TRABAJO DE FIN DE GRADO

Grado en Estadística

Título: Análisis del “atractivo” de las viviendas de Airbnb en Barcelona para un periodo de tiempo seleccionado.

Autor: Andreu Companys Rufian

Director: Laura Marquès Padreny

Departamento: Departament d’Econometria, Estadística i Economia Aplicada

Convocatoria: Juny 2022

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Resumen

Hoy en día tenemos mucha información antes de realizar el alquiler de una vivienda, toda esta información puede contener las claves que ayuden al éxito de sus propietarios, aunque estos no lo puedan ver. Utilizando herramientas de *Machine Learning*, intentaremos predecir el éxito de las diferentes viviendas y de esta manera podremos ayudar a sus propietarios a tener unas condiciones que resulten más atractivas para el público. Esto implicara obtener una base de datos, realizar las distintas transformaciones necesarias en esta, aplicar un modelo de regresión y finalmente utilizar los resultados obtenidos para hacer una función que pueda ser utilizado por un público no especializado.

Palabras clave: *Machine Learning*, Tratado de la base de datos, Datos faltantes, KNN, XGBoost, Modelos de regresión, *Overfitting*.

Abstract

Tittle: Analysis of the "attractive" of the Airbnb properties in Barcelona in a certain time period.

Nowadays we have a lot of information before renting a property, this information may be misleading to some, but it has the potential to carry the key to success to his owners, even they can't see it. Using *Machine Learning* tools we will try to predict the success of the different properties and try to find a way to help this owners to have a conditions that make his property more attractive to the public. For doing this we will begin creating a data base, make the proper transformations to this data base, apply a regression model and finally use the given results to create a function that can be used and understood for people non specialized on this matter.

Key words: Machine Learning, Data base treatment, Missing data, KNN, XGBoost, Regression models, Overfitting.

Clasificación AMS (MSC 2010)

- 68U05 Computer graphics; computational geometry
- 68T05 Learning and adaptive systems

Agradecimientos

A la directora de mi trabajo Laura Marquès, por darme la idea de este trabajo y apoyarme con todas las dificultades que hayan podido surgir.

Índice general

Capítulo 1

Introducción

1.1 Contexto

Antes de empezar con este trabajo es importante conocer Airbnb, ya que este trabajo se realizara a partir de bases de datos obtenidas de esta plataforma, pasamos entonces con una breve introducción a esta compañía.

Airbnb es una compañía que ofrece una plataforma digital dedicada a la oferta de alojamientos a particulares y turísticos (alquiler vacacional) 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. El nombre es un acrónimo de *airbed and breakfast*. Airbnb tiene una oferta de unas 2 000 000 propiedades en 192 países y 33 000 ciudades. Desde su creación en noviembre de 2008 hasta junio de 2012 se realizaron 10 millones de reservas.

FIGURA 1.1: Logotipo de la empresa Airbnb.

1.2 Objetivos

El objetivo principal de este trabajo es poder realizar un análisis del "atractivo" de los pisos en alquiler de Airbnb de Barcelona, poder entender que variables hacen que un piso sea más popular y poder hacer predicción sobre esta popularidad para así hacer los cambios deseados por el propietario y obtener un mayor número de clientes y una mayor satisfacción de estos.

Este "atractivo" será medido mediante analizar un periodo de tiempo para estos pisos y, viendo que éxito han tenido estos en dicho periodo.

1.3 Estructura del documento

El primer paso para hacer todo esto será construir la base de datos con la que trabajaremos, esta contendrá dos variables nuevas que crearemos nosotras las cuales utilizaremos para medir este "atractivo". Estas variables consistirán en seleccionar el número de días que este piso ha sido alquilado y el número de reseñas que ha obtenido en el periodo de tiempo seleccionado. Una vez creadas estas dos variables y añadidas a nuestra base de datos podríamos continuar.

El siguiente paso sería limpiar la base de datos, ya que esta es muy amplia y tiene información completamente irrelevante para el análisis mencionado previamente, donde solo interesan datos que puedan ser analizados, por ejemplo, variables que simplemente contienen URLs u otros datos no analizables no interesan y lo único que hacen es aumentar el tamaño de una ya muy grande base de datos, por lo tanto, es crucial deshacernos de ellos.

A continuación, pasaremos a hacer un análisis completo de las variables restantes de la base de datos, ya que, pese a que estas variables han sido catalogadas como validas, no todas están listas para ser utilizadas por ningún tipo de modelo de regresión, se observan distintos problemas como datos faltantes o variables que no están catalogadas de la manera óptima para ser empleadas en un modelo.

Una vez tenemos nuestra base de datos lista pasaremos a la fase final, donde utilizaremos herramientas de Machine Learning para hacer predicción de nuestras dos variables respuesta. Para esto tendremos que hacer una búsqueda de nuestros parámetros óptimos para el modelo y así obtener una predicción lo mejor posible.

Capítulo 2

Metodologia

2.1 Procedencia de los datos

Los datos para este trabajo han sido obtenidos mediante Inside Airbnb. Esta es una página web que contiene datos de Airbnb para distintas ciudades de todo el mundo. En este documento utilizaremos los datos de Barcelona para la ventana entre dos instantes del tiempo, del 10 de septiembre al 7 de diciembre de 2021.

2.2 Preparación de los datos

En este apartado utilizaremos distintos métodos para dejar nuestra base de datos en un buen estado para después poder proceder correctamente en su análisis. Utilizaremos el lenguaje Python, y para dicho tratado de los datos estaremos utilizando principalmente la librería Pandas de este. Se hará una limpieza de variables en un inicio y, luego utilizaremos distintas herramientas para solucionar todos los problemas que observamos en esta base de datos. La mayoría de técnicas empleadas en este apartado son bastante sencillas y no necesitan de una introducción previa pero, si que necesitaremos una para el algoritmo KNN el cual emplearemos para rellenar los datos faltantes de nuestra base de datos.

2.2.1 KNN

K-nearest neighbours (KNN) el cual su traducción al castellano sería los K vecinos más cercanos, es un algoritmo de machine learning supervisado, el cual puede ser utilizado para tanto problemas de clasificación como de regresión, en nuestro caso lo utilizaremos para imputar datos faltantes.

El funcionamiento de este algoritmo es sencillo, se basa en encontrar los vecinos más cercanos para la observación faltante y colocar en esta observación la media de dichos vecinos, el número de vecinos es determinado previamente por el usuario.

KNN es capaz de encontrar estos vecinos calculando la distancia euclidiana para toda la base de datos, es por eso que a pesar de ser un algoritmo que permite imputar tanto variables numéricas como categóricas, solo se pueden utilizar variables numéricas para la imputación, ya que no podemos calcular la distancia euclidiana con variables categóricas. Observamos a continuación la formula para el calculo de la distancia euclidiana.

$$d_E(P,Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

A continuación vemos un ejemplo para clasificación.

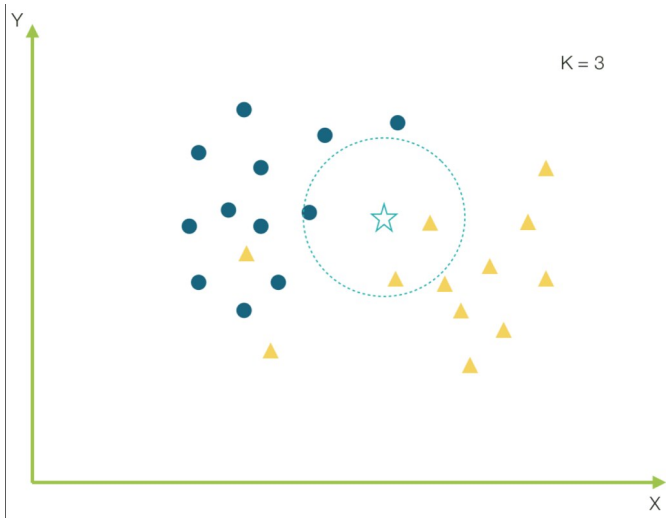

FIGURA 2.1: Ejemplo de clasificación KNN para K=3.

Como observamos en la figura 2.1 este valor faltante que tenemos, el cual viene representado con una estrella, seria imputado con el triangulo, ya que la mayoría de sus vecinos son triángulos. En el caso de ser una variable numérica, esta se vería imputada por la media de estos 3.

2.3 XGBoost

XGBoost, que proviene de *eXtreme Gradient Boosting* es un algoritmo de machine learning, el cual es una extensión de los denominados "gradient boosted decisión trees"(GBDT) diseñado para mejorar su velocidad y rendimiento. Para entender este es importante que empezemos por explicar que es un “decision tree”.

Los "decision trees" o arboles de decisión, crean un modelo de predicción para una variable respuesta (numérica o categórica indistintamente) mediante valores booleanos para diferentes preguntas formuladas, de esta manera podemos ir bajando por el árbol hasta llegar a nuestra predicción. Vemos un ejemplo a continuación.

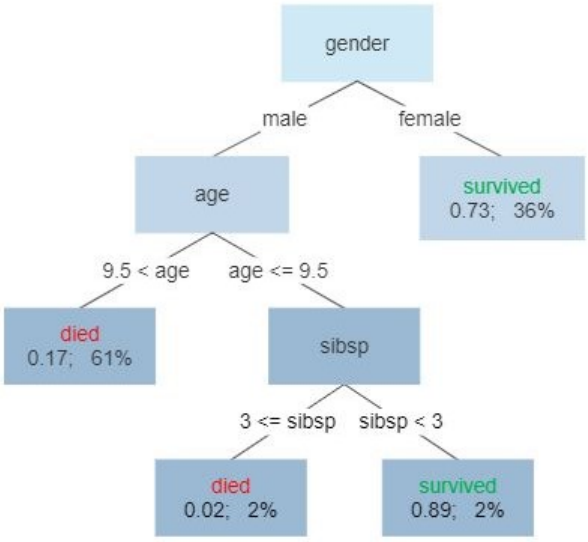

FIGURA 2.2: Ejemplo de árbol de decisión para el dataset Titanic.

En este ejemplo observamos la supervivencia de los pasajeros del Titanic y vemos que en este árbol la supervivencia de estos viene determinada por las variables genero, edad y número de familiares.

Pasamos ahora a explicar el "gradient boosting", el cual proviene del "boosting" que este se basa en mejorar un modelo débil con otros modelos débiles para así poder generar un modelo colectivo mucho más fuerte. El "gradient boosting" es una extensión de esto donde el proceso de ir generando modelos adicionalmente es realizado por un algoritmo de "gradient descent" para una función objetivo. Para el siguiente modelo a generar se utilizan los resultados del anterior, con la intención de minimizar el error del anterior, de esta manera vamos reduciendo el error del modelo y obtenemos predicciones más precisas. La predicción final es una suma ponderada de todas las predicciones de los árboles obtenidos.

Capítulo 3

Cuerpo del trabajo

3.1 Creación de la base de datos

Como hemos explicado anteriormente en el apartado de metodología obtenemos la base de datos a partir de Inside Airbnb, de donde obtendremos distintos ficheros csv con los cuales crearemos nuestra base de datos. Los csv empleados son los siguientes:

Fichero	Dimensiones	Contenido
listings_dec.csv	(15707,74)	Información de las viviendas el día 07/12/2021
listings_sept.csv	(16206,74)	Información de las viviendas el día 10/09/2021
calendar_sept.csv	(5914834,7)	Estatus diario de las viviendas a partir del día 10/09/2021

CUADRO 3.1: Ficheros utilizados para la base de datos.

Primero de todo lo que haremos es asegurarnos de que no haya observaciones repetidas en los csv de *listing_dec* y *listing_sept*, esto es fácilmente comprobable gracias a que cuentan con un *id*. Una vez hecho esto procedemos a crear las dos variables respuesta que utilizaremos, las cuales denominaremos *days_booked* y *reviews_diff*.

Empezados con la variable respuesta *days_booked*, la cual contiene el número de días que se ha alquilado la vivienda en este periodo de tiempo, para esto utilizaremos el csv *calendar_sept* y lo filtraremos para los días que hemos seleccionado, con esto obtenemos el número de días que ha sido alquilado. Observamos que el máximo de esta variable es 89, que es el número de días que hay en el periodo de tiempo seleccionado.

Pasamos ahora con variable respuesta *reviews_diff*, esta variable tendrá el número de reseñas obtenidas en el periodo de tiempo seleccionado, para esta variable lo tendremos más fácil ya que tenemos el número total de reseñas en la base de datos de septiembre y en la de diciembre, por lo tanto con una simple resta obtenemos nuestra variable respuesta.

Una vez hecho esto juntamos estas dos variables a *listing_sept*, pero antes de esto tenemos que ir con cuidado , ya que estos conjuntos de datos no tenían el mismo número de observaciones y, algunas de ellas no estaban en ambas, por lo tanto solo nos quedaremos con las observaciones que estén tanto en *listing_sept* como en *listing_dec*. Obtenemos una base de datos final con 13340 observaciones y 76 variables.

3.2 Preparación de los datos

Ahora procederemos a preparar la base de datos para que este en un estado óptimo para utilizar los modelos de machine learning deseados. Lo primero de todo será borrar las variables que no consideremos útiles, ya sea porque creamos que no vayan a ser de utilidad o porque no pueden ser utilizadas debido a su estructura, en este último caso la mayoría de ellas serán variables que contendrán un URL. Variables que también borraríamos debido a su estructura serian las variables que contengan un texto, como podrían ser la descripción de la vivienda o la información del propietario. Una vez eliminadas todas las variables que eran incorrectas ya de estructura pasaríamos a hacer el tratado de las variables que contienen signos que dificultan su lectura para el modelo.

Pasamos a tratar las variables que tienen signos, estas son, la variable *price*, que contiene un signo $ y, las variables de *rate*, las cuales contienen un signo de %. Las tratamos de la siguiente manera:

```
# QUITAMOS LOS SIMBOLOS ($,%)
listing["host_response_rate"] = listing["host_response_rate"].replace({'\%':''},
    regex = True)
listing["host_acceptance_rate"] =
    listing["host_acceptance_rate"].replace({'\%':''}, regex = True)
listing["price"] = listing["price"].replace({'\$':''}, regex = True)
# TRANSFORMAMOS LAS VARIABLES
listing["price"] = pd.to_numeric(listing["price"], errors="coerce")
listing["price"] = listing["price"].astype("float")
listing["host_response_rate"] = listing["host_response_rate"].astype("float")
listing["host_acceptance_rate"] = listing["host_acceptance_rate"].astype("float")
```

LISTING 3.1: Código empleado para eliminar los signos.

Una vez arregladas estas variables con errores de signos, empezamos a mirar las variables que contienen datos faltantes, que serán el principal problema de esta base de datos, ya que como veremos a continuación hay muchos datos faltantes. Empezamos primero mirando un gráfico del número de datos faltantes para tener un primer contacto con nuestro problema.

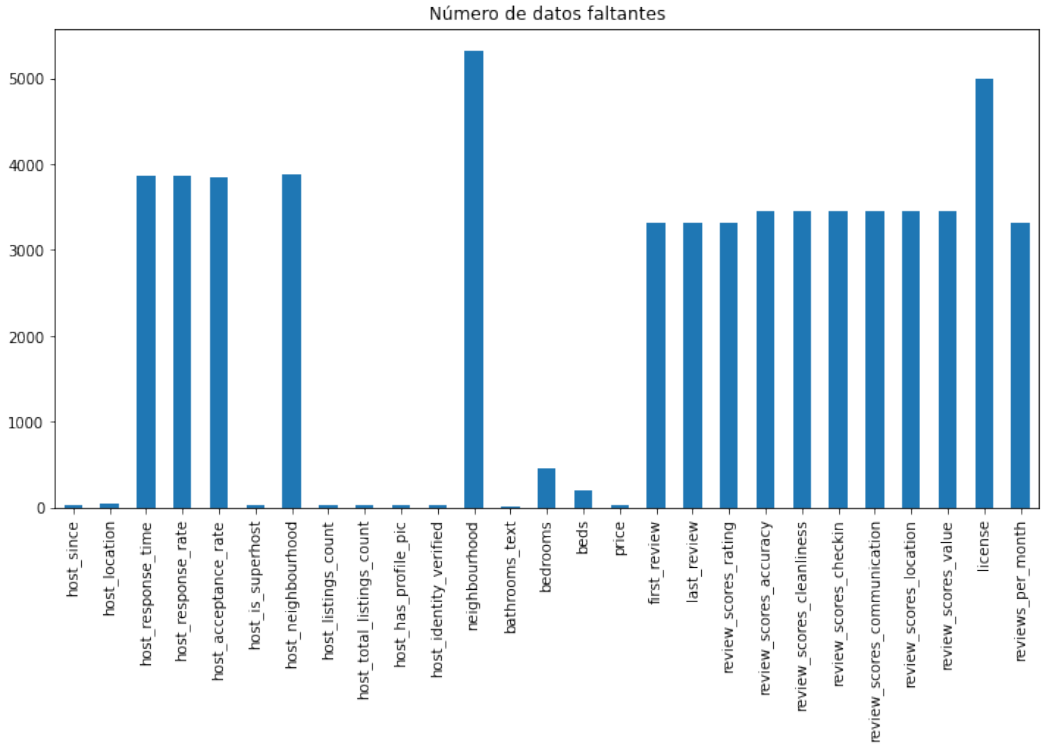

FIGURA 3.1: Gráfico para las variables que tienen datos faltantes.

Podemos observar un gran número de missings en las variables *neighbourhood* y en *license*, después de esas dos variables obtenemos perfiles más similares, ya que para la gran mayoría de variables relacionadas con *reviews* y información acerca del *host*, obtenemos un número parecido. También observamos que tenemos diferentes variables con un número muy bajo de missings pero muy parecido, estas variables se tendrán que revisar porque, debido a su bajo número, si todos estos missings coinciden para las mismas observaciones, podremos simplemente eliminar dichas observaciones y limpiar muchas variables de manera sencilla, en cambio con las que tienen un gran número de datos faltantes tendremos que buscar alguna otra solución. A continuación miramos con detalle la tabla para saber los números exactos y ver si estos coinciden con los razonamientos hechos anteriormente.

Variables	Datos faltantes
host_since	33
host_location	44
host_response_time	3859
host_response_rate	3859
host_acceptance_rate	3848
host_is_superhost	33
host_neighbourhood	3887
host_listings_count	33
host_total_listings_count	33
host_has_profile_pic	33
host_identity_verified	33
neighbourhood	5319
bathrooms_text	11
bedrooms	463
beds	195
price	33
first_review	3318
last_review	3318
review_scores_rating	3318
review_scores_accuracy	3451
review_scores_cleanliness	3449
review_scores_checkin	3455
review_scores_communication	3450
review_scores_location	3455
review_scores_value	3456
license	4996

CUADRO 3.2: Variables con datos faltantes.

Como podemos observar en el cuadro 3.2 nuestras dudas del gráfico 3.1 han sido resueltas y observamos que en las variables que tenían pocos missings, hay exactamente el mismo número de valores missing 33, menos en el caso de *host_location* y *bathrooms_text* que observamos 44 y 11 respectivamente. Esto es positivo, ya que si para todos estos valores 33 coinciden las observaciones, podremos eliminarlas, porque no interesa tener observaciones de las cuales nos falta mucha información.

Miramos las observaciones que tienen NA en *host_since* (día en que el propietario se unió

a la plataforma) y, observamos que como nos temíamos estas observaciones tienen valores faltantes en todas o prácticamente todas las variables que observamos en el cuadro 3.2, por lo tanto las eliminamos. Una vez eliminados estas observamos que restan 11 missings en *host_location*, 11 en *bathrooms_text* y 33 en *price*, debido a que son una parte ínfima de la base de datos y en el caso de la variable *price*, la consideramos una de las más importantes sino la que más por lo tanto eliminamos estas observaciones.

Después de estos ajustes que hemos realizado volvemos a observar la con los NA restantes para seguir adelante.

Variables	Datos faltantes
host_response_time	3800
host_response_rate	3800
host_acceptance_rate	3796
host_neighbourhood	3837
neighbourhood	5258
bedrooms	459
beds	191
first_review	3283
last_review	3283
review_scores_rating	3283
review_scores_accuracy	3412
review_scores_cleanliness	3410
review_scores_checkin	3416
review_scores_communication	3411
review_scores_location	3416
review_scores_value	3417
license	4968

CUADRO 3.3: Variables restantes con datos faltantes.

Observamos que las variables que contienen missings restantes tienen un número bastante elevado de NA por lo tanto necesitaremos hacer uso de alguna metodología para imputar valores faltantes, emplearemos el algoritmo KNN explicado en el apartado 2.2.1 pero antes, trataremos las variables categóricas, empezando por *license*.

Para la variable *license* vemos que esta puede contener el tipo de licencia, como por ejemplo HUTB-002062 o HUTB-002444, puede contener también *Exempt* o un NA. Para esta variable la cual contiene un gran número de tipos de licencia y debido al desconocimiento de estas

licencias, haremos esta variable más simple, le pondremos un valor *Yes* en caso de tener licencia, mantendremos los *Exempt* y en los valores NA imputaremos un *Unknown* de esta manera eliminaremos los NA y haremos más simple esta variable. De esta manera nos quedara una variable con *Unknown* en caso de no conocer su estado, *Exempt* en caso de que la licencia este exenta o *Yes* en caso de que esta tenga una licencia. Procedemos a observar el código para tal transformación y el resultado de dicha variable.

```
# TRANSFORMAMOS LA VARIABLE LICENSE
listing["license"] = listing["license"].astype("object")
listing["license"] = listing["license"].fillna("Unknown")
l1 = listing["license"][listing["license"] != "Unknown"]
l2 = listing["license"][listing["license"] != "Exempt"]
listing["license"][list(set(l1.index) & set(l2.index))] = "Yes"
listing["license"] = listing["license"].astype("category")
```

LISTING 3.2: Código empleado para transformar la variable license.

Niveles	Recuento
Yes	5268
Unknown	4968
Exempt	3016

CUADRO 3.4: Contenido de la variable license

Vemos que la variable queda muy bien distribuida y observamos que los tres niveles tienen un gran peso en la variable. Una vez tratada esta variable haremos un tratado más simple para las variables *host_response_time* y *host_neighbourhood*, para estas simplemente convertiremos los NA en un categoría *Unknown*. Hecho esto solo resta una variable categórica por tratar y esta es *neighbourhood*. Esta variable la eliminaremos, ya que contiene un gran número de valores faltantes y no esta bien codificada y, además existe una variable llamada *neighbourhood_cleansed* que esta mejor codificada y no contiene NA. Con esto terminamos el tratado de datos faltantes de las variables categóricas.

Antes de proceder con el KNN para la imputación de las variables numéricas restantes, hemos de tratar dos variables las cuales están en formato de lista, estas variables son *host verifications* y *amenities*. Estas dos pese estar en lo que podría parecer el tipo de formato *list* de python no lo están, y están en formato *string*, por lo tanto se precisara de un bucle para transformar estas variables en un conjunto de variables binarias, teniendo un 1 en caso de

que esa sea una característica de dicha vivienda o un 0 en caso contrario. Vemos a continuación el código empleado para la transformación de estas variables, en concreto la variable *host_verifications*.

```
# TRANSFORMACION A BINARIO PARA HOST_VERIFICATIONS
host_verifications = listing["host_verifications"]
cont = 0
df_host_verifications = pd.DataFrame(host_verifications)

for l in host_verifications:
    l = l.replace("[","")
    l = l.replace("]","")
    l = l.replace("\"","")
    l = l.replace("'","")
    l = l.split(",")
    if l != ['']:
        for elem in l:
            if elem[0] == " ":
                elem = elem[1:]
            if list(df_host_verifications.columns).count(elem) == 0:
                df_host_verifications[elem] = 0
            df_host_verifications[elem].iloc[cont] = 1
    cont = cont + 1
```

LISTING 3.3: Código para transformar la variable host_verifications.

Emplearíamos el mismo código para la variable *amenities* y conseguimos 659 variables binarias en el caso de *amenities* y 22 en el caso de *host_verifications*, para reducir este número nos quedaremos solo con las que tengan 1000 observaciones o más, en el caso de *amenities* y, las que tengan 500 o más en el caso de *host_verifications*. Esto resultara en que tendremos 52 variables binarias relacionadas con *amenities* y 11 relacionadas con *host_verifications*.

En cuanto a las variables *first_review* y *last_review* las eliminaremos, ya que no consideramos que tengan una gran utilidad y tenemos una variable similar que consideramos más útil, *host_since*, la cual contiene la fecha en la que el propietario se dio de alta en la plataforma, ya que esta variable esta en formato fecha, la cambiaremos a formato entero, y esta contendrá el número de días que han pasado. Utilizaremos la siguiente función.

```
from datetime import datetime

def days_between(d):
    d = datetime.strptime(d, "%Y-%m-%d")
    return abs((d - datetime.strptime("2022-09-10", "%Y-%m-%d")).days)

listing["host_since"] = listing["host_since"].apply(days_between)
```

LISTING 3.4: Función "days_between"para obtener el número de días.

Una vez hecho todo esto, lo único que queda es la imputación KNN para los datos faltantes de las variables numéricas. Para esto tenemos que crear una matriz que contenga todas las variables numéricas de la base de datos, y también no incluiremos algunas variables numéricas ya que no aportan ninguna información, como por ejemplo todas las variables relacionadas con *id* o por ejemplo también las variables de latitud y longitud. También utilizaremos las variables categóricas que están catalogadas como Verdadero y Falso, ya que estas las convertiremos en 1 y 0. Después de esto ejecutaremos la función KNNImputer de paquete sklearn con K=5.

```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5)
df = pd.DataFrame(imputer.fit_transform(df),columns = df.columns)
```

LISTING 3.5: Aplicación del KNN para imputación con K 5.

Observamos que después de esto no tenemos datos faltantes en la base de datos pero, ya que hemos escogido una K de 5 esta hace la media de 5 vecinos y, en el caso de las variables *beds* y *bedrooms*, que son variables enteras, obtenemos algunos valores con decimales, por lo tanto redondearemos estas dos variables.

Para finalizar observaremos la matriz de correlaciones, ya que no interesa tener variables que estén correlacionadas entre si. Observamos una fuerte correlación entre las variables relacionadas con el mínimo y el máximo de noches. Debido a su fuerte correlación nos quedaremos solo con *minimum_nights* y *maximum_nights*, ya que son las más simples y estas no están correlacionadas entre si. También observamos que las variables *host_listings_count* y *host_total_listings_count* son la misma, con lo que eliminamos esta última. Con esto finalizamos la preparación de los datos y nos guardamos el archivo csv resultante que tiene 13252 observaciones y 104 variables.

3.3 Análisis descriptivo de la base de datos

Este apartado lo dividiremos en cuatro partes, en función del tipo de variables, en nuestra base de datos tenemos variables numéricas, como podría ser por ejemplo la variable *price*, tenemos variables binarias, como podrían ser todas las variables obtenidas de *host_verifications* y *amenities*, tenemos variables categóricas, como podrían ser *host_response_time* o *host neighbourhood* y finalmente tenemos nuestra dos variables respuesta *days_booked* y *reviews_diff*. Comentaremos las variables en los siguientes subapartados pero, debido a las grandes dimensiones de esta base de datos no todas las variables serán comentadas en su totalidad, pero todas podrán ser encontradas en Github.

Antes de empezar con las distintas variables vamos a realizar un mapa de calor para el número de viviendas de nuestra base de datos, de esta manera podremos ubicar en que partes de Barcelona hay más viviendas en alquiler.

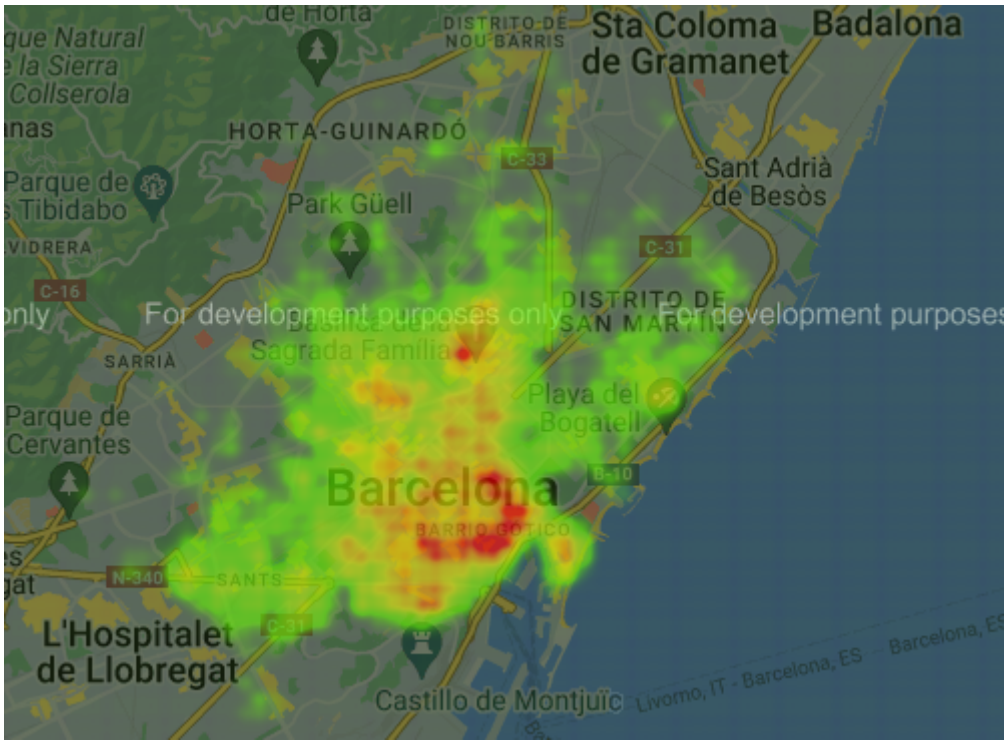

FIGURA 3.2: Mapa de calor para el número de viviendas en alquiler.

Observamos que tenemos un mayor número de viviendas en alquiler en el centro y, esto va disminuyendo a medida que nos alejamos de este. Una vez visto esto pasamos con el análisis de las variables numéricas.

3.3.1 Variables numéricas

Variable	Definición	Ejemplo
host_response_rate	Porcentaje de respuesta del propietario	100
host_since	Días desde que el propietario puso un alquiler	4617
latitude	Latitud de la ubicación de la vivienda	41.40556
calculated_..._shared_rooms	Propiedades habitaciones compartidas en la zona	0
calculated_..._private_rooms	Propiedades habitaciones privadas en la zona	0
calculated_..._entire_homes	Propiedades casas enteras en la zona	19
calculated_..._count	Propiedades en alquiler en la zona	19
number_of_reviews	Número de reseñas	21
maximum_nights	Máximo de noches	1125
minimum_nights	Mínimo de noches	1
accommodates	Capacidad máxima	8
review_scores_value	Valoración de las reseñas general	4.3
host_acceptance_rate	Porcentaje de aceptación del propietario	83
review_scores_location	Valoración de las reseñas de localización	4.75
review_scores_communication	Valoración de las reseñas de comunicación	4.9
review_scores_checkin	Valoración de las reseñas de checkin	4.8
review_scores_cleanliness	Valoración de las reseñas de limpieza	4.75
review_scores_accuracy	Valoración de las reseñas en precisión	4.55
review_scores_rating	Valoración de las reseñas en rating	4.4
price	Precio del alquiler de la vivienda	121
beds	Número de camas	6
bedrooms	Número de dormitorios	3
host_listings_count	Número de propiedades en alquiler	35
longitude	Longitud de la ubicación de la vivienda	2.17262

CUADRO 3.5: Definición de las variables numéricas de la base de datos.

Ahora vamos a ver gráficamente algunas de las variables numéricas más importantes. Empezamos por la variable *price*.

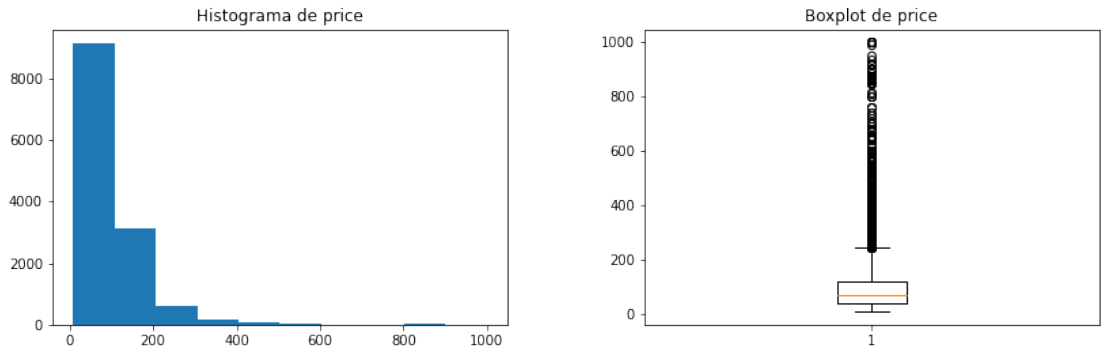

FIGURA 3.3: Gráficos de la variable price.

En estos gráficos podemos observar que la mayoría de viviendas que hay en alquiler su precio esta entre 0-100 euros, observamos también que a partir de 200 euros ya no hay prácticamente viviendas en alquiler por esos precios y, después de los 300 euros no encontramos prácticamente nada. Podemos observar que la media del precio esta en torno a 100 y que los precios mayores de 250 aproximadamente estarían fuera del 95 %. Observamos a continuación un grafico de calor para la variable *price* para identificar en que zonas de Barcelona resulta más caro alquilar en Airbnb.

FIGURA 3.4: Mapa de calor para la variable price.

En este mapa podemos observar que encontramos los pisos más caros en la zona de la playa y en el centro, también observamos precios elevados en las zonas de Vallvidriera y Pedralbes aunque estas ultimas pueden ser confusas ya que este gráfico se ha realizado para la media

de precio en el área con lo que si en esas zonas hay pocos pisos pero alguno de ellos es muy caro, este dispararía esa media.

Comentaremos también la variable *host_acceptance_rate* ya que es de interés ver que porcentaje de aceptación tienen los propietarios de Airbnb.

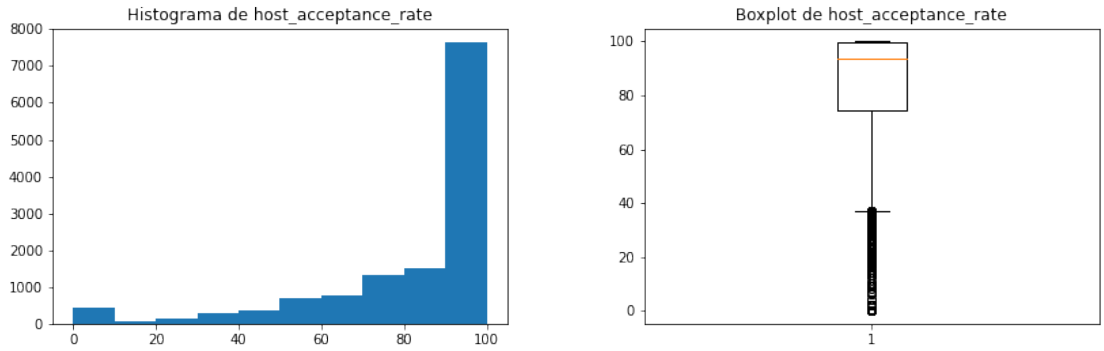

FIGURA 3.5: Gráficos de la variable host_acceptance_rate.

Podemos ver que la gran mayoría de propietarios tienen un porcentaje de aceptación entre el 90 y el 100, teniendo este grupo un gran peso sobre la base de datos. Vemos también que el porcentaje va disminuyendo poco a poco, aunque observamos que hay un grupo de propietarios que se encuentra entre el 0 y el 10, cosa que es curiosa ya que este grupo tiene más peso que el que va entre 10 y 40, siendo estos los más cautelosos con quien meter en su propiedad.

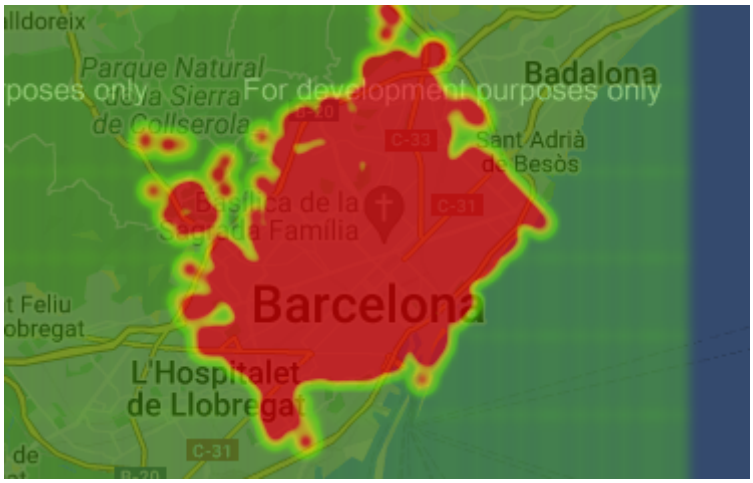

FIGURA 3.6: Mapa de calor para la variable host_acceptance_rate.

A diferencia con la variable *price* parece que aquí no importa la ubicación y vemos que esos propietarios comentados antes no están vinculados a ningún sitio en particular de Barcelona, en general la tendencia es a aceptar a la gran mayoría de clientes y esto no varia para ningún barrio.

Es interesante conocer de nuestra base de datos, cuantos propietarios, son propietarios de una o dos viviendas o, cuantos de ellos, son propietarios de muchas de ellas, para esto miraremos la variable *calculated_host_listings_count*.

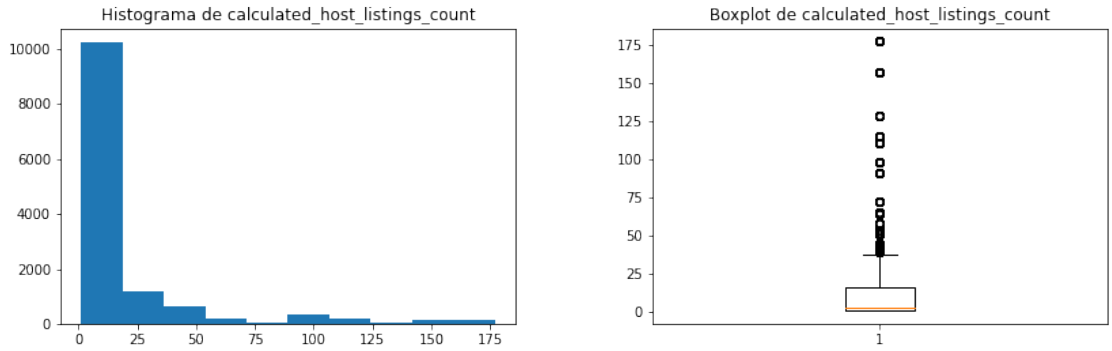

FIGURA 3.7: Gráficos de la variable calculated_host_listings_count.

Observamos que la gran mayoría de propietarios se encuentran con un número bajo, así lo confirma el boxplot el cual nos muestra una media muy baja y el histograma también pero, hay un error en estos dos gráficos anteriores y es que estamos utilizando la base de datos entera y por lo tanto si un propietario tiene por ejemplo 150 viviendas, en la base de datos habrá 150 que marquen que el propietario tiene 5 viviendas, por lo tanto con esta variable es necesario hacer un *subset* de la base de datos en el cual eliminemos los duplicados de la variable *host_id*.

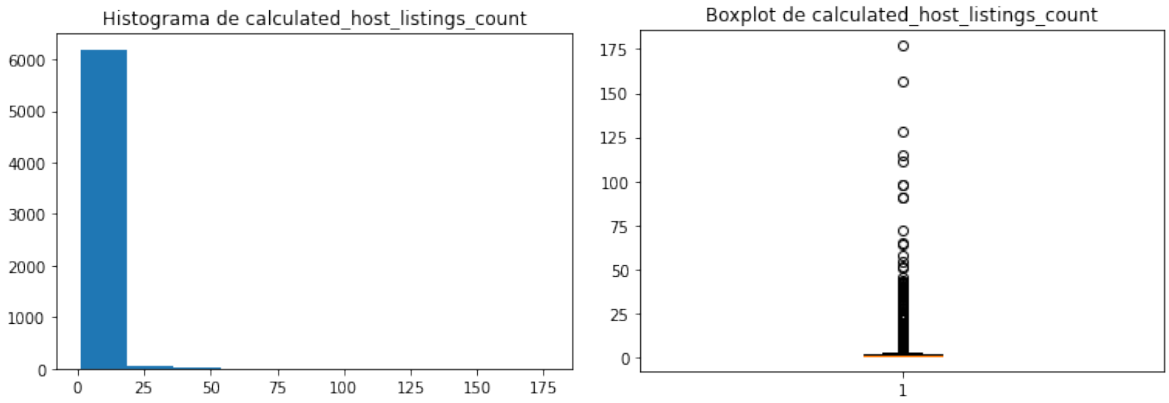

FIGURA 3.8: Gráficos de la variable calculated_host_listings_count después del subset.

Observamos ahora que claramente la inmensa mayoría de propietarios de nuestra base de datos son particulares que tienen una sola vivienda en alquiler y, después hay algunas pocas empresas que tienen un gran número de viviendas en alquiler en Barcelona. Finalmente echaremos un ojo a algunas de las variables de puntuación de reseñas y miraremos un poco como están distribuidas.

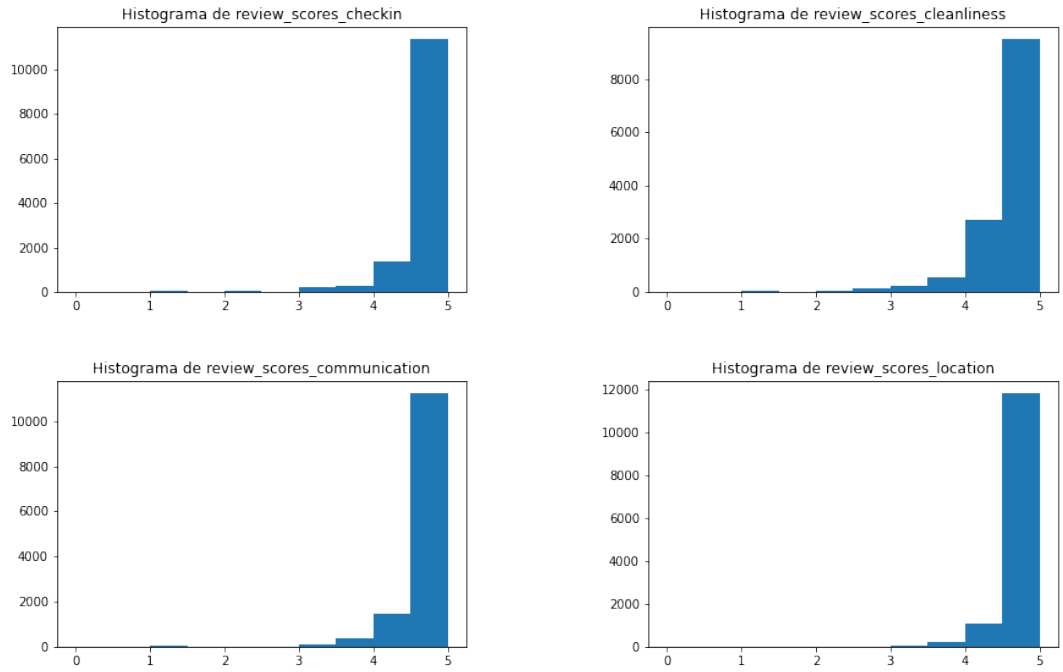

FIGURA 3.9: Gráficos de algunas variables de puntación de reseñas.

Observamos que todas estas puntuaciones de reseñas tienen una distribución muy similar, teniendo todas estas su mayoría de viviendas en una nota entre 4.5 y 5, un grupo notable de viviendas entre 4 y 4.5 y, poquísimas viviendas que este por debajo de 4. Una vez vistas estas variables pasaremos a comentar las variables binarias que encontramos en nuestra base de datos.

3.3.2 Variables binarias

En cuanto a las variables binarias tenemos 4 originarias de la base de datos y, tenemos todas las que resultaron al convertir las variables *host_verifications* y *amenities*, de los cuales obtenemos 12 y 52 respectivamente. Comentamos a continuación las 4 variables binarias originales.

Variable	Definición	Ejemplo
host_is_superhost	El propietario tiene la acreditación de superhost	f
host_has_profile_pic	El propietario tiene foto de perfil	t
host_identity_verified	El propietario ha confirmado su identidad	t
instant_bookable	El cliente puede reservar sin la necessidad de confirmación	f

CUADRO 3.6: Definición de las variables binarias de la base de datos.

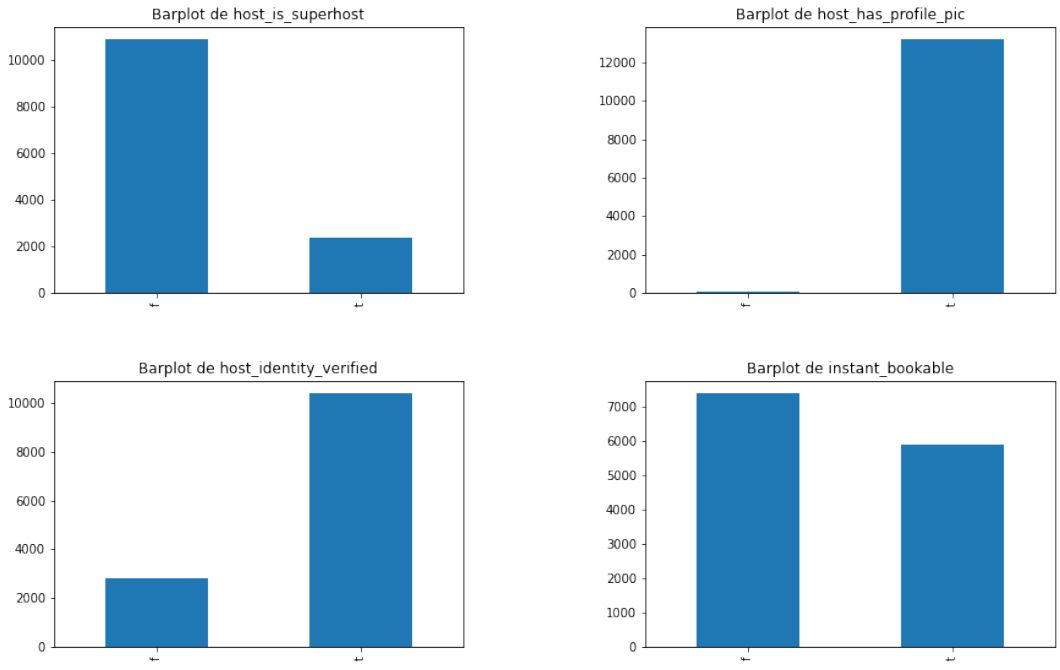

FIGURA 3.10: Gráficos para las variables binarias originales.

Podemos ver que hay un porcentaje bajo de propietarios que tengan la acreditación de *superhost*, observamos que hay un porcentaje ínfimo de propietarios que no tienen foto de perfil y, la mayoría de propietarios tienen su identidad verificada. En cuanto a la variable *instant_bookable*, observamos una distribución más pareja pero vemos la mayoría de propietarios necesitan de confirmación.

Pasamos ahora a hablar de la variable *host_verifications*, esta contiene las posibles maneras que tiene el propietario para confirmar el alquiler. Pasamos a ver 4 de ellas.

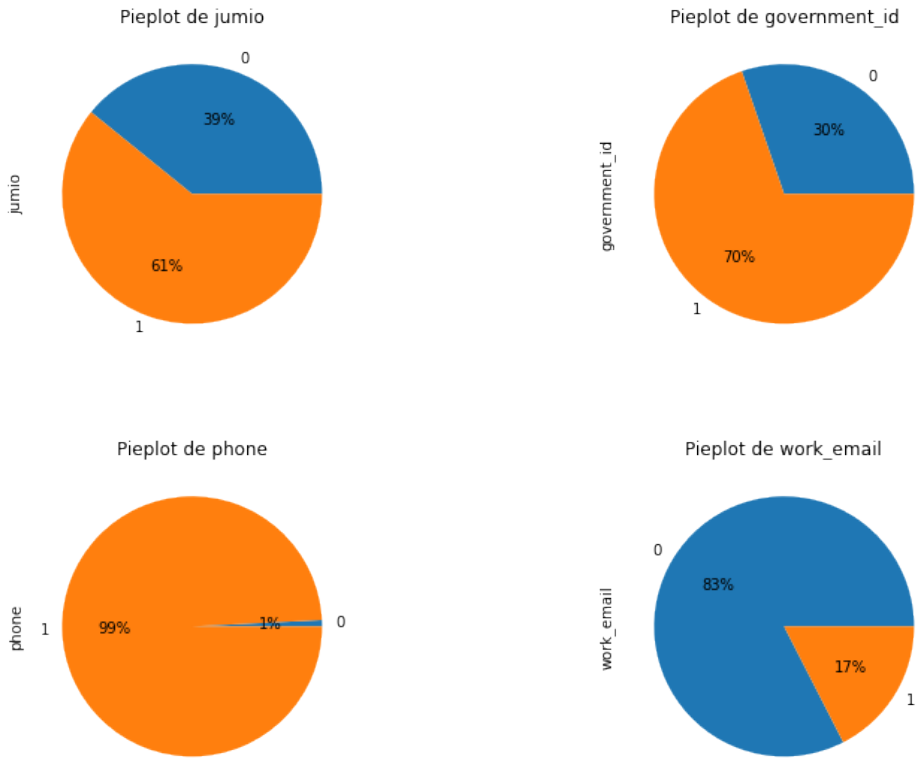

FIGURA 3.11: Gráficos para algunas variables binarias de host_verifications.

Observamos que se ofrecen distintas maneras de verificación, vemos que por ejemplo en el caso del teléfono tiene una gran presencia y, en cambio todas las demás no tanto, aún así solo hemos visto 4 de ellas hay otras como por ejemplo podrían ser *google* o *email*.

Vemos también 4 de las variables binarias resultantes de la variable original *amenities*, esta variable contiene todas las cosas que contiene un piso, pudiendo ser algunas de ellas por ejemplo: televisión, wifi, cocina, horno, secador, friegaplatos, etc.

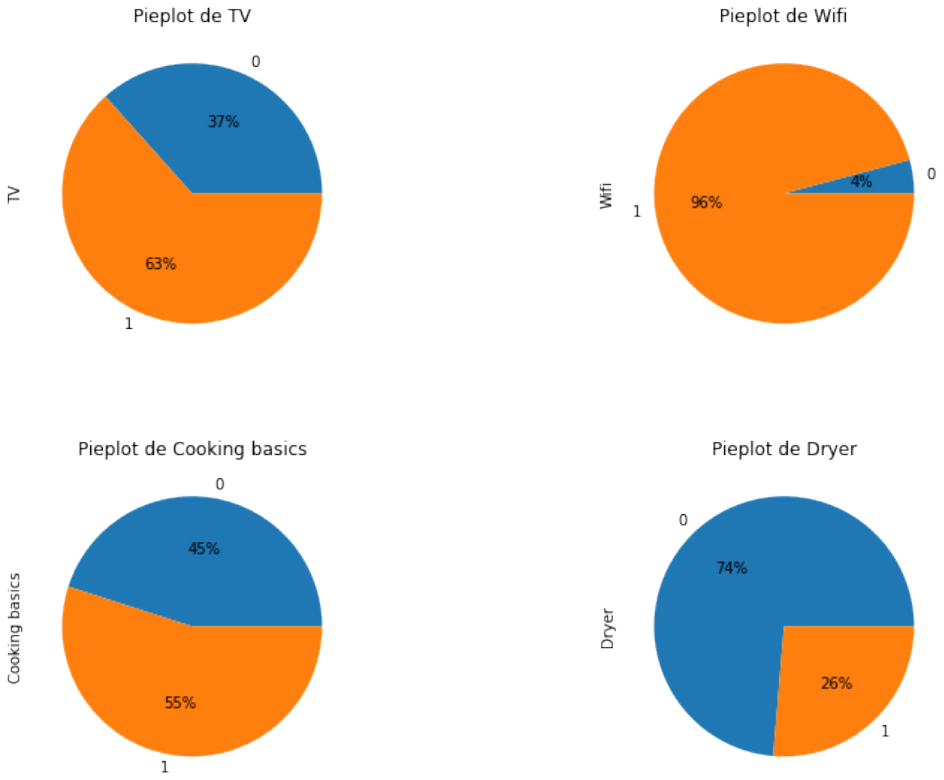

FIGURA 3.12: Gráficos para algunas variables binarias de amenities.

Podemos ver como la gran mayoría de viviendas ofrecen wifi, vemos también que aproximadamente una tercera parte de las viviendas no ofrece televisor, una mitad de estas no ofrece elementos básicos para cocinar y una cuarta parte de estas no tienen secador. Estas son solo 4 de las 52 variables *amenities* que tenemos pero todas estas siguen esta misma estructura, una vez vistas estas variables pasamos a las variables categóricas.

3.3.3 Variables categóricas

Para las variables categóricas tenemos unas cuantas de estas que serán difíciles de interpretar gráficamente ya que tienen muchos niveles, por eso en este apartado haremos una definición de todas pero solo miraremos algunas pocas de ellas en el apartado gráfico.

Variables	Definición	Ejemplo
host_location	Localización del propietario	Catalonia, Spain
host_response_time	Tiempo de respuesta del propietario	within an hour
host_neighbourhood	Barrio del propietario	La Barceloneta
neighbourhood_cleansed	Barrio de la vivienda	Sant Antoni
neighbourhood_group_cleansed	Distrito de la vivienda	Eixample
property_type	Tipo de vivienda	Entire rental unit
room_type	Tipo de habitación	Entire home/apt
bathrooms_text	Número de baños y tipo de estos	1 bath
license	Información de la licencia	Yes

CUADRO 3.7: Definición de las variables categóricas de la base de datos.

Observamos a continuación algunas de estas variables.

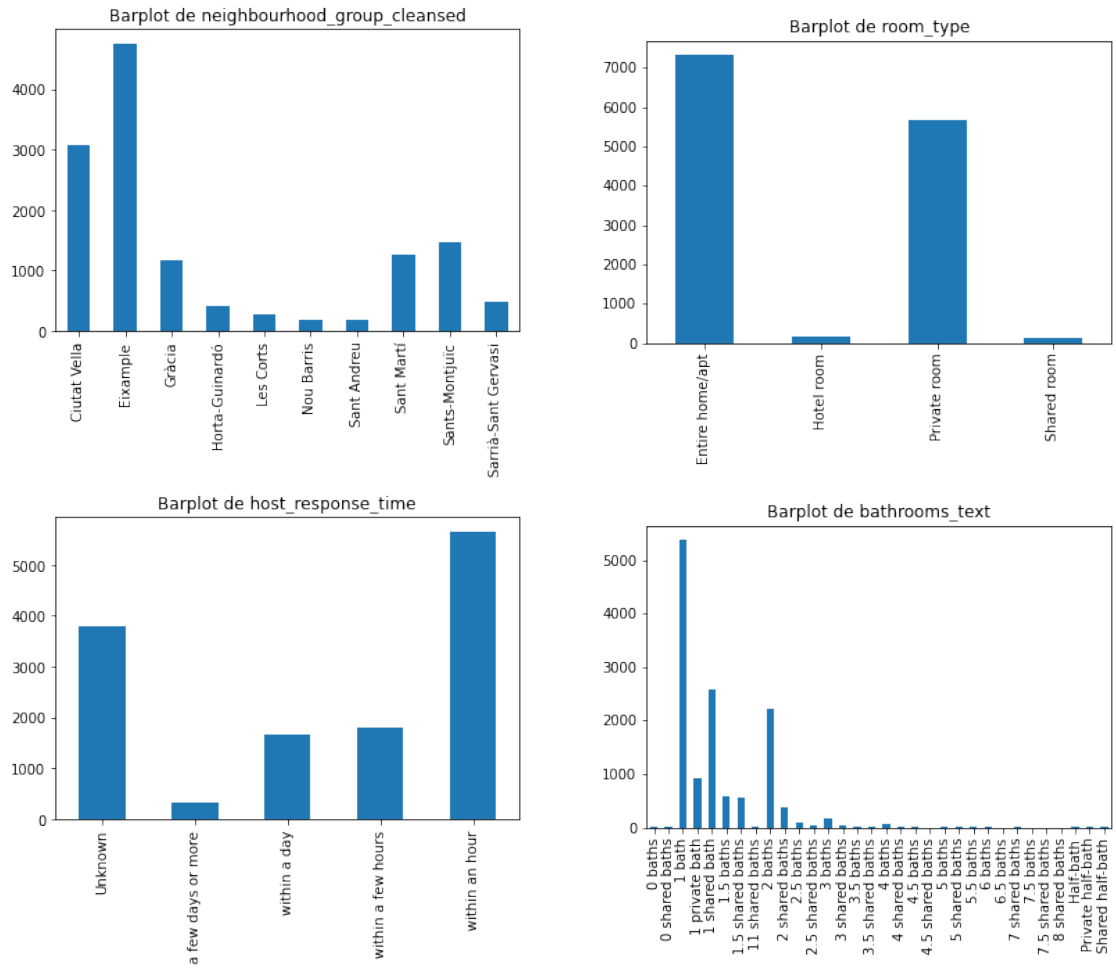

FIGURA 3.13: Gráficos de algunas variables categóricas.

Primero de todo observamos la variable que nos muestra los distritos, podemos ver que el distrito con más viviendas es l'Eixample, esto concuerda con el mapa de calor observado en la figura 3.2, donde podíamos ver que el centro era la zona con más viviendas. Podemos ver que prácticamente todas las viviendas son o apartamentos enteros o habitaciones privadas. En cuanto al tiempo de respuesta observamos que la mayoría de propietarios contestan en menos de una hora y también hay una gran parte de clientes de los cuales no tenemos esta información. Finalmente miramos los baños de las viviendas y podemos ver que la gran mayoría tienen 1 baño.

Hablaremos ahora de los distintos niveles que tienen las variables que no hemos mostrado gráficamente. Para la variable *host_location* tenemos un total de 545 niveles, para *host neighbourhood*, tenemos 125, para *neighbourhood_cleansed* tenemos 70 y finalmente para la variable *property type* tenemos 51. Una vez comentadas estas ultimas variables, pasaremos a las únicas variables que faltan, estas son nuestras dos variables respuesta.

3.3.4 Variables respuesta

Técnicamente estas dos variables son numéricas pero queríamos dedicarles un apartado para estas ya que son las variables en las que se basara nuestro modelo.

Variable	Definición	Ejemplo
days_booked	Número de dias alquilados en el periodo de tiempo	88
reviews_diff	Número de reseñas obtenidas en el periodo de tiempo	10

CUADRO 3.8: Definición de las variables respuesta de la base de datos.

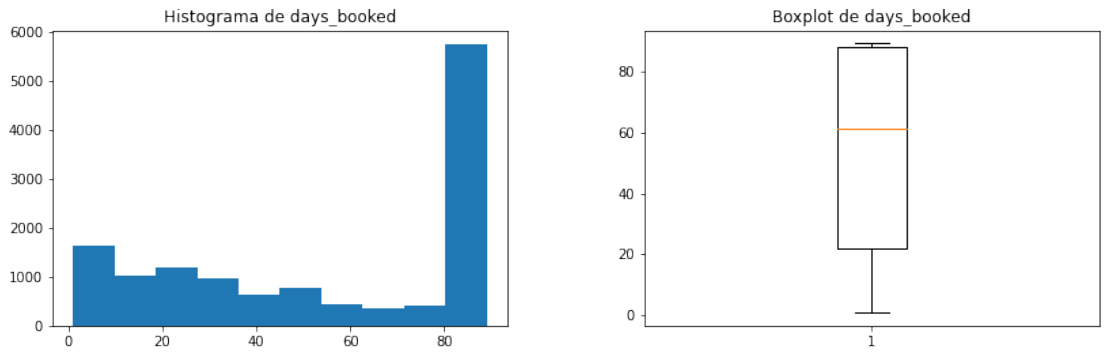

FIGURA 3.14: Gráficos de la variable respuesta days_booked.

Observamos que la gran mayoría de estos pisos se alquilo más de 80 días, sabemos que el máximo de esta variable es de 89 días, ya que son el número de días que hay en el periodo de tiempo seleccionado. Vemos también que la media de esta se sitúa alrededor de 60, miraremos a continuación también esta variable situada en un mapa de Barcelona.

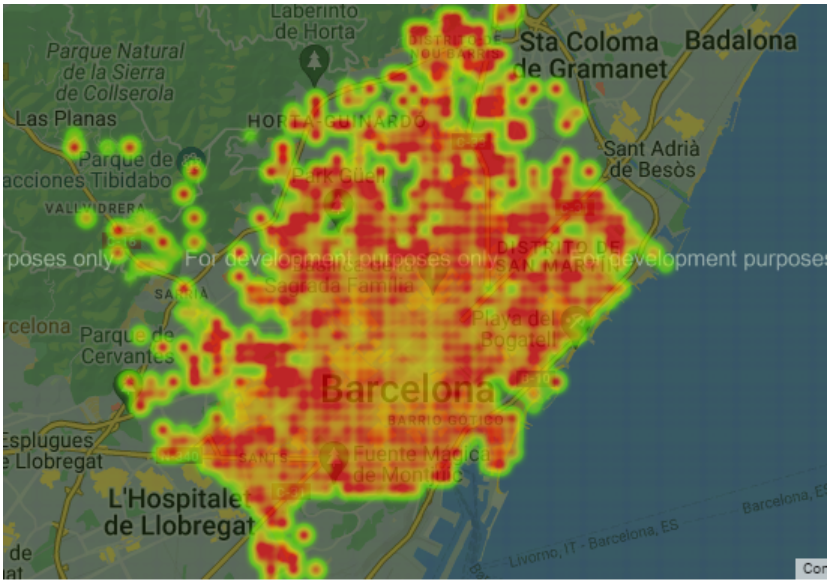

FIGURA 3.15: Mapa de calor para la variable respuesta days_booked.

Como podemos ver en este mapa la variable *days_booked* no esta vinculada a su ubicación y podemos ver que independientemente del sitio parece que estos tienen éxito. Seguimos a continuación con la variable respuesta *reviews_diff*.

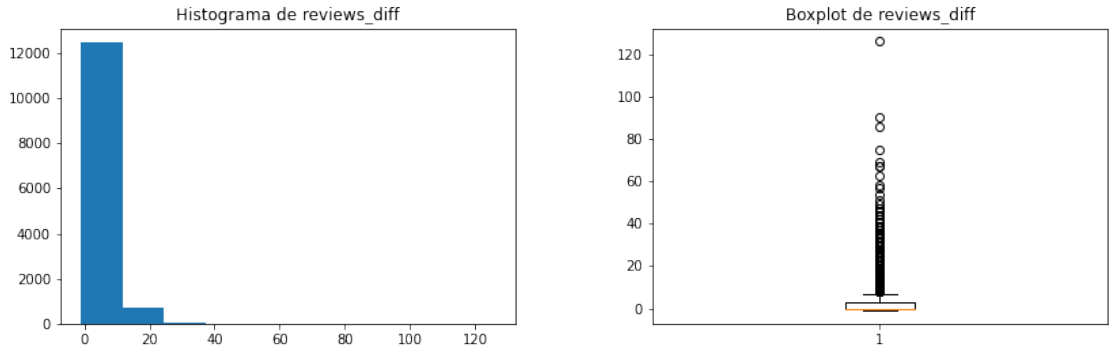

FIGURA 3.16: Gráficos de la variable respuesta reviews_diff.

Para esta variable observamos que la mayoría de observaciones si sitúan entre el 0 y el 10, unas pocas entre el 10 y el 20 y, una cantidad ínfima tiene una cantidad superior a 20.

Vemos en el boxplot que la media de esta es muy baja y, observamos que existen algunos valores singulares realmente muy altos ya que hay que recordar que estas reseñas son las obtenidas en un periodo de 89 días, por lo tanto valores como algunos que se observan en el boxplot son realmente descabellados, por eso de cara a hacer el mapa de calor de esta variable fijaremos un valor máximo de 25, ya que sin este máximo no podríamos sacar nada en clave y, consideramos que 25 reseñas en un periodo de 89 días ya es un gran éxito.

FIGURA 3.17: Mapa de calor para la variable respuesta reviews_diff.

En esta variable observamos diversos puntos que reciben muchas reseñas y estos están distribuidos por distintas partes de toda Barcelona, lo único que parece que podríamos sacar en clave es que la zona de el centro parece recibir menos reseñas.

Una vez comentadas las variables de la base de datos, lo único que nos queda es hacer la transformación de nuestras dos variables respuesta, ya que queremos tener estas dos en un formato de puntuación de 0 a 5. Realizaremos esta transformación cogiendo los valores de estas variables, dividiendo por su máximo y luego multiplicando por 5. En el caso del número de reseñas dividiremos por 12 y luego fijaremos a 5 los valores que resulten más grandes que este. Pasamos a ver esta transformación.

```
# TRANSFORMACION DE NUESTRAS VARIABLES RESPUESTA
df["days_booked"] = (df["days_booked"]/max(df["days_booked"])) * 5
df["reviews_diff"] = (df["reviews_diff"]/12) * 5
df["reviews_diff"].where(df["reviews_diff"] <= 5, 5, inplace=True)
df["reviews_diff"].where(df["reviews_diff"] >= 0, 0, inplace=True)
df.rename(columns = {'days_booked':'booking_score', 'reviews_diff':'review_score'},
    inplace = True)
```

LISTING 3.6: Código empleado para transformar las variables respuesta.

Ya con estas dos variables listas tenemos la base de datos preparada para aplicar nuestro modelo.

3.4 Selección del modelo XGBoost

Utilizaremos el modelo XGBoost el cual en Python lo encontraremos en el paquete xgboost. Para este modelo emplearemos distintas funciones tanto de este paquete xgboost como del paquete sklearn.

Antes de empezar tendremos que realizar *one hot encoding* para las variables categóricas ya que XGBoost no las puede tratar. Transformamos estas variables a través de la función *OneHotEncoder* del paquete sklearn, observamos el código a continuación.

```
# TRANSFORMACION ONE HOT ENCODING
X_cat = X.select_dtypes("category")
encoder = OneHotEncoder()
encoded_x_array = encoder.fit_transform(X_cat[X_cat.columns])
colnames = encoder.get_feature_names(X_cat.columns)
encoded_x = pd.DataFrame(encoded_x_array.todense(), columns = colnames)
```

LISTING 3.7: Código empleado para aplicar el one hot encoding.

Obtenemos una base de datos con 938 variables, más adelante utilizaremos una función para deshacernos de las variables que no sean relevantes, debido a que sabemos que este gran número de variables implicara seguramente *overfitting*.

Una vez la base de datos lista podemos empezar a estimar los distintos modelos. Para poder conseguir llegar a nuestro modelo óptimo utilizaremos distintas funciones que nos ayudaran.

Primero de todo utilizaremos la función *GridSearchCV*, la cual obtiene la mejor combinación de parámetros para el conjunto de parámetros que se quiere estudiar, de esta manera podremos probar muchos valores distintos en una simple función. Observamos un ejemplo:

```
params = { 'max_depth': [10,12,15],
           'learning_rate': [0.01, 0.03, 0.05, 0.075],
           'n_estimators': [750, 1000, 1250, 1500],
           'colsample_bytree': [0.3, 0.45]}

xgbr = xgb.XGBRegressor(seed = 123)

clf = GridSearchCV(estimator=xgbr,
                   param_grid=params,
                   scoring='neg_mean_squared_error',
                   verbose=1)

clf.fit(selected_X, y1)

print("Best parameters:", clf.best_params_)
print("Lowest RMSE: ", (-clf.best_score_)**(1/2.0))
```

LISTING 3.8: Ejemplo de GridSearchCV.

Después para los mejores parámetros obtenidos utilizaremos la función cv, la cual nos permite utilizar la *Cross Validation*, esto lo que hace es dividir el dataset en 5, luego empleara una de estas partes como test y las demás como train, repetirá esto para cada una de las divisiones y obtendremos la media de RMSE de todas estas, esto hará que obtengamos una estimación del RMSE más robusta y nos ayudara a detectar más fácilmente el *overfitting*. Observamos un ejemplo de código y su respectiva salida.

```
params = {'objective':'reg:linear',
          'max_depth': 10,
          'learning_rate': 0.01,
          'colsample_bytree': 0.3}

data_dmatrix = xgb.DMatrix(data=selected_X,label=y1)

cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=3,
    num_boost_round=1000, early_stopping_rounds=10,metrics="rmse", as_pandas=True,
    seed=123)
```

LISTING 3.9: Ejemplo de la función cv de xgboost.

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
995	1.043990	0.016611	1.406296	0.021401
996	1.043658	0.016335	1.406290	0.021407
997	1.043336	0.016355	1.406288	0.021413
998	1.043141	0.016420	1.406264	0.021390
999	1.043112	0.016434	1.406256	0.021385

CUADRO 3.9: Ejemplo de salida de la función cv

También emplearemos la función *SelectFromModel*, la cual nos permitirá eliminar las variables que tienen una importancia menor que el valor que nosotros asignaremos, de esta manera eliminaremos aquellas variables que no resulten relevantes y podremos reducir el *overfitting*, ya que sabemos que un gran número de observaciones puede provocar *overfitting*. Observamos el código:

```
# select features using threshold
selection = SelectFromModel(selection_model, threshold=0.003, prefit=True)
select_X_train2 = selection.transform(select_X_train)
feature_idx = selection.get_support()
select_X_train2 = pd.DataFrame(select_X_train2,
    columns=selected_X.columns[feature_idx])

# train model
selection_model2 = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree =
    0.3, learning_rate = 0.1, max_depth = 10, alpha = 10, n_estimators = 1000)
selection_model2.fit(select_X_train2, y1_train)

# eval model
select_X_test2 = selection.transform(select_X_test)
y1_pred2 = selection_model2.predict(select_X_test2)
train_pred2 = selection_model2.predict(select_X_train2)
select_X_test2 = pd.DataFrame(select_X_test2,
    columns=selected_X.columns[feature_idx])
```

LISTING 3.10: Ejemplo de la función SelectFromModel de sklearn.

A través de estas 3 funciones explicadas encontraremos nuestro mejor modelo, primero buscamos los mejores parámetros, después miramos el rendimiento de este y observamos si hay *overfitting*, después obtenemos una nueva base de datos las variables más importantes y

volvemos a repetir el proceso, repetimos esto hasta que eventualmente encontremos nuestro modelo óptimo.

A continuación observamos las iteraciones descritas anteriormente para la variable *booking_score*.

3.4.1 Para la variable respuesta *booking_score*

Iteración	Variables	Mejores Parámetros	Train - RMSE	Test - RMSE
1	938	max_depth = 10 learning_rate = 0.05 colsample_bytree = 0.3 n_estimators = 1000	0.501666	1.339469
2	97	max_depth = 10 learning_rate = 0.01 colsample_bytree = 0.3 n_estimators = 1000	1.105746	1.421048
3	67	max_depth = 8 learning_rate = 0.01 colsample_bytree = 0.45 n_estimators = 1500	1.217414	1.416290
4	34	max_depth = 7 learning_rate = 0.005 colsample_bytree = 0.45 n_estimators = 2100	1.266820	1.429755

CUADRO 3.10: Modelos estudiados para la variable respuesta booking_score.

Podemos observar que en el modelo inicial teníamos una mejor predicción que en los modelos con variables reducidas pero observamos un claro caso de *overfitting*, una vez reducimos las variables vemos como este *overfitting* se va reduciendo y, en el modelo 4 el cual consideramos definitivo vemos que el RMSE de train y test son más cercanos, siendo el de train un poco menor pero es lo normal, observamos también que tampoco hemos perdido mucho en el conjunto de test respecto al modelo inicial. Hay que remarcar también que las predicciones del conjunto de test para el modelo inicial eran muy genéricas, este modelo simplemente predecía con valores entre 2 y 2.8 y debido a esto obtenía este error, en cambio en el modelo escogido este hace predicciones más concretas, errando estas alguna vez. Observamos a continuación las variables del modelo definitivo y su importancia dentro de este.

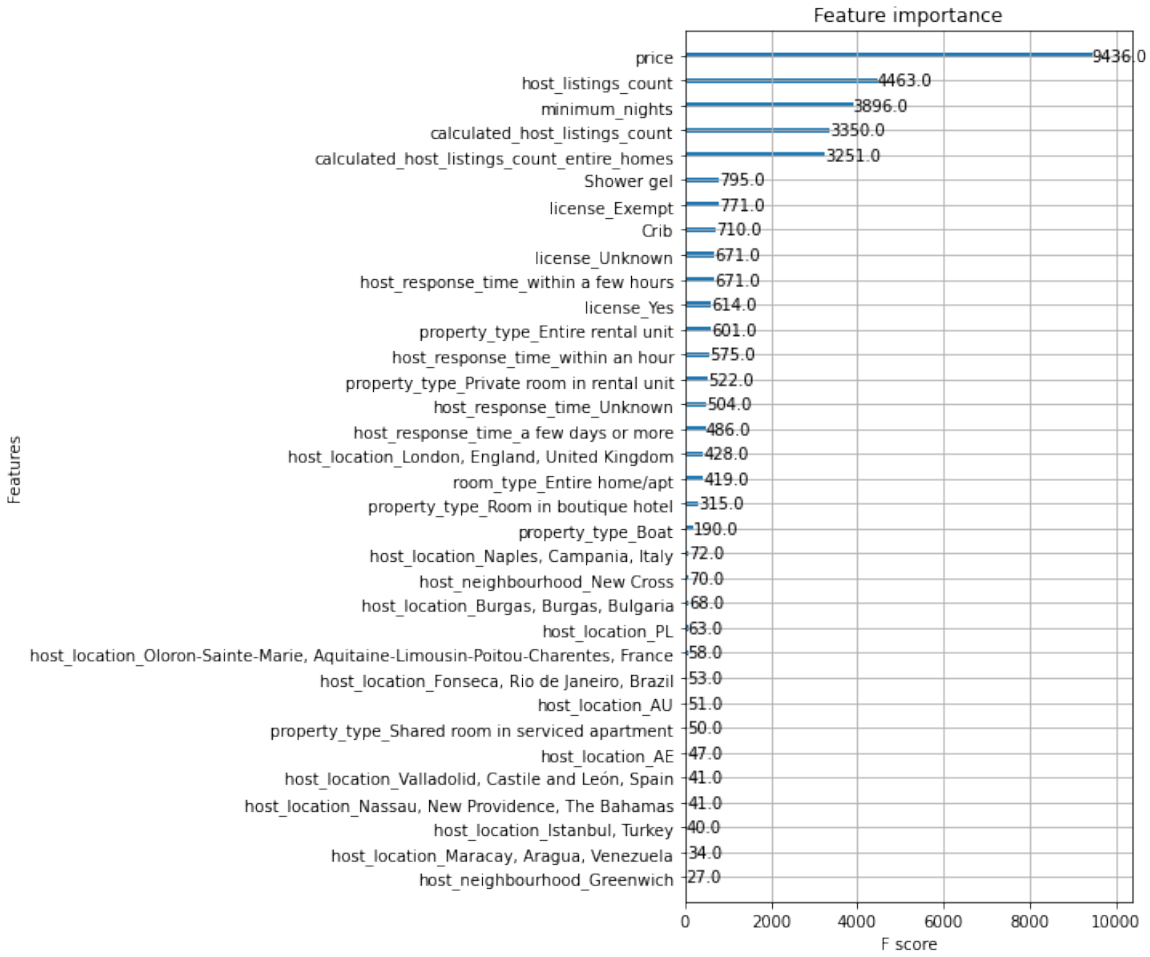

FIGURA 3.18: Variables y su importancia en el modelo para booking_score.

Observamos que la variable más importante y, por mucha diferencia es la variable precio. Vemos también que es importante el número de propiedades que tiene el propietario en alquiler ya que observamos distintas variables que están relacionadas con esto como son *host_listing_count* y todas las variables relacionadas con esta. También esta presente la variable *minimum_nights*, la cual sera interesante para experimentar en la función que se hará en el siguiente apartado. Seguimos con la otra variable respuesta *review_score*.

3.4.2 Para la variable respuesta *review_score*

Iteración	Variables	Mejores Parametros	Train - RMSE	Test - RMSE
1	938	max_depth = 10 learning_rate = 0.05 colsample_bytree = 0.3 n_estimators = 1000	0.211783	0.950134
2	123	max_depth = 8 learning_rate = 0.01 colsample_bytree = 0.35 n_estimators = 1900	0.473304	0.935016
3	81	max_depth = 8 learning_rate = 0.01 colsample_bytree = 0.45 n_estimators = 1700	0.463737	0.932299
4	40	max_depth = 9 learning_rate = 0.01 colsample_bytree = 0.35 n_estimators = 700	0.582629	0.964596
5	28	max_depth = 10 learning_rate = 0.01 colsample_bytree = 0.45 n_estimators = 450	0.616290	0.971504
6	28	max_depth = 9 learning_rate = 0.01 colsample_bytree = 0.35 min_child_weight = 10 min_split_loss = 10 n_estimators = 450	0.864944	0.989903

En este caso observamos algo muy parecido que para la variable anterior, podemos ver como en el modelo inicial tenemos un claro caso de *overfitting*, que ira disminuyendo a medida que reducimos el número de variables. A diferencia que en la variable anterior, para esta añadiremos dos parámetros más que tiene XGBoost, estos son *min child weight* y *min split loss*, estos sirven para reducir el *overfitting* del modelo y debido a que reduciendo las variables habíamos llegado a un punto muerto donde este no se reducía, el uso de estos dos parámetros sera imprescindible. Miramos a continuación las variables del modelo definitivo

y su importancia dentro de este.

FIGURA 3.19: Variables y su importancia en el modelo para review_score.

Podemos ver variables bastante diferentes que en el modelo anterior, ya que por ejemplo la variable precio no tiene importancia en este modelo y a cambio observamos que en este modelo las variables más importantes son el número de reseñas, el número de días que hace desde que es propietario en la plataforma y el porcentaje de aceptación del propietario. Observamos también algunas variables comunes entre los dos modelos, ambos tienen variables de *host_listings_count*, la variable mínimo de noches esta también en ambos, las variables cuna y gel de ducha también y las variables derivadas de la categórica *host_response_time* también las vemos en ambas.

Una vez obtenidos estos dos modelos haremos una función con la que podamos determinar en que ranking esta el piso que queremos estudiar y, nos deje manipular las distintas características de este y así podamos ver cual seria nuestra posición después de estos cambios.

3.5 Aplicación

Esta función esta pensada para ayudar a los distintos propietarios a hacer su vivienda más atractiva para el público. La idea principal de las funciones que vamos a ver a continuación es dejar al propietario experimentar con los atributos de su piso y, ver si esto mejora o empeora su atractivo y que pueda valorar si le sale este cambio a cuenta o no.

Primero de todo tendremos que seleccionar que variables se pueden cambiar, ya que hay algunas que por mucho que el propietario desee no se pueden cambiar, como podrían ser el número de reseñas que tiene la vivienda, el número de viviendas que tiene el propietario o el barrio en el que esta situado. En cambio hay otras que si puede cambiar el propietario, como el precio o algunos de los servicios que este ofrece. Entonces para esto tenemos que hacer una lista de las variables que se pueden intercambiar para cada uno de los dos modelos, obtenemos lo siguiente:

- Variables que se pueden modificar en X y1: price, minimum nights, Shower gel, host response time y license.
- Variables que se pueden modificar en X y2: hosta cceptance rate, minimum nights, Coffe maker, Crib, Shower gel, Wifi, Cleaning products, Wine glasses, reviews, host response time, host is superhost, license y instant bookable.

Una vez determinadas las variables que se pueden cambiar pasamos a la creación del las distintas funciones que necesitaremos para hacer posible la función final. Empezaremos con la función que usaremos para las variables numéricas. Esta función necesitara un data frame y el nombre de la variable.

```
# FUNCION PARA VARIABLES NUMERICAS
def set_numeric_variable_value(series, variable):
    message = "The value of " + variable + " is " + str(series[variable].values[0])
        + " . Set new value (enter to remain the same): "
    new_value_str = input(message)
    if (not new_value_str == ""):
        series[variable] = float(new_value_str)
```

LISTING 3.11: Función set_numeric_variable_value().

Esta función te devuelve el valor actual de la variable y te pregunta por cual lo quieres cambiar, te ofrece también no cambiarlo pulsando la tecla enter.

Seguimos con la función empleada para el cambio de variables categóricas, esta es más compleja que la anterior ya que al haber convertido las variables categóricas en binarias mediante el *one hot encoding*, no es tan sencillo como cambiar un simple valor sino que todos los valores relacionados con esta variable tienen que ser 0. También encontramos un problema en que no todas las variables *dummy* que hay tienen que estar en el modelo, de hecho se experimenta un problema ya que en los dos modelos tenemos una variable de *host_response_time* que no coincide. Para esto haremos uso de los diccionarios de Python. Teniendo en cuenta todo esto creamos la función de la siguiente manera. Esta función necesitara un data frame, el nombre de la variable y un diccionario de las variables categóricas.

```
# FUNCION PARA VARIABLES CATEGORICAS
def set_dummy_variable_value(series, variable, dictionary):
    cont = 0
    index = 0
    for i in range(0,len(dictionary[variable])-1):
        if (dictionary[variable][i] in list(series.columns)):
            if((series[dictionary[variable][i]] == 1).bool()):
                val = dictionary[variable][i]
                cont = 1
        else:
            index = i
    if (cont == 0):
         val = dictionary[variable][index]

    ins = input("The value of " + variable + " is " + val + " : Select which one to
        set to 1: " + str([(i,dictionary[variable][i]) for i in
        range(len(dictionary[variable]))]) + "(enter to remain the same)")

    if (ins != ""):
        for value in dictionary[variable]:
            if (value in list(series.columns)):
                series[value] = 0
        if((series[dictionary[variable][int(ins)]].name in list(series.columns))):
            series[dictionary[variable][int(ins)]] = 1
```

LISTING 3.12: Función set_dummy_variable_value().

Vemos el diccionario que usamos para esta función a continuación.

```
dictionary = {"license" : ["license_Exempt", "license_Unknown", "license_Yes"],
              "host_response_time" : ["host_response_time_Unknown",
                  "host_response_time_a few days or more","host_response_time_within
                  a day", "host_response_time_within a few hours",
                  "host_response_time_within an hour"],
              "instant_bookable" : ["instant_bookable_f", "instant_bookable_t"],
              "host_is_superhost" : ["host_is_superhost_f", "host_is_superhost_t"]}
```

LISTING 3.13: Diccionario para la función set_dummy_variable_value().

Una vez hechas estas dos funciones haremos la función para cambiar estos valores la cual combinara estas dos anteriores. Esta función necesitara dos data frames (para cada uno de los modelos), una lista que contenga las variables que se desean cambiar y un diccionario para las variables categóricas.

```
# FUNCION PARA TRANSFORMAR EL TEST
def test_change(series1, series2, variables, dictionary):
    for name in variables:
        if (name in list(dictionary.keys())):
            if (name in list(series1.columns)):
                set_dummy_variable_value(series1, name, dictionary)
                if (name in list(series2.columns)):
                    series2[name] = series1[name]
            else:
                set_dummy_variable_value(series2, name, dictionary)
        else:
            if (name in list(series1.columns)):
                set_numeric_variable_value(series1, name)
                if (name in list(series2.columns)):
                    series2[name] = series1[name]
            else:
                set_numeric_variable_value(series2,name)
```

LISTING 3.14: Función test_change().

Esta función tiene en cuenta el hecho de que hay variables que se repiten para los dos modelos y guarda el valor en ambos data frames. Una vez tenemos lista esta función, necesitamos una función que nos devuelva la posición el la que nos encontramos. Para esto haremos la siguiente función, la cual necesitara los conjuntos de train y test (en este caso el test es 1 observación) para los dos modelos y la estimación de estos dos, la cual se hace previamente con los parámetros obtenidos en la sección anterior.

```
# FUNCION PARA CALCULAR EL RANKING
def ranking_score(X_y1_train, X_y2_train, X_y1_test, X_y2_test, xg_reg1, xg_reg2):
    pred_train_y1 = xg_reg1.predict(X_y1_train)
    pred_train_y2 = xg_reg2.predict(X_y2_train)
    pred_test_y1 = xg_reg1.predict(X_y1_test)
    pred_test_y2 = xg_reg2.predict(X_y2_test)
    scores_y1 = np.append(pred_train_y1,pred_test_y1)
    scores_y2 = np.append(pred_train_y2,pred_test_y2)
    scores_y1 = list(scores_y1)
    scores_y2 = list(scores_y2)
    scores_y1 = [5 if ele > 5 else ele for ele in scores_y1]
    scores_y1 = [0 if ele < 0 else ele for ele in scores_y1]
    scores_y2 = [5 if ele > 5 else ele for ele in scores_y2]
    scores_y2 = [0 if ele < 0 else ele for ele in scores_y2]
    mixed_score = [x + y for x, y in zip(scores_y1, scores_y2)]
    score_pred = pred_test_y1 + pred_test_y2
    mixed_score.sort(reverse=True)
    rank = (mixed_score.index(score_pred)/len(df)) * 100
    return rank
```

LISTING 3.15: Función ranking_score().

Haremos ahora una pequeña demostración para la observación número 3 de nuestra base de datos. Calculamos el ranking para esta sin realizar ningún cambio en ella y obtenemos 31.5197 %. Le haremos algunos cambios a esta variable, por ejemplo bajaremos el precio de 189 a 185, añadiremos gel de ducha y añadiremos también productos de limpieza. Una vez realizados estos cambio obtenemos un ranking de 20.0573 %, queda visto que al reducir un poco el precio y añadir algunos productos más obtenemos un mejor rendimiento para nuestro piso. Gracias a estas funciones se puede personalizar perfectamente para todos los propietarios y que estos puedan modificar según sus prioridades y valorar si este cambio les resulta suficientemente significativo.

Capítulo 4

Conclusiones

Primero de todo hay que hablar de lo que en este trabajo hemos considerado "atractivo" o "éxito", ya que estas dos no tienen una definición numérica, no existe ningún valor que mida directamente el "atractivo" de algo, con lo que en este estudio se le ha definido como la suma de los días alquilados y el número de reseñas obtenidas pero, se le podría dar otra definición y seria correcta también (siempre esta tenga alguna coherencia con lo que se desea estudiar), ya que se trata de una interpretación subjetiva que cambiara dependiendo de a quien le preguntes. Observaríamos un caso similar en finanzas con el riesgo, ya que este no tiene una definición fija y depende a quien le preguntes puede variar su respuesta, la medida más utilizada es la volatilidad pero no es la única que existe, ya que al igual que con nuestro caso se trata de un concepto subjetivo.

Una vez comentadas las distintas interpretaciones que podemos obtener debido a la subjetividad del concepto que se desea estudiar, hay que comentar la implementación del modelo. Todo el estudio que se ha realizado ha sido utilizando una base de datos que contenía datos de septiembre a diciembre, esto quiere decir que el modelo obtenido no sería práctico para emplearlo ahora ya que el contexto socioeconómico ha variado respecto a el período de tiempo estudiado. Para implementar el modelo correctamente utilizaría solo los datos de 1 mes, por ejemplo junio y utilizaría los datos obtenidos de este para predecir el modelo del junio del año siguiente, realizando algún cambio en algunas de las variables como por ejemplo en la variable precio, a la cual le sumaría la inflación experimentada en el sector inmobiliario ese año. Es sabido por todos que los precios de los alquileres van vinculados al turismo, estos son más caros en periodos de verano o navidades y son más baratos en periodos de "temporada baja", por eso hago especial mención al utilizar bases de datos de un mes específico ya que si utilizamos de tres meses el precio de este podría variar mucho y nosotros no seriamos conscientes de ello. Otra razón para especificar tanto el periodo de tiempo de las bases de datos es que las preferencias de los clientes cambian según el mes, por ejemplo en invierno pueden ser muy buscados los pisos que tengan calefacción y, esto

pasa a ser completamente irrelevante en meses de verano y lo mismo podríamos decir para muchos atributos de cualquier vivienda.

Otro factor que perjudica a este análisis es el de la pandemia a causa del Covid-19, porque hace que varíe mucho la situación entre años según el estado en el que estábamos por la pandemia, con lo que para aplicar este modelo de la mejor manera seria mejor utilizar siempre bases de datos que estén fuera del periodo de pandemia, ya que estas pueden conducir a predecir de manera errónea.

Una vez comentados los posibles errores del modelo y como se tendría que ejecutar este, hay que comentar que una vez obtenidos los distintos modelos, las funciones creadas en el capitulo 3.5, serian validas igualmente, y estas podrían otorgar al propietario la información necesaria para decidir si realizar cambios en su vivienda o simplemente dejar todo como esta, al fin y al cabo el objetivo de este estudio no era predecir con una precisión magistral, sino que poder ayudar a los usuarios de la plataforma a entender mejor a los clientes y entender mejor las ventajas y desventajas de su vivienda.

También serian validos el procedimiento utilizado en el tratamiento de los datos y la selección del modelo aunque estos dos tendrían que ser ajustados para el conjunto de datos que se quiera estudiar. Como hemos visto estos modelos proporcionan una alta fiabilidad ya que son muy precisos, estos implementados con un conjunto de datos correcto como hemos definido anteriormente obtendrían aun mejores resultados y por lo tanto obtendríamos una mayor fiabilidad en las funciones realizadas.

Apéndice A

Repositorio del código

Todo el código empleado para este trabajo y sus distintas bases de datos puede ser consultado en https://github.com/AndreuComp/TFG

Bibliografía

[1] Prashant Banerjee. «A Guide on XGBoost hyperparameters tuning». En: (jul. de 2020). URL: https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning/notebook.

[2] Jason Brownlee. «Data Preparation for Gradient Boosting with XGBoost in Python». En: (ago. de 2016). URL: https://machinelearningmastery.com/data-preparation-gradient-boosting-xgboost-python/.

[3] Jason Brownlee. «kNN Imputation for Missing Values in Machine Learning». En: (jun. de 2020). URL: https://machinelearningmastery.com/knn-imputation-for-missing-values-in-machine-learning/.

[4] Md Didarul Islam y col. «Airbnb rental price modeling based on Latent Dirichlet Allocation and MESF-XGBoost composite model». En: *Machine Learning with Applications* 7 (2022), pág. 100208. URL: https://www.sciencedirect.com/science/article/pii/S2666827021001043.

[5] David Martins. «XGBoost: A Complete Guide to Fine-Tune and Optimize your Model». En: (mayo de 2021). URL: https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663.

[6] Didrik Nielsen. «Tree boosting with xgboost-why does xgboost win "every" machine learning competition?» Tesis de mtría. NTNU, 2016. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2433761/16128_FULLTEXT.pdf.

[7] Manish Pathak. «Using XGBoost in Python Tutorial». En: (nov. de 2019). URL: https://www.datacamp.com/tutorial/xgboost-in-python.

[8] Sean Wang. «Predicting Market Rank for Airbnb Listings». En: (feb. de 2020). URL: https://towardsdatascience.com/predicting-market-rank-for-airbnb-listings-59009a886d6.

[9] Floris Wu. «How to build a heatmap in Python». En: (feb. de 2019). URL: https://www.storybench.org/how-to-build-a-heatmap-in-python/.

[10] Suhang Yao. «Use XGBOOST to Predict the Rental Based on Airbnb Open Data». En: (2022). URL: http://ceur-ws.org/Vol-3150/paper5.pdf.