IMF Smart Education

MÁSTER EN DATA SCIENCIE Y BUSINESS ANALYTICS ONLINE

Análisis en profundidad y Machine Learning para establecer el precio adecuado de tu Airbnb

TFM elaborado por: Eduard Juan Noordermeer Montoya

Tutor/a de TFM: Abel Soriano Márquez

- Sevilla, junio 2024 –

Agradecimientos

Quiero aprovechar este pequeño espacio para dar las gracias a todas las personas que han decidido mantener coexistencia conmigo mientras realizaba el máster y este trabajo de fin de máster.

Gracias a mi señora por su cariño mientras invertía mi tiempo en bases de datos.

Gracias a mis compañeros de trabajo por motivarme en la rutina.

Gracias a el equipo de escalada por ayudarme a subir literal y metafóricamente.

Gracias a Juan Manuel Moreno por su motivación al inicio de este viaje y Abel Soriano Vázquez por su inestimable ayuda para terminarlo.

Y por último muchas gracias a todos los creadores de contenido que me han regalado cientos de horas de podcast para escuchar mientras aprendía a ser un Analista de Datos hecho y derecho.

índice

Contenido

1. Presentación del proyecto 2
1.1 Abstract 2
1.2 Introducción y antecedentes. 2
1.3 Objetivos del proyecto. 3
1.4 Materiales y métodos. 4
2. Importación del dataset y visualización inicial 6
2.1 Importación del Listing 6
2.2 Eliminación inicial de variables 8
3. Transformación de variables 13
3.1 Transformar Precio 13
3.2 Transformar Amenities 14
3.3 Transformar Baños 17
3.4 Convertir variables categóricas a numéricas 20
4. Tratamiento y limpieza de datos 22
5. Establecimiento del target 25
6. Análisis gráfico y correlación de las variables 26
6.1 Boxplot 26
6.2 Histograma de distribuciones 27
6.3 Correlación de Pearson 29
7. Test & Train para Machine Learning. 33
8. Modelos de Machine Learning 34
8.1 Modelo - Regresión Lineal 34
8.2 Modelos - Ridge y Lasso 36
8.3 Modelos - Decision Tree y Random Forest 39
8.4 Modelo - Gradient Boosting 42
8.5 Conclusiones respecto a los modelos 44
9. Estimador de precios 45
9.1 Estimador Manual 45
9.2 Estimador con Widgets 47
9.3 Demostración y resultados. 56
10. Conclusiones 60

1. Presentación del proyecto

1.1 Abstract

En el siguiente proyecto se efectúa un tratamiento y análisis de datos desde una perspectiva de Científico de Datos a fin de crear un estimador de precios preciso y confiable para viviendas de alquiler temporal preciso y confiable.

Los datos para tratar provienen de un amplio dataset con datos obtenidos vía Scraping de la web de Airbnb se realiza una selección, limpieza y tratamiento de datos. Esta obtención exhaustiva de datos representa la oferta disponible de mercado de una forma realista para su correcto análisis.

Durante el mismo se realiza un análisis estadístico y gráfico de las variables, con la intención de encontrar patrones y correlaciones significativas entre las características de las propiedades ofertadas y los precios del alquiler.

Una vez preparados los datos se establece un modelo de estimación basado en técnicas de Machine Learning, donde el modelo se selecciona tras haber analizado y tratado distintos modelos a fin de obtener el algoritmo con resultados más precisos y mejor rendimiento generalizado. El estimador obtenido es capaz de prever el precio del alquiler de una vivienda con una interfaz de usuario accesible, intuitiva y fácil de utilizar, de forma que no sean necesarios conocimientos técnicos para su uso.

Este proyecto ofrece una herramienta valiosa para potenciales arrendatarios y propietarios, demostrando además como la utilidad de técnicas de Machine Learning en el mercado real para la solución de problemas reales.

1.2 Introducción y antecedentes.

Airbnb es una plataforma digital cuya oferta es dar a los particulares la capacidad de publicar y gestionar alojamientos para estancias generalmente a corto y medio plazo, y a su vez a los usuarios arrendadores la capacidad de alquilarlas de una forma sencilla. Desde su fundación en 2008 la empresa ha experimentado un crecimiento exponencial, revolucionando la industria de la hostelería y el turismo. Su modelo de negocio innovador ha transformado la manera en que las personas viajan y se hospedan, al proporcionar una alternativa viable y atractiva a los tradicionales servicios hoteleros.

Si bien existían previamente motores de búsqueda generalizados para el alquiler como Booking, estos se centraban en recopilar ofertas de negocios de hospedaje. El principal atractivo de Airbnb radica en su capacidad para redirigir el turismo desde el hospedaje en hoteles hacia la estancia en habitaciones, apartamentos o casas de particulares, aunque como veremos en los datos analizados, hoy en día recopilan también ofertas hoteleras a fin de abarcar la mayor cuota de mercado posible. Los arrendamientos ofrecidos permiten a los clientes tener una mayor independencia y privacidad que en los hoteles, hostales, pensiones y similares convencionales.

Estas particularidades permiten experiencias más personalizadas para los propietarios, que tienen acceso a una gran cantidad de datos para elegir la mejor ubicación. Ahí radica el potencial para en análisis de datos, y es que al ser pública una gran cantidad de información, podemos discriminar los distintos factores, entendiendo sus dependencias y correlaciones.

Al ser Airbnb accesible por todo particular que quieran sacar un rendimiento a sus propiedades, y para negocios que tienen distintas propiedades para gestionar permite que un gran parque de viviendas esté disponible. Esto suma a la gran cantidad de datos individuales de cada propiedad en alquiler, la gran suma total de propiedades. A mayor cantidad de datos más sencillo es establecer un algoritmo fiable.

En conclusión, Airbnb ofrece una gran cantidad de datos tanto en su totalidad como con respecto a las particulares de cada oferta. Su uso popularizado en todo el mundo y su crecimiento que permite añadir con el tiempo más datos al modelo otorga el contexto perfecto para establecer un modelo vía Machine Learning, que permita a los arrendadores establecer un precio competitivo para su vivienda.

1.3 Objetivos del proyecto.

El objetivo principal es establecer un modelo de Machine Learning capaz de asimilar y analizar los factores que influyen el precio del alquiler de las viviendas de alquiler temporal, a fin de obtener predicciones precisas, y convirtiéndose en una herramienta accesible para el público general, permitiendo a los propietarios estimar el precio adecuado para publicar sus propiedades en plataformas de alquiler. Para ello se tendrán que cumplir una serie de objetivos fundamentales:

1. **Recopilación y preparación de datos satisfactoria de una base de datos de Airbnb.**

Para ello tendremos que obtener una base de datos con un conjunto de datos extenso y variado de la plataforma Airbnb, asegurando que los datos sean representativos y actualizados. Realizaremos una limpieza de datos exhaustiva de limpieza de datos y el tratamiento de las distintas variables, tanto para su procesamiento como su análisis a nivel estadístico.

2. **Obtener las claves para comprender los factores determinantes del precio de alquiler.**

Con el fin de obtener este objetivo se tendrá que completar un análisis de variables, el cual implica Identificar y analizar las variables clave que influyen en el precio de alquiler, como el tamaño de la vivienda, el número de habitaciones y baños, las comodidades ofrecidas o la ubicación. Será necesario llegar un entendimiento profundo de las interacciones de variables, evaluando cómo interactúan entre sí a través de su correlación y cómo afectan colectivamente el precio de alquiler. Estas correlaciones deberán proporcionar información suficiente para discernir las variables realmente relevantes para la implementación del modelo.

3. **Desarrollar del modelo exitoso de Machine Learning:**

Para cumplir este objetivo habrá que seleccionar evaluar distintos algoritmos para seleccionar el más adecuado para los datos planteados. Entrenar un modelo con datos de entrenamiento y prueba, obteniendo resultados plausibles y evitando problemas típicos como sobreajustes. Los datos de evaluación del modelo como el error cuadrático medio y el coeficiente de determinación tendrán que dar una combinación de precisión y robustez aceptable.

4. **Implementar un estimador accesible que proporcione al modelo de Machine Learning los datos necesarios para su predicción.**

Para ello tendremos que desarrollar de la interfaz de usuario entendible y en la que sea sencillo para los usuarios ingresar las características de su propiedad y obtener una estimación del precio de alquiler de manera rápida y sencilla. Esta tendrá que estar correctamente integrada con el modelo de Machine Learning proporcionándole los datos de entradas adecuados para una buena predicción y asegurando que este da un dato de salida acorde.

1.4 Materiales y métodos.

Para elaborar completar los objetivos mentados se utilizará como software principal **Jupyter Notebook**. Este es un entorno de programación en Python que nos permitirá realizar todos los pasos del tratamiento de datos, análisis, visualización y construcción del modelo.

Esto entorno de nutre de distintas **librerías**, la función detallada de las mismas en el contexto del proyecto se desarrollará a medida que se empleen para la resolución. El listado las librerías utilizadas es el siguiente:

- **Pandas:** Tratamiento y análisis de datos estructurados.
- **Numpy**: Operaciones y manejo de arrays multidimensionales.
- **Ast:** Análisis y manipulación de árboles de sintaxis abstracto.
- **Counter** (*Collections*): Contar elementos en colecciones.
- **Re**: Tratamiento de expresiones regulares.
- **Standardscaler** (*sklearn.preprocessing*): Normalización de datos.
- **Matplotlib**: Creación de gráficos y visualizaciones de datos.
- **Seaborn**: Visualización mejorada a partir de matplotlib.
- **Warnings**: Gestión de mensajes de advertencia.
- **Train_test_split** (*sklearn.model_selection*): División de datasets (train and test).
- **LinearRegression** (*sklearn.linear_model*): Establecer una regresión lineal.
- **Mean_squared_error** (*sklearn.metrics*): Cálculo del error cuadrático medio.
- **Ridge** (*sklearn.linear_model*): Establecer regresión lineal de Ridge.
- **Lasso** (*sklearn.linear_model*): Establecer regresión lineal de Lasso.
- **DecisionTreeRegressor** (*sklearn.tree*): Establecer modelo de regresión basado en árboles de decisión.
- **RandomForestRegressor** (*sklearn.ensemble*): Establecer modelo de regresión basado en bosques aleatorios.
- **GradientBoostingRegressor** (*sklearn.ensemble*): Establecer modelo de regresión basado en boosting.
- **Ipywidgets**: Creación de widgets interactivos.
- **Display** (*IPython.display*): Mostrar objetos de salida complejos.
- **HTML** (*IPython.display*): Crear contenido en formato HTML.

En la lista de librerías se pueden apreciar algunas especializadas en establecer modelos. Los modelos que se utilizarán para posteriormente decidir el más adecuado serán los siguientes:

- **Regresión Lineal:**

Se utiliza para predecir valores continuos estableciendo una relación lineal entre una variable dependiente y una o más variables independientes. En nuestro caso, recordamos, la variable dependiente es el precio.

- **Ridge y Lasso:**

El método Ridge utiliza la regresión lineal, pero incluye una penalización L2 para reducir el sobreajuste y manejar la multicolinealidad.
Lasso funciona de forma similar pero su penalización, conocida como L1, puede reducir coeficientes a 0 para eliminar características irrelevantes.

- **Decision Tree y Random Forest:**

El modelo vía Decision Tree divide los datos en subconjuntos según sus características comunes, desarrollando una estructura en forma de árbol de decisiones. Por otro lado, Random Forest conjunta una gran cantidad de árboles de decisión, mejorando el procesado de relaciones complejas y mejorando la precisión con respecto al Decision Tree.

- **Gradient Boosting:**

Crea un modelo que combina múltiples modelos, realizando secuencias que corrigen el modelo anterior, reduciendo los errores y optimizando en algoritmo a medida que itera repeticiones con sus mejoras respectivas

Referente al **Dataset.**
Este proviene de la plataforma Kaggle. Esta es una plataforma online de acceso libre especializada en Ciencia de Datos y aprendizaje automático.
En esta web los usuarios pueden compartir datasets para uso público, siendo un repositorio muy valioso para el análisis de datos.

