lambda x: 365 if (x>365) else x)  
#availability 365 valores negativos y faltantes se reemplazan con la media:  
df['availability 365'] = df['availability 365'].fillna(df['availability 365'].median())  
df['availability 365'] = df['availability 365'].apply(lambda x: df['availability 365'].medi
```

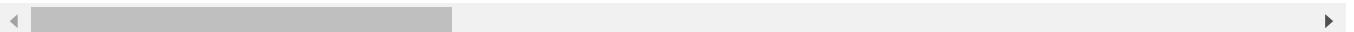
```
#Borrar los registros que no tienen valores en la columna number of reviews
df = df.dropna(subset=['number of reviews'])

#pasar review rate number a número
df['review rate number'] = df['review rate number'].astype(float)
df['review rate number'] = df['review rate number'].fillna(df['review rate number'].median()

df.reset_index(drop=True, inplace=True)
return df
```

```
dataH = preprocess(data)
dataH.shape
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:63: FutureWarning: The def
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:65: FutureWarning: The def
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:68: FutureWarning: The def
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:70: FutureWarning: The def
(100783, 16)
```



Luego del procesamiento de calidad de datos, se pasa de 102083 a 100783 registros, esto es solo se eliminaron 1.27% de los datos.

```
ProfileReport(dataH)
```

Summarize dataset:	110/110 [00:22<00:00, 4.72it/s,
100%	Completed]
Generate report structure: 100%	1/1 [00:07<00:00, 7.99s/it]
Render HTML: 100%	1/1 [00:03<00:00, 3.23s/it]

Overview

Dataset statistics

Number of variables	16
Number of observations	100783

Se muestra como quedaron los datos despues de la limpieza de datos

missing cells (%) 0.0 / 0

```
dataH.head(20)
```

1	35079903	Manhattan	Midtown	40.74623	-73.98499	United States
2	50681273	Manhattan	Upper West Side	40.78859	-73.97568	United States
3	13039267	Manhattan	Financial District	40.70817	-74.00511	United States
4	8998640	Manhattan	Lower East Side	40.72130	-73.98900	United States
5	1629301	Manhattan	Hell's Kitchen	40.75636	-73.99390	United States
6	2656579	Manhattan	Greenwich Village	40.73236	-73.99920	United States
7	2126371	Manhattan	Morningside Heights	40.80776	-73.96540	United States
8	37291864	Queens	Crown Heights	40.67497	-73.94086	United States
9	11288476	Manhattan	Upper West Side	40.79371	-73.96708	United States
10	14167063	Queens	Astoria	40.76473	-73.91183	United States
11	36508151	Brooklyn	Clinton Hill	40.68971	-73.96544	United States
12	20567668	Manhattan	Financial District	40.70619	-74.00640	United States
13	42009611	Manhattan	Morningside Heights	40.80511	-73.95824	United States
14	13900855	Manhattan	Upper West Side	40.77356	-73.98007	United States
15	7500010	Manhattan	Midtown	40.74623	-73.98499	United States

Dado que vamos a usar regresión logística, hay que pasar los datos categóricos a numéricos:

```
16 6104588    Queens      Bushwick 40.69682 -73.91458 2019
```

```
def encoderF(df2):
```

```
    # Eliminar columna País e id:
```

```
    df2 = df2.drop(['country', 'id'], axis=1)
```

```
# instant_bookable to num
```

```
df2['instant_bookable'] = df2['instant_bookable'].replace({
```

```
    False: 0,
```

```
    True: 1
```

```
)
```

```
#encoder para neighbourhood group
```

```
df2 = pd.get_dummies(df2,columns=['neighbourhood group'],drop_first = True)

#Dado que el barrio tiene muchas categorias, y que se tienen otros valores de geolocalizaci
df2 = df2.drop(['neighbourhood'], axis=1)

#encoder para cancellation_policy
df2 = pd.get_dummies(df2,columns=['cancellation_policy'],drop_first = True)

#encoder para room type
df2 = pd.get_dummies(df2,columns=['room type'],drop_first = True)

df2.reset_index(drop=True, inplace=True)
return df2
```

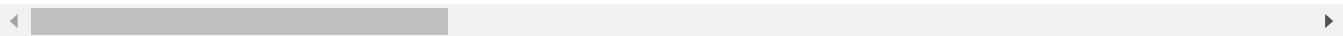
```
dataH=enconderF(dataH)
```

```
dataH.shape
```

```
(100783, 19)
```

```
dataH.head(10)
```

	lat	long	instant_bookable	construction year	price	service fee	minimum nights	availab
0	40.76212	-73.98820	1	2011.0	851.0	170.0	4.0	
1	40.74623	-73.98499	1	2021.0	466.0	93.0	1.0	
2	40.78859	-73.97568	0	2004.0	874.0	175.0	14.0	
3	40.70817	-74.00511	0	2012.0	813.0	163.0	1.0	
4	40.72130	-73.98900	1	2007.0	326.0	65.0	4.0	
5	40.75636	-73.99390	1	2022.0	786.0	157.0	30.0	
6	40.73236	-73.99920	0	2008.0	863.0	173.0	5.0	
7	40.80776	-73.96540	1	2007.0	749.0	150.0	2.0	
8	40.67497	-73.94086	1	2004.0	397.0	79.0	2.0	
9	40.79371	-73.96708	1	2009.0	462.0	92.0	30.0	



▼ Modelamiento

Dividimos los datos en: datos de entrenamiento (train) y datos de evaluación (test) con una relación 80% - 20% respectivamente.

```
train, test = train_test_split(dataH, test_size=0.4, random_state=33)
train.head()
```

	lat	long	instant_bookable	construction_year	price	service_fee	minimum_nights	av
94486	40.76291	-73.99327		1	2016.0	151.00	30.0	30.0
82662	40.71864	-73.99388		0	2010.0	73.00	15.0	15.0
16601	40.83193	-73.93640		0	2003.0	820.00	164.0	2.0
7181	40.76281	-73.91554		0	2013.0	1.17	234.0	1.0
36366	40.80747	-73.95076		1	2008.0	651.00	130.0	5.0



▶

Para trabajar el problema de regresión, se deben pasar las columnas de categoricas a numericas, como todos los predios estan en Estados Unidos vamos a dejar de disponer de esta columna

```
#separando la variable objetivo:
x_train = train.drop('number of reviews',axis=1)
y_train = train['number of reviews']

#separando la variable objetivo:
x_test = test.drop('number of reviews',axis=1)
y_test = test['number of reviews']
```

▼ a) Primer modelo de regresion lineal

```
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
lin_reg.intercept_, lin_reg.coef_

(6381.78548264043,
array([-3.79270302e+00, 8.50127725e+01, 5.63482605e-01, 4.00618853e-02,
       1.28060805e-04, 3.03783694e-03, -6.55514844e-02, 3.59295574e-02,
      -8.06753948e-01, 8.02146554e+00, 3.14008370e+00, 3.73496295e+00,
```

```
2.27710536e+01, 2.73688337e-01, -4.76917146e-01, 4.15589460e+01,
-6.95486322e-01, -9.20428668e+00]))
```

```
y_pred_train = lin_reg.predict(x_train)
```

```
y_pred_test = lin_reg.predict(x_test)
```

#Función para ver los resultados de las métricas de un modelo entrenamiento y validación

```
def metricsPrint (titulo, X_train, y_train, y_pred_train, X_val, y_val, y_pred_val):
    n,p = X_train.shape

    print('----- Regresión ', titulo, ' con data entrenamiento-----')
    print("Residual sum of squares (MSE): %.2f" % mean_squared_error(y_train,y_pred_train))
    print("R2-score: %.5f" % r2_score(y_train, y_pred_train) )
    print("Adj R2-score: %.5f" % ( 1-(1-r2_score(y_train, y_pred_train))*(n-1)/(n-p-1)) )
```

```
n,p = X_val.shape
```

```
print('----- Regresión ', titulo, ' con data test -----')
print("Residual sum of squares (MSE): %.2f" % mean_squared_error(y_val,y_pred_val))
print("R2-score: %.5f" % r2_score(y_val, y_pred_val) )
print("Adj R2-score: %.5f" % ( 1-(1-r2_score(y_val, y_pred_val))*(n-1)/(n-p-1)) )
```

```
metricsPrint ('Regresion Lineal', x_train, y_train, y_pred_train, x_test, y_test, y_pred_test)
```

```
----- Regresión Regresion Lineal con data entrenamiento-----
Residual sum of squares (MSE): 2335.63
R2-score: 0.02130
Adj R2-score: 0.02101
----- Regresión Regresion Lineal con data test -----
Residual sum of squares (MSE): 2500.79
R2-score: 0.02100
Adj R2-score: 0.02057
```

Con un primer modelo básico de regresión lineal, se observa que el error aumenta un poco con los datos de test, lo que hace que la métrica del R2-score disminuya ligeramente con los datos de test, y la métrica del R2-score no es un gran valor de predicción.

▼ b) Modelo regresión lineal con normalización de datos

```
scaler = MinMaxScaler()
scaler.fit(x_train)
```

```
columns = x_train.columns
X_train_norm = scaler.fit_transform(x_train)
X_train_norm = pd.DataFrame(X_train_norm, columns=columns)
X_train_norm
```

	lat	long	instant_bookable	construction year	price	service fee	minimum nights
0	0.630892	0.465800		1.0	0.684211	0.158061	0.086957
1	0.524745	0.464669		0.0	0.368421	0.075869	0.021739
2	0.796384	0.571275		0.0	0.000000	0.863014	0.669565
3	0.630653	0.609963		0.0	0.526316	0.000179	0.973913
4	0.737736	0.544642		1.0	0.263158	0.684932	0.521739
...
60464	0.541073	0.486479		1.0	0.000000	0.000021	0.847826
60465	0.435141	0.590990		1.0	0.947368	1.000000	0.800000
60466	0.535942	0.482418		0.0	0.578947	0.959958	0.747826
60467	0.626121	0.832561		1.0	0.842105	0.954689	0.743478
60468	0.805232	0.553822		1.0	0.368421	0.527924	0.391304

60469 rows × 18 columns



◀ ▶

```
lin_regN = LinearRegression()
lin_regN.fit(X_train_norm, y_train)
```

LinearRegression()

```
y_pred_trainN = lin_regN.predict(X_train_norm)
```

```
X_test_norm = scaler.fit_transform(x_test)
X_test_norm = pd.DataFrame(X_test_norm, columns=columns)
X_test_norm
```

```
y_pred_testN = lin_regN.predict(X_test_norm)
```

```
metricsPrint ('Regresion Lineal', X_train_norm, y_train, y_pred_trainN, X_test_norm, y_test,
----- Regresión Lineal con data entrenamiento-----
Residual sum of squares (MSE): 2335.63
```

```
R2-score: 0.02130
Adj R2-score: 0.02101
----- Regresión Regresion Lineal con data test -----
Residual sum of squares (MSE): 2506.07
R2-score: 0.01894
Adj R2-score: 0.01850
```

Con este modelo aumenta el error en test, pero disminuye un poco el sobreajuste, respecto al primer modelo

▼ c) Modelo Regresión polinomial, p=2

Dada la cantidad de datos, y que la ejecución Polinomial requiere de más ram permitida por el google Colab, se aplica reducción de la dimensionalidad PCA a 10, para poder ejecutar la regresión polinomial con grado 2.

```
pca = PCA(n_components=10, random_state=32)
pca.fit(x_train)
x_train_PCA = pca.transform(x_train)
x_train_PCA

array([[-2.85295415e+02,  2.01126626e+02, -8.42401465e+01, ...,
       1.57173740e-03, -5.24473155e-01, -1.85146285e-01],
      [-3.63293137e+02, -8.91949212e+01, -9.64261940e+01, ...,
       -1.27163673e-02,  4.75579331e-01, -2.66707671e-01],
      [ 3.88372441e+02,  1.34855665e+02,  2.45919066e+01, ...,
       -7.20890346e-01,  5.22604244e-01,  7.12543180e-01],
      ...,
      [ 4.81227022e+02, -6.97629928e+00,  3.90955203e+01, ...,
       6.90926191e-01,  5.56434232e-01,  6.96121440e-01],
      [ 4.75956533e+02,  1.29982623e+02,  3.83207984e+01, ...,
       2.15710190e-02, -4.81254141e-01,  4.02440898e-01],
      [ 6.80909913e+01,  1.99291000e+02, -2.74611115e+01, ...,
       -7.01109239e-01, -4.78185722e-01,  7.65627633e-01]])
```

```
poly_features = PolynomialFeatures(degree=2, include_bias=False)
x_poly_train = poly_features.fit_transform(x_train_PCA)
dataset = pd.DataFrame(x_poly_train)
x_poly_train.shape

(60469, 65)
```

```
lin_regP = LinearRegression()
lin_regP.fit(x_poly_train, y_train)

LinearRegression()
```

```

y_pred_train = lin_regP.predict(x_poly_train)

x_test_PCA = pca.transform(x_test)
x_poly_test = poly_features.fit_transform(x_test_PCA)
y_pred_test = lin_regP.predict(x_poly_test)

metricsPrint ('Regresion Polinomial P=2', x_poly_train, y_train, y_pred_train, x_poly_test, y_pred_test)

----- Regresión Regresion Polinomial P=2 con data entrenamiento-----
Residual sum of squares (MSE): 2265.93
R2-score: 0.05051
Adj R2-score: 0.04948
----- Regresión Regresion Polinomial P=2 con data test -----
Residual sum of squares (MSE): 2435.28
R2-score: 0.04665
Adj R2-score: 0.04511

```

▼ d) Modelo Regresión polinomial, p=3

Ahora se hace regresión polinomial con grado 3, y la reducción de dimensionalidad la dejaremos en 3 (buscando trabajar con una dimensión mas pequeña y evitar el error de ram)

```

pca = PCA(n_components=3, random_state=32)
pca.fit(x_train)
x_train_PCA = pca.transform(x_train)
x_train_PCA

array([[-285.29541494,  201.12662557,  -84.24014646],
       [-363.29313659,  -89.19492119,  -96.42619396],
       [ 388.3724407 ,  134.85566543,   24.59190656],
       ...,
       [ 481.22702202,    -6.97629928,   39.09552031],
       [ 475.95653274,  129.98262346,   38.32079844],
       [ 68.09099129,  199.29100003,  -27.46111153]])

```

```

poly_features = PolynomialFeatures(degree=3, include_bias=False)
x_poly_train = poly_features.fit_transform(x_train_PCA)
dataset = pd.DataFrame(x_poly_train)
x_poly_train.shape

(60469, 19)

```

```

lin_regP = LinearRegression()
lin_regP.fit(x_poly_train, y_train)

```

```
LinearRegression()
```

```
y_pred_train = lin_regP.predict(x_poly_train)

x_test_PCA = pca.transform(x_test)
x_poly_test = poly_features.fit_transform(x_test_PCA)
y_pred_test = lin_regP.predict(x_poly_test)
```

```
metricsPrint ('Regresion Polinomial P=3', x_poly_train, y_train, y_pred_train, x_poly_test, y_pred_test)

----- Regresión Regresion Polinomial P=3 con data entrenamiento-----
Residual sum of squares (MSE): 2301.42
R2-score: 0.03563
Adj R2-score: 0.03533
----- Regresión Regresion Polinomial P=3 con data test -----
Residual sum of squares (MSE): 2467.82
R2-score: 0.03391
Adj R2-score: 0.03346
```

▼ e) Modelo Regresión polinomial y regularización lasso

Dado que el polinomio grado 2 funcionó un poco mejor que grado 3, ahora le vamos a aplicar una regularización lasso para que si restringiendo alguna característica se tiene un mejor valor

```
pca = PCA(n_components=10, random_state=32)
pca.fit(x_train)
x_train_PCA = pca.transform(x_train)
x_train_PCA

array([[-2.85295415e+02,  2.01126626e+02, -8.42401465e+01, ...,
       1.57173740e-03, -5.24473155e-01, -1.85146285e-01],
      [-3.63293137e+02, -8.91949212e+01, -9.64261940e+01, ...,
       -1.27163673e-02,  4.75579331e-01, -2.66707671e-01],
      [ 3.88372441e+02,  1.34855665e+02,  2.45919066e+01, ...,
       -7.20890346e-01,  5.22604244e-01,  7.12543180e-01],
      ...,
      [ 4.81227022e+02, -6.97629928e+00,  3.90955203e+01, ...,
       6.90926191e-01,  5.56434232e-01,  6.96121440e-01],
      [ 4.75956533e+02,  1.29982623e+02,  3.83207984e+01, ...,
       2.15710190e-02, -4.81254141e-01,  4.02440898e-01],
      [ 6.80909913e+01,  1.99291000e+02, -2.74611115e+01, ...,
       -7.01109239e-01, -4.78185722e-01,  7.65627633e-01]])
```

```
poly_features = PolynomialFeatures(degree=2, include_bias=False)
x_poly_train = poly_features.fit_transform(x_train_PCA)
```

```
dataset = pd.DataFrame(x_poly_train)
x_poly_train.shape

(60469, 65)

x_test_PCA = pca.transform(x_test)
x_poly_test = poly_features.fit_transform(x_test_PCA)
```

Se decide un alfa de 35 como hiperparámetro de la regularización Lasso

```
LassoModel = Lasso(alpha=35)
LassoModel.fit(x_poly_train, y_train)

Lasso(alpha=35)

y_pred_lass_train_1 = LassoModel.predict(x_poly_train)
y_pred_lass_train_1

array([20.89584041, 26.64615928, 36.77282934, ..., 37.2157436 ,
       40.20701751, 28.45800649])
```

```
y_pred_lass_test_1 = LassoModel.predict(x_poly_test)
```

```
metricsPrint ('Regresion Lasso alfa = 35', x_poly_train, y_train, y_pred_lass_train_1, x_poly_test)

----- Regresión Regresion Lasso alfa = 35 con data entrenamiento-----
Residual sum of squares (MSE): 2290.17
R2-score: 0.04035
Adj R2-score: 0.03932
----- Regresión Regresion Lasso alfa = 35 con data test -----
Residual sum of squares (MSE): 2455.68
R2-score: 0.03866
Adj R2-score: 0.03711
```

▼ Comparación de modelos:

```
columnas = ['a.Reglineal', 'b.Reg linealyMinMaxScaler',      'c.RegPoli(2)_PCA=10', 'd.RegPoli'
filas = ['MSE Train', 'MSE Test', 'R2-score Train', 'R2-score Test'] # definimos los nombres

datos = [
    ['2335.63', '2335.63', '2265.93', '2301.42', '2290.17'],
    ['2500.79', '2506.07', '2435.28', '2467.82', '2455.68'],
    ['2.130%', '2.130%', '5.051%', '3.563%', '4.035%'],
    ['2.100%', '1.894%', '4.665%', '3.391%', '3.866%']
]
```

```
table = pd.DataFrame(datos, columns=columnas, index=filas)
```

```
table.head(4)
```

	a.Reglineal	b.Reg linealyMinMaxScaller	c.RegPoli(2)_PCA=10	d.RegPoli(3)_PCA=3	e.Re
MSE Train	2335.63	2335.63	2265.93	2301.42	
MSE Test	2500.79	2506.07	2435.28	2467.82	



De acuerdo a las comparaciones de las métricas, aunque ninguno tiene una valore de predicción aceptable, de los modelos revisados el mejor fue el c) de regresión polinomial grado 2 y PCA de 10. Por su metrica con data de evaluación del 4.665% que es la mas alta no es suficiente para el uso en la imobiliaria para ello se esperaría un modelo de almenos el 60%, y dado que es un valor muy pequeño no se considera pueda responder adecuadamente al objetivo planteado.

▼ Estimación de los datos con el modelo seleccionado

```
pca = PCA(n_components=10, random_state=32)
pca.fit(x_train)
x_train_PCA = pca.transform(x_train)

poly_features = PolynomialFeatures(degree=2, include_bias=False)
x_poly_train = poly_features.fit_transform(x_train_PCA)
dataset = pd.DataFrame(x_poly_train)
```

```
lin_regP = LinearRegression()
lin_regP.fit(x_poly_train, y_train)
```

```
LinearRegression()
```

```
x_train_PCA.shape
```

```
(60469, 10)
```

Se aplica las funciones de preprocesamiento a los datos de los inmuebles que están próximos a publicarse, para el caso de las columnas innexistentes se colo como numero de reviews 7 y como review rate number 3 que son los datos de la mediana vista en los historicos (esto para que las

funciones no fallen, y en adición la cantidad de reviews igual luego se quita dado que es la variable

```
dataP2 = dataP.copy()
dataP2['number of reviews']=7
dataP2['review rate number']=3
dataPRED = preprocess(dataP2)
dataPRED.shape
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:63: FutureWarning: The def
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:65: FutureWarning: The def
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:68: FutureWarning: The def
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:70: FutureWarning: The def
(512, 16)
```

```
dataPRED.head(10)
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_bookable
0	27883434	Queens	Ozone Park	40.68432	-73.85862	United States	False
1	55448727	Manhattan	Civic Center	40.71317	-74.00654	United States	False
2	56858749	Queens	East Elmhurst	40.76441	-73.88943	United States	True
3	39029953	Manhattan	Gramercy	40.73442	-73.98383	United States	True
4	5567200	Manhattan	Upper West Side	40.79660	-73.97154	United States	True
5	18931756	Manhattan	Chelsea	40.74222	-73.99444	United States	False
6	19445947	Queens	Williamsburg	40.71640	-73.95438	United States	True
7	17607892	Queens	Jackson Heights	40.74879	-73.88342	United States	False
8	46760496	Manhattan	Midtown	40.75287	-73.97352	United States	True
9	8196700	Manhattan	Midtown	40.74673	-73.98667	United States	False

```
dataPRED_enc=enconderF(dataPRED)
```

```
dataPRED_enc = dataPRED_enc.drop('number of reviews',axis=1)
```

```
x_PRED_PCA = pca.transform(dataPRED_enc)
```

```
x_poly_PRED = poly_features.fit_transform(x_PRED_PCA)
```

```
y_PRED = lin_regP.predict(x_poly_PRED)
```

```
y_PRED.size
```

```
512
```

```
dataPRED = dataPRED.drop('review rate number',axis=1)
```

Se adiciona la columna 'number of reviews' con los datos de la predicción del modelo:

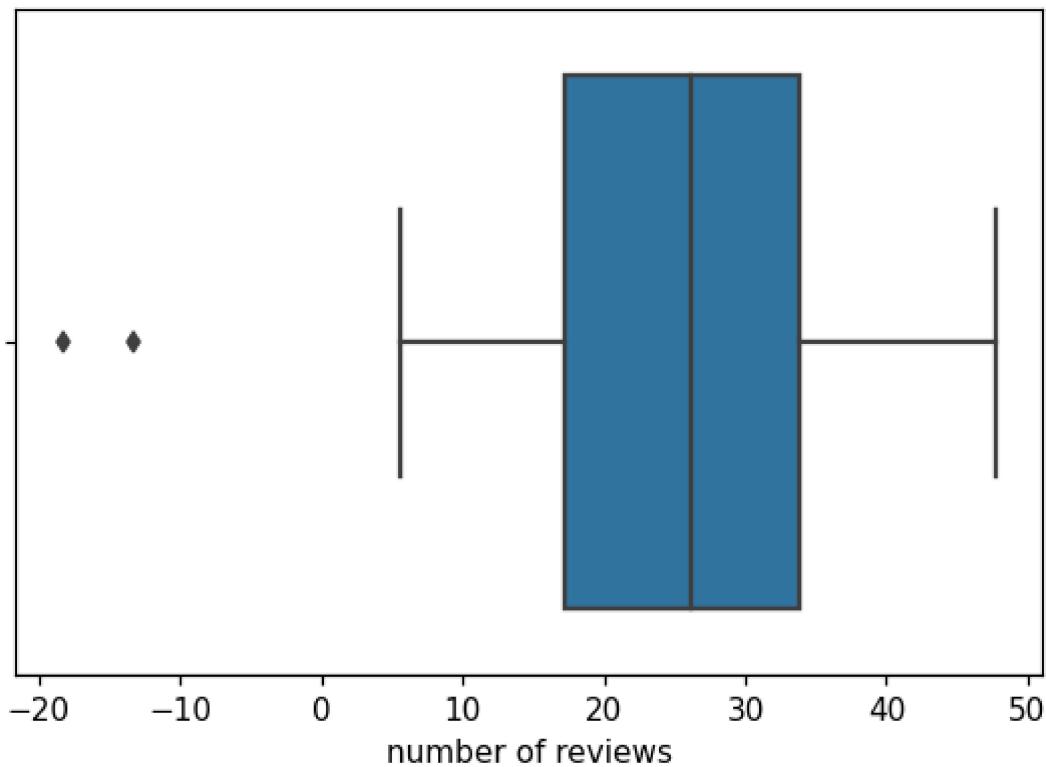
```
dataPRED['number of reviews']=y_PRED
```

```
dataPRED
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_booker
0	27883434	Queens	Ozone Park	40.68432	-73.85862	United States	F
1	55448727	Manhattan	Civic Center	40.71317	-74.00654	United States	F
2	56858749	Queens	East Elmhurst	40.76441	-73.88943	United States	.
						United	

```
sns.boxplot(x=dataPRED["number of reviews"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9336fe0710>
```



```
dataPRED["number of reviews"].describe()
```

```
count      512.000000
mean       25.938070
std        10.486011
min       -18.296706
25%        17.295370
50%        26.112665
75%        33.828690
max       47.676799
Name: number of reviews, dtype: float64
```

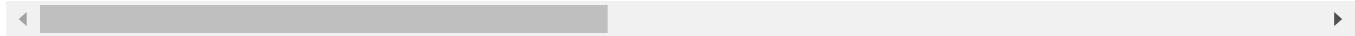
Dentro de lo visto se considera no popular si su número de reviews no supera la media (en este caso 25, en el caso del histórico era del 27)

```
dataPRED[ 'costo publicidad' ] = dataPRED[ 'price' ].apply(lambda x: x*0.02)
```

```
dataPRED[ 'costo publicidad' ] = dataPRED[ "number of reviews" ].apply(lambda x: 0 if (x >= 25) else x)
```

	id	neighbourhood group	neighbourhood	lat	long	country	instant_bookable
0	27883434	Queens	Ozone Park	40.68432	-73.85862	United States	F
1	55448727	Manhattan	Civic Center	40.71317	-74.00654	United States	F
2	56858749	Queens	East Elmhurst	40.76441	-73.88943	United States	.
3	39029953	Manhattan	Gramercy	40.73442	-73.98383	United States	.
4	5567200	Manhattan	Upper West Side	40.79660	-73.97154	United States	.
...
507	14305691	Queens	Bedford-Stuyvesant	40.69232	-73.93446	United States	F
508	21955046	Queens	Bedford-Stuyvesant	40.69122	-73.93753	United States	.
509	35125192	Queens	Williamsburg	40.71896	-73.94270	United States	F
510	46665500	Brooklyn	Fort Greene	40.68531	-73.97451	United States	.
511	53037385	Queens	Ridgewood	40.69928	-73.90616	United States	.

512 rows × 16 columns



Productos pagados de Colab - Cancela los contratos aquí

✓ 0 s se ejecutó 11:57