El directorio de origen se titula “Inside Airbnb USA” y su link para su consulta es el siguiente: “*https://www.kaggle.com/datasets/konradb/inside-airbnb-usa*”.
El usuario que compartió el dataset es Konrad Banachewicz.
Esta dataset ha sido compartido bajo licencia *“ATTRIBUTION 4.0 INTERNATIONAL”*. Esta licencia permite copiar, alterar y distribuir el material en cualquier medio o formato para cualquier propósito, incluido económico.

Este usuario realizó el scraping de datos mediante la herramienta Inside Airbnb, que también presta su uso libre bajo la misma licencia. *“http://insideairbnb.com/get-the-data/”*.

El formato de los datos viene dado en archivos CSV, referentes a 31 localizaciones en Estados Unidos, principalmente estados y sus principales ciudades. Para este trabajo se usarán los estados de Hawái, Los Ángeles, Nueva York, Rhode Island y Seattle.
Las variables presentes se explicarán

- **Resultados.**

Los resultados se presentan en el cuerpo del proyecto, mediante una explicación de la limpieza de datos, estudio de las variables, creación de modelos de machine learning y establecimiento del sistema para introducir los datos. Indexando cada uno de los pasos fundamentales. Se incluirán conclusiones para finalizar el documento.

2. Importación del dataset y visualización inicial

2.1 Importación del Listing

Para realizar el análisis es necesario importar correctamente los archivos que contienen la información relevante para el proyecto. Para ello, establecemos las rutas de archivo (file paths) de cinco archivos CSV que forman el conjunto de datos utilizado. Estos archivos contienen información detallada sobre propiedades en alquiler en diferentes localizaciones.

Para una organización clara y eficiente de los datos, el conjunto de datos importado se ha establecido con el nombre "XY". Aunque el nombre en sí no es crucial, este ayuda a mantener una estructura ordenada y facilita las referencias durante el análisis ya que, dentro de este conjunto de datos, se distinguirán posteriormente los conjuntos de variables:

- Y (Variable de Estudio): Esta es la variable objetivo que queremos predecir, en este caso, el precio de alquiler de las propiedades.
- X (Variables Predictoras): Estas son todas las demás variables que se utilizarán para predecir la variable de estudio. Incluirán el resto de las variables.

En primer lugar, importamos la librería “pandas”, una herramienta fundamental en el tratamiento de datos en formato de DataFrames en Python. Pandas es especialmente útil ya que permite leer y escribir datos en varios formatos, incluyendo CSV, facilitando la manipulación y análisis de grandes volúmenes de información de manera eficiente:

```
import pandas as pd
```

Se establece el file path, ya que los datos están descargados en directorio personal.

```
file_paths = {
    'Hawaii': r'C:\Users\edybl\Desktop\TFM Data\All USA Listings\Hawaii Listings.csv',
    'Los Angeles': r'C:\Users\edybl\Desktop\TFM Data\All USA Listings\Los Angeles Listings.csv',
    'New York City': r'C:\Users\edybl\Desktop\TFM Data\All USA Listings\New York City Listings.csv',
    'Rhode Island': r'C:\Users\edybl\Desktop\TFM Data\All USA Listings\Rhode Island  Listings.csv',
    'Seattle': r'C:\Users\edybl\Desktop\TFM Data\All USA Listings\Seattle Listings.csv', }
```

Se establece una lista para almacenar los dataframes, esto se utilizará para conjuntar los CSVs mencionados.

```
Dataframes = []
```

Ahora podemos cargamos cada uno de los dataframes.

Podemos añadir además una columna que vamos a denominar 'location' y tiene en cuenta el lugar de origen de los datos, que se ha establecido al denominar la ruta.

Al ser archivos muy pesados agregamos low_memory. Este se utiliza para especificar a pandas que puede cargar los archivos de forma completa a expensas de un mayor uso de la memoria.

```
 for location, file_path in file_paths.items():
df = pd.read_csv(file_path, low_memory=False)
df['location'] = location
Dataframes.append(df)
```

Por último, se concatenan los datos para establecer un solo Dataframe con los 5 CSV incluidos.

```
XY = pd.concat(Dataframes, ignore_index=True)
print(XY.head())
```

```
                   id                                        listing_url  \
0  822170245153601161  https://www.airbnb.com/rooms/822170245153601161
1            24382028           https://www.airbnb.com/rooms/24382028
2            19037646           https://www.airbnb.com/rooms/19037646
3             1920293            https://www.airbnb.com/rooms/1920293
4            51204529           https://www.airbnb.com/rooms/51204529

        scrape_id last_scraped           source  \
0  20230316044204   2023-03-17      city scrape
1  20230316044204   2023-03-16      city scrape
2  20230316044204   2023-03-17      city scrape
3  20230316044204   2023-03-17      city scrape
4  20230316044204   2023-03-17  previous scrape

                                          name  \
0  Peaceful Retreat W/Private Lanai & A/C-Aloha Surf
1                                  Rusty Sunrise
2   Great $Value$ | 2 block to Waikiki Beach | KV-6F
3  Pacific Shores-Ocean View Top Floor-A/C All Rooms
4                   private and cozy double bedroom

                                       description  \
0  Relax within a few blocks of Waikiki Beach and...
1  Custom Home Walking Distance to the Grand Hyat...
2  **LOW price, NEWLY renovated condo is your doo...
3  We want you and your family to enjoy our two e...
4  We offer a comfortable double bedroom with a q...

                             neighborhood_overview  \
0                                              NaN
1                                              NaN
2  Location, Location, Location! We are in the he...
3  Kihei is safe...You can stroll the beach on a ...
```

Ilustración 1 - Display Inicial

Se puede apreciar como los datos están en primera instancia bien importados, pero hay muchas variables en el modelo por lo que no se muestran todas. También podemos ver la columna location añadida como última columna del Dataframe. Vamos a analizar mediante “format” las características del dataframe. Esto nos permitirá conocer el número de filas y columnas.

```
print(u'- Número de filas: {}'.format(XY.shape[0]))
print(u'- Número de columnas: {}'.format(XY.shape[1]))
print(u'- Lista de variables: {}'.format(list(XY.columns)))
```

- Número de filas: 127260
- Número de columnas: 76
- Lista de variables: ['id', 'listing_url', 'scrape_id', […]]

El DataFrame resultante de la importación de los archivos CSV contiene una gran cantidad de datos. La calidad y utilidad de estos datos son cruciales para la eficacia del análisis posterior. Un conjunto de datos bien estructurado y preciso proporciona una base sólida para llevar a cabo análisis significativos y obtener conclusiones válidas. Sin embargo, si hay información inexacta, valores nulos, datos faltantes o formatos inconsistentes puede conducir a diagnósticos erróneos y afectar negativamente los resultados del análisis.

Dado que el DataFrame incluye una columna ID única para cada registro, se puede identificar y eliminar registros duplicados. Los duplicados pueden distorsionar los resultados del análisis, por lo que su eliminación es un paso fundamental para asegurar la integridad de los datos

```
duplicated_ids = XY.duplicated(subset=['id']).sum()
print(f"Total de IDs duplicados: {duplicated_ids}")
```

Total de IDs duplicados: 0

Confirmamos que el dataframe no tiene datos duplicados así que procedemos a tratar las variables.

2.2 Eliminación inicial de variables

Al tener el dataframe muchas variables es posible que muchas de ellas no sean útiles para el mismo. La siguiente tabla presenta una descripción muy breve del motivo del descarte tanto por motivos justificativos del modelo como para entender por qué una variable puede no ser relevante por sus características como dato.

- **ID:** Una vez que hemos comprobado que no están duplicados son irrelevantes ya que no nos dan información.
- **Datos del scrap** (*listing_url, scrape_id, last_scraped. source*): Son datos del scrap, nos sirven a los analistas para conocer datos del scraping como el origen de los datos y la fecha, pero no otorga datos para el análisis.
- **Descripciones** *(name, description, neighborhood_overview*): Contienen texto genérico y variable, es difícilmente clasificable ya que usan redacción humana y subjetiva para describir la propiedad.
- **Datos del host** *(picture_url, host_id, host_url, host_name, host_since, host_location, host_about, host_response_time, host_response_rate, host_acceptance_rate, host_is_superhost, host_thumbnail_url, host_picture_url,*

host_neighbourhood, host_listings_count, host_total_listings_count, host_verifications, 'host_has_profile_pic, host_identity_verified, neighbourhood). Son datos sobre el anfitrión, en general no tienen por qué afectar al precio de la vivienda, y la mayoría de ellos el cliente que utilice la aplicación no sabría introducirlos ya que no tiene por qué saber la mayoría de sus características propias o si hará acciones como ponerse una foto de perfil. Estas variables pueden ser útiles para estudiar el tipo de host respecto a la satisfacción para recomendar optimizar por ejemplo el tiempo de respuesta o usar foto de perfil, pero no es definitorio para el precio. En un modelo previo se ha intentado tener en cuenta, por ejemplo el "host response rate" pero su correlación de Pearson con respecto al precio era muy cercana a 0.

- **Coordenadas** (*Latitude, longitude*) Viene dada la ubicación, es una variable que podría ser útil, no obstante, el cliente no es viable utilizarlo para el modelo. El cliente desconoce sus coordenadas para introducirlas, por lo que tendríamos que saber cómo las asigna según la calle y número del cliente, necesitaríamos un mapa con las direcciones y coordenadas en mapa para la transición y que coincidiese como presenta los casos Airbnb.

- **Días de acceso** (*minimum_minimum_nights, maximum_minimum_nights, minimum_maximum_nights, maximum_maximum_nights, minimum_nights_avg_ntm, maximum_nights_avg_ntm*): Son variables respecto a al número máximo y mínimo de noches. Estas se incluyen en el modelo, el resto son datos dependiendo del historial de la vivienda, por lo que los valores básicos de mínimo y máximo de días establecidos por el cliente son suficientes.

- **Disponibilidad** (*calendar_updated, has_availability, availability_30, availability_60, availability_90, availability_365, calendar_last_scrape*). Son los días que estará disponible la vivienda a corto, medio y largo plazo según las reservas ya realizadas. Como el cliente aún no tiene reservas no es un valor que pueda introducir para determinar el precio.

- **Variables características** (*'property_type', 'neighbourhood_cleansed'*) estas variables presentan una categorización mayor respecto al tipo de propiedad y el barrio en el que se encuentra. Son variables muy útiles que implican la introducción de decenas de variables estilo "dummy" en el modelo. Estas se han aplicado en el modelo, pero con la cantidad de datos disponibles actualmente no son beneficiosas para el modelo. Esto es debido a que por ejemplo puede haber un barrio con solo 3 viviendas en él que vicia las predicciones de una forma muy concreta y en general reduce la capacidad predictiva del modelo, presentando un R2 menor al introducirse y reduciendo su capacidad de predicción. En caso de disponer de un dataframe más amplio, por ejemplo, añadiendo nuevas viviendas a lo largo del tiempo mientras se tiene en cuenta la temporalidad, estas variables mejorarían el modelo, pero no es el caso para este proyecto.

- **Reviews**: (*number_of_reviews, number_of_reviews_ltm, number_of_reviews_l30d, first_review, last_review, review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value, license, instant_bookable, calculated_host_listings_count, calculated_host_listings_count_entire_homes, calculated_host_listings_count_private_rooms, calculated_host_listings_count_shared_rooms,'reviews_per_month*). Son datos referentes a la puntuación del sitio, al ser una nueva vivienda aún no ha sido valorada por los usuarios.

```
    columnas_borrar = ['id', 'listing_url', 'scrape_id', 'last_scraped', 'source', 'name', 'description', 'neighborhood_overview', 'picture_url', 'host_id', 'host_url', 'host_name', 'host_since', 'host_location', 'host_about', 'host_response_time', 'host_response_rate', 'host_acceptance_rate', 'host_is_superhost', 'host_thumbnail_url', 'host_picture_url', 'host_neighbourhood', 'host_listings_count', 'host_total_listings_count', 'host_verifications', 'host_has_profile_pic', 'host_identity_verified', 'neighbourhood', 'minimum_minimum_nights', 'maximum_minimum_nights', 'minimum_maximum_nights', 'maximum_maximum_nights', 'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'calendar_updated', 'has_availability', 'availability_30', 'availability_60', 'availability_90', 'availability_365', 'calendar_last_scraped', 'number_of_reviews', 'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_review', 'last_review', 'review_scores_rating', 'review_scores_accuracy', 'review_scores_cleanliness', 'review_scores_checkin', 'review_scores_communication', 'review_scores_location', 'review_scores_value', 'license', 'instant_bookable', 'calculated_host_listings_count', 'calculated_host_listings_count_entire_homes', 'calculated_host_listings_count_private_rooms', 'calculated_host_listings_count_shared_rooms','bathrooms','reviews_per_month','latitude','longitude', 'property_type', 'neighbourhood_cleansed'
```

Se eliminan todas las variables de la lista

```
    XY = XY.drop(columns=columnas_borrar)
```

Verificamos las columnas actuales

```
    print(XY.head())
    print("Columnas después de eliminar:", len(XY.columns))
```

```
  neighbourhood_group_cleansed        room_type  accommodates bathrooms_text  \
0                     Honolulu  Entire home/apt             3         1 bath
1                        Kauai  Entire home/apt            10        4 baths
2                     Honolulu  Entire home/apt             2         1 bath
3                         Maui  Entire home/apt             6        2 baths
4                       Hawaii     Private room             2  1 shared bath

   bedrooms  beds                                          amenities    price  \
0       NaN   2.0  ["Microwave", "Coffee maker", "Hair dryer", "T...  $103.00
1       4.0   4.0  ["Microwave", "Free parking on premises", "Hai...  $542.00
2       NaN   1.0  ["Lockbox", "Free street parking", "TV", "Smok...  $109.00
3       2.0   3.0  ["Baking sheet", "Toaster", "Ethernet connecti...  $265.00
4       1.0   NaN  ["Hair dryer", "Essentials", "Hangers", "Shamp...  $180.00

   minimum_nights  maximum_nights location
0               1             365   Hawaii
1               1            1125   Hawaii
2               2            1125   Hawaii
3               3            1125   Hawaii
4               3            1125   Hawaii
Columnas después de eliminar: 11
```

Ilustración 2 - Lista de variables

Tenemos un total de 11 variables restantes. Vamos a ver que valores presentan y por qué son útiles para el modelo

- **location:** Es el territorio que en el que se han recopilado los datos del dataframe. En este caso son Hawái, Los Ángeles, New York, Rhode Island y Seattle. Estos dataframes no incluyen solo la capital del estado sino que también otras grandes ciudades en estos.
- **neighbourhood_group_cleansed:** Contiene la ciudad concreta dentro de la localización. Es importante puntualizar que en el contexto de estados unidos estas ciudades son colindantes así que presentan distintas características, funcionando de forma similar a barrios. Por ejemplo New York distingue Brooklyn, Staten Island, Bronx, Queens y Manhattan. Todas colindantes y con una gran densidad de población.
- **room_type**: Distingue entre casas completas, habitaciones privadas, habitaciones compartidas o habitaciones de hotel.
- **accommodates:** Es el número de huéspedes para los que está preparada la instalación.
- **bathrooms_text:** Incluye el número y tipo de baños presentes.
- **bedrooms**: Es la suma de habitaciones disponibles.
- **bed**: Son la cantidad de camas que hay el sitio. Hay que puntualizar que no coincide con el número de accommodates ya que puede haber camas dobles en la propiedad.
- **amenities:** Es la lista de comodidades presentes en la instalación.
- **price:** Precio por noche establecido para la vivienda.
- **minimum_nights:** Es la cantidad mínima de noches que se permite el alquiler.
- **maximum_nights:** Es la cantidad máxima de noches que se permite el alquiler.

Revisamos como están categorizadas las variables para su tratamiento.

```
XY_types = XY.dtypes
print(XY_types)
```

```
neighbourhood_group_cleansed     object
room_type                        object
accommodates                      int64
bathrooms_text                   object
bedrooms                        float64
beds                            float64
amenities                        object
price                            object
minimum_nights                    int64
maximum_nights                    int64
location                         object
dtype: object
```

Ilustración 3 - Tipos de variables

Aquí podeos ver el primer indicativo de que es necesario tratar las variables para su uso correcto. Por ejemplo, nuestra variable objetivo, el precio, está catalogada como objeto, por lo tanto, es reconocido como una columna con texto y no numérica.

Revisamos los valores de las variables numéricas a fin de obtener una idea general de los datos posibles e identificar posibles outliers, asunto que trataremos próximamente.

```
XY.describe()
```

	accommodates	bedrooms	beds	minimum_nights	maximum_nights
count	127260.000000	115058.000000	125200.000000	127260.000000	1.272600e+05
mean	3.854220	1.676215	2.123458	15.231966	1.772405e+04
std	2.606521	1.059384	1.535405	27.862121	6.020345e+06
min	0.000000	1.000000	1.000000	1.000000	1.000000e+00
25%	2.000000	1.000000	1.000000	2.000000	6.000000e+01
50%	3.000000	1.000000	2.000000	4.000000	3.650000e+02
75%	5.000000	2.000000	3.000000	30.000000	1.125000e+03
max	16.000000	24.000000	50.000000	1250.000000	2.147484e+09

Ilustración 4 - Valores estadísticos

Miramos los valores de las variables objeto.
Obtenemos la lista de columnas y mostramos los valores únicos para cada una de ellas.

```
obj_cols = XY.select_dtypes(include=['object']).columns
for col in obj_cols:
        print(col, ":", XY[col].unique())
```

```
bathrooms_text : ['1 bath' '4 baths' '2 baths' '1 shared bath' 'Half-bath' '1 private bath'
 '3 baths' '1.5 shared baths' '2.5 baths' '1.5 baths' '5.5 baths'
 '3.5 baths' '0 shared baths' '0 baths' '11 baths' '2 shared baths']
amenities : ['["Microwave", "Coffee maker", "Hair dryer", "TV", "Dedicated workspace", "Mini fridg
e", "Wifi", "Air conditioning", "Iron"]'
 '["Microwave", "Free parking on premises", "Hair dryer", "Dishwasher", "TV with standard cable",
"Dishes and silverware", "Dryer", "Kitchen", "Stove", "Gym", "Refrigerator", "Pool", "Oven", "Esse
ntials", "Bathtub", "Wifi", "Air conditioning", "Patio or balcony", "Washer"]'
 '["Lockbox", "Free street parking", "TV", "Smoke alarm", "Paid parking off premises", "Hangers",
price : ['$103.00' '$542.00' '$109.00' ... '$3,714.00' '$13.00' '$2,384.00']
```

Ilustración 5 – Variables con texto problemático

Hay ciertos datos que llaman la atención.
Encontramos problemas en el precio ya que por el símbolo del dólar está detectando miles de valores de texto aislados.
Los baños vienen dados como texto por una parte número y por otro tipo.
Los amenities son cadenas de texto que concatenan distintas comodidades en la propiedad.

Vamos a tratar estas variables para que sean correctamente utilizables.
Realizamos una primera limpieza de valores nulos antes de tratar las variables. Esto se hace ya que, por ejemplo, podemos querer transformar una variable de texto en numérica, o como dummies pero presentar valores rompen la regla utilizada para su tratamiento.

```
XY.isnull().sum()
```

```
neighbourhood_group_cleansed        0
room_type                           0
accommodates                        0
bathrooms_text                    147
bedrooms                        12202
beds                             2060
amenities                           0
price                               0
minimum_nights                      0
maximum_nights                      0
location                            0
dtype: int64
```

Ilustración 6 - Valores nulos

Podemos confirmar que hay valores nulos, especialmente respecto a los dormitorios. Es importante eliminar los mismos, ya que además el número de dormitorios es una variable fundamental para determinar el precio.

```
XY = XY.dropna()
```

Procedemos a tratar las variables para su correcto uso.

3. Transformación de variables

3.1 Transformar Precio

El precio viene dado como una variable objeto ya que además de la cifra, tiene el símbolo del $ por lo que debemos eliminarlo y establecer la variable como una numérica, en este caso un número enero. Generalmente los precios se dan como números no enteros el tener decimales, pero Airbnb no permite establecer centavos.
Para ello reemplazaremos el símbolo del $ por un valor en blanco, y predeterminaremos el tipo de variable. Para ello utilizamos “np”, una función de la liberaría NumPy que otorga herramientas para el tratamiento y cálculo numérico.

```
import numpy as np
XY['price'] = XY['price'].replace('[\$,]', '', regex=True).astype(float)
XY['price'] = XY['price'].astype(np.int64)
print(XY['price'].head())
```

```
1      542
3      265
7      499
9       96
10     403
Name: price, dtype: int64
```

Ilustración 7 - Precios corregidos

Podemos verificar que se ha establecido el precio correctamente como un numero entero.

3.2 Transformar Amenities

Los "Amenities" son las comodidades y servicios adicionales que una propiedad ofrece a sus huéspedes, como wifi, aire acondicionado, calefacción, utensilios de cocina, entre otros. Estas comodidades pueden tener un impacto significativo en el precio de alquiler de una propiedad, ya que añaden valor y mejoran la experiencia del huésped. Por lo tanto, es esencial analizar esta variable en profundidad para comprender su influencia en los precios.

El principal desafío al tratar la variable "Amenities" es la amplia variedad de posibles valores. Cada propiedad puede ofrecer una combinación única de comodidades, y la lista de todas las comodidades disponibles es extensa. Además, los datos suelen estar almacenados como una concatenación de texto en una única columna, lo que complica su análisis directo.

Utilizaremos los módulos “ast” y “counter” para facilitar el procesamiento de los datos. Ast permite trabajar con árboles de sintaxis abstracta por lo que puede descomponer la lista de amenities. Por otro lado counter permite contar todos los elementos, en este caso cada una de las amenities independientes en la línea de texto.

```
import ast
from collections import Counter
```

Convertimos la cadena de texto en una lista y la aplicamos a la variable amenities.

```
def parse_amenities(amenities_str):
return ast.literal_eval(amenities_str)
XY['amenities'] = XY['amenities'].apply(parse_amenities)
```

Obtenemos una lista de todas las amenities contando el número de veces que aparece cada una.

```
all_amenities = [item for sublist in XY['amenities'] for item in sublist]
amenities_counter = Counter(all_amenities)
```

Establecemos el conteo como dataframe y lo ordenamos de forma descendente para ver los valores que más aparecen.

```
amenities_df = pd.DataFrame(amenities_counter.items(), columns=['Amenity', 'Count'])
amenities_df = amenities_df.sort_values(by='Count', ascending=False).reset_index(drop=True)
print("Conteo de todas las amenities:")
print(amenities_df)
```

```
Conteo de todas las amenities:
                                                 Amenity   Count
0                                            Smoke alarm  104416
1                                                   Wifi  103942
2                                                Kitchen  101890
3                                              Essentials   96227
4                                                 Hangers   86601
...                                                   ...     ...
20102  55" HDTV with HBO Max, Apple TV, Hulu, Disney+...       1
20103                    65" TV with Roku, Netflix, Hulu       1
20104               Smeg stainless steel induction stove       1
20105  65" HDTV with Roku, Amazon Prime Video, Disney...       1
20106  HDTV with Amazon Prime Video, HBO Max, Hulu, N...       1

[20107 rows x 2 columns]
```

Ilustración 8 - Tipos de amenities

Podemos confirmar que existen más de 20107 valores independientes únicos. La mayoría de estos valores son individuales, es decir, son comodidades que aparecen en muy pocas propiedades. Este elevado número de valores únicos presenta un desafío para el análisis, ya que incluir todas las comodidades en su forma original podría sobrecargar el modelo y dificultar la interpretación de los resultados.

Como solución se ha tomado dos aproximaciones al tratamiento de estos datos:

Por un lado, identificar y seleccionar los amenities más comunes. Se analizará la frecuencia de cada amenity para identificar las más comunes, las cuales son también fácilmente reconocibles por los clientes. Solo las amenities que aparecen con mayor frecuencia serán seleccionadas como variables individuales en el modelo. Esto permite reducir significativamente el número de variables, manteniendo a su vez una gran parte de la información relevante.

Por otro lado, el cálculo total de Amenities ya que para no perder la información proporcionada por las amenities menos comunes podemos discernir su relevancia en el mero hecho de estar presentes en el total de datos. Este enfoque permite capturar el valor agregado de tener más comodidades, sin necesidad de analizar cada una individualmente.

Este tratamiento permite reducir la complejidad del modelo mientras preservamos la información relevante del mismo.

Establecemos las 20 amenities más frecuentes. Los clientes podrán seleccionar individualmente cuales se encuentras en la propiedad y cuales no.

```
top_20_amenities = [amenity for amenity, count in
amenities_counter.most_common(20)]
for amenity in top_20_amenities:
count = amenities_counter[amenity]
print(f"{amenity}: {count} veces")
```

```
Smoke alarm: 104416 veces
Wifi: 103942 veces
Kitchen: 101890 veces
Essentials: 96227 veces
Hangers: 86601 veces
Carbon monoxide alarm: 84343 veces
Hair dryer: 84262 veces
Iron: 79725 veces
Hot water: 79415 veces
Dishes and silverware: 76350 veces
Refrigerator: 74031 veces
Shampoo: 72698 veces
Microwave: 70914 veces
Cooking basics: 68329 veces
Bed linens: 67194 veces
Fire extinguisher: 66464 veces
Air conditioning: 65226 veces
Heating: 61946 veces
First aid kit: 55835 veces
Self check-in: 55505 veces
```

Ilustración 9 - Amenities más frecuentes

Para dejar estas variables correctamente establecidas sin necesitar más tratamiento posterior, vamos a establecerlas como dummies, de forma que si la vivienda presenta la comodidad presente un 1 y en caso de no disponer de ella un 0.

```
for amenity in  top_20_amenities:
  XY[amenity] =
        XY['amenities'].apply(lambda
        x: 1 if amenity in x else 0)
        for amenity in
top_20_amenities:
          print(f"{amenity}:")

print(XY[amenity].head())
          print()
```

```
Smoke alarm:
1     0
3     1
7     0
9     1
10    1
Name: Smoke alarm, dtype: int64

Wifi:
1     1
3     1
7     1
9     1
10    0
Name: Wifi, dtype: int64
```

Ilustración 10 - Dummies aplicados

Ahora vamos a crear la variable del total de amenities, separando el texto por su delimitante y contando el número de valores

Creamos la columna 'total_amenities' que contiene el número total de amenities para cada fila, obtenido mediante len.

```
XY['total_amenities'] = XY['amenities'].apply(len)
print(XY['total_amenities'])
```

```
1          19
3          65
7          12
9          16
10         11
           ..
127255     54
127256     41
127257     44
127258     11
127259     51
Name: total_amenities, Length: 113382, dtype: int64
```

Ilustración 11 - Establecido el número total de amenities con éxito

Ahora tenemos las variables más comunes como dummies y la suma de amenities totales como valor numérico, por lo que podemos eliminar la columna original de amenities que es puro texto sin tratamiento.

Podemos eliminar la columna original amenities ya que como texto sin tratar ya no tiene ningún valor para el modelo.

```
XY = XY.drop('amenities', axis=1)
```

3.3 Transformar Baños

Estudiamos los posibles valores para su tratamiento.

```
valores_posibles = XY['bathrooms_text'].unique()
print("Valores posibles de bathrooms_text:")
print(valores_posibles)
```

```
Valores posibles de bathrooms_text:
['4 baths' '2 baths' '1 bath' '3 baths' '1.5 shared baths' '2.5 baths'
 '1.5 baths' '5.5 baths' '3.5 baths' '1 shared bath' '0 shared baths'
 'Half-bath' '1 private bath' '0 baths' '11 baths' '2 shared baths'
 '4.5 baths' '4 shared baths' '6 baths' '6.5 baths' '3 shared baths'
 '4.5 shared baths' '5 baths' '7.5 baths' '8 baths' '14 baths' '13 baths'
 '3.5 shared baths' 'Shared half-bath' '2.5 shared baths' '8.5 baths'
 '7 baths' '6.5 shared baths' 'Private half-bath' '9 baths' '22 baths'
 '10 shared baths' '10 baths' '9.5 baths' '10.5 baths' '12 baths'
 '8 shared baths' '11 shared baths' '11.5 baths' '5 shared baths'
 '8.5 shared baths' '5.5 shared baths' '21.5 baths' '13.5 baths'
 '21 baths' '20 baths' '12.5 baths' '15 baths' '11.5 shared baths'
 '6 shared baths' '15.5 baths' '16 shared baths']
```

Ilustración 12- Posibles valores de baños

La variable baños se presenta como número con texto. En general nos podemos referir a:

- bath: Baño genérico, sin concretar.

- shared bath: Si se comparte con más huéspedes.

- Private bath: Si es exclusivo para el arrendador.

- half-bath: Cuando es un baño simple, generalmente pequeño con solo inodoro y pileta.

Esto explica la existencia de valores decimales, ya que en Airbnb los half bath toman un valor decimal de 0.5.

Podemos pues dividir por la variable para obtener dos nuevas. Por un lado una variable objeto categórica que distinga el tipo de baño o baños y por otra el número total de los mismos.

Importamos re que funciona para buscar patrones en cadenas de texto.

```
import re
```

Definimos la función para extraer por un lado el texto y por otro el número.

Encontramos números en el texto, los separamos del mismo y eliminamos los números en el texto.

```
def extraer_numero_baños(texto):
    numero = re.findall(r'(\d+(\.\d+)?)', texto)
    num_baños = float(numero[0][0]) if numero else 1
    texto_limpio = re.sub(r'\d+(\.\d+)?', '', texto)
    return num_baños, texto_limpio
```

Aplicamos la función definida a la columna, distinguiendo el número de baños y el texto ya limpiado.

Eliminamos la columna de *bathrooms_text* ya que el texto sin tratar no nos es de interés.

```
XY['num_bathrooms'], XY['bathrooms_text_clean'] =
zip(*XY['bathrooms_text'].apply(extraer_numero_baños))
XY.drop(columns=['bathrooms_text'], inplace=True)
print(XY[['num_bathrooms', 'bathrooms_text_clean']])
```

```
        num_bathrooms bathrooms_text_clean
1                 4.0                baths
3                 2.0                baths
7                 2.0                baths
9                 1.0                 bath
10                3.0                baths
...               ...                  ...
127255            1.0                 bath
127256            1.0          shared bath
127257            1.0                 bath
127258            1.0                 bath
127259            1.5                baths

[113382 rows x 2 columns]
```

Ilustración 13 - Nuevas variables de baños

Hemos creado una nueva variable numérica y objeto. Es importante revisar que categorías existen ahora como objeto para asegurarnos de que son correctas y no crear dummies de más próximamente. Para ello recopilamos los valores únicos.

```
valores_unicos = XY['bathrooms_text_clean'].unique()
print(valores_unicos)
```

```
[' baths' ' bath' ' shared baths' ' shared bath' 'Half-bath'
 ' private bath' 'Shared half-bath' 'Private half-bath']
```

Ilustración 14 - Valores de texto (Baños)

Como podemos apreciar el texto distingue en singlar y plural el tipo de baño. Esto no es necesario ya que el número de baños viene establecido por la nueva variable que los cuantifica, por lo que vamos a establecer las categorías en plural como su equivalente en singlar.

```
XY['bathrooms_text_clean'] = XY['bathrooms_text_clean'].replace({' shared baths': ' baths'})
XY['bathrooms_text_clean'] = XY['bathrooms_text_clean'].replace({' baths': ' bath'})
valores_unicos_actualizados = XY['bathrooms_text_clean'].unique()
print(valores_unicos_actualizados)
```

```
[' bath' ' shared bath' 'Half-bath' ' private bath' 'Shared half-bath'
 'Private half-bath']
```

Ilustración 15 - Valores de texto tratados (Baños)

Confirmamos que los valores categóricos posibles para los baños han sido correctamente actualizados.

3.4 Convertir variables categóricas a numéricas

El entrenamiento de modelos de aprendizaje automático generalmente es necesario que las variables utilizadas sean de tipo numérico. Esto es debido a que la mayoría de los algoritmos de Machine Learning, entre ellos los 4 modelos que se emplearán en este proyecto, están diseñados para procesar entradas numéricas. No procesan datos categóricos del tipo “object”, es decir, entradas de texto. Esto implica que las variables tendrán que transformarse para que sean entendibles para el modelo dándoles un valor numérico.

Hay distintas técnicas para realizar esta transformación. Para entender el razonamiento detrás de la decisión de utilizar un modo u otro vamos a plantear las distintas opciones más populares:

- **Label Encoding** establece un número para cada categoría, generalmente dando orden de tiempo o jerárquico en las variables.
- **One-Hot Encoding** crea una columna nueva para cada categoría posible en una variable. Se le asigna un valor binario de 0 a 1, significando 0 la inexistencia de la característica y 1 la existencia de este. Esta transformación no otorga jerarquía a los datos, pero a cambio aumenta la dimensión de los datos y son más difíciles de procesar y comprender para varios modelos de Machine Learning.
- **Target Encoding** otorga un valor a la variable según su media de aparición con respecto a una variable objetivo.
- **Frequency Encoding** sustituye cada una de las categorías por su frecuencia, es decir le da un valor numérico que es igual al número de veces que aparece la variable entre los datos. Es un sistema sencillo y polivalente pero no es bueno capturando relaciones complejas.

En primer lugar, vamos a analizar las posibles categorías de las variables objeto. Establecemos una lista de estas para su análisis, mirando sus posibles valores independientes.

Según las categorías de los datos es necesario decidir que tipo de transformación es más útil para obtener la información más útil posible sin sobrecargar la capacidad del modelo.

```
object_columns = [ 'neighbourhood_group_cleansed', 'room_type',
'bathrooms_text_clean', 'location']
for column in object_columns:
    print(f"Valores únicos en '{column}':")
    print(XY[column].unique())
    print("\n")
```

```
Valores únicos en 'neighbourhood_group_cleansed':
['Kauai' 'Maui' 'Honolulu' 'Hawaii' 'Unincorporated Areas' 'Other Cities'
 'City of Los Angeles' 'Brooklyn' 'Staten Island' 'Bronx' 'Queens'
 'Manhattan' 'Providence' 'Newport' 'Kent' 'Washington' 'Bristol'
 'Other neighborhoods' 'West Seattle' 'Ballard' 'Cascade' 'Capitol Hill'
 'Queen Anne' 'Rainier Valley' 'Magnolia' 'Beacon Hill' 'Downtown'
 'Lake City' 'Central Area' 'University District' 'Delridge' 'Northgate'
 'Interbay' 'Seward Park']


Valores únicos en 'room_type':
['Entire home/apt' 'Private room' 'Shared room' 'Hotel room']


Valores únicos en 'bathrooms_text_clean':
[' bath' ' shared bath' 'Half-bath' ' private bath' 'Shared half-bath'
 'Private half-bath']


Valores únicos en 'location':
['Hawaii' 'Los Angeles' 'New York City' 'Rhode Island' 'Seattle']
```

Ilustración 16 - Valores de las variables dummies

Podemos apreciar como los valores tienen sentido. Dado que el objetivo del modelo es obtener el efecto en la variable precio de muchas variables, prevemos complejidad. A su vez, es difícil justificar una jerarquía en las distintas variables.

Calificar los barrios y la localización de forma jerárquica sería inadecuado, esto podría hacerse si en un análisis ajeno pudiésemos calificar los barrios bajo un modelo de criterios con un nivel de calidad, pero no es adecuado para este modelo.

A su vez el tipo de habitación y baño respecto a su tamaño o consideración general social. No obstante, en estos casos también es subjetivo, ya que es difícil evaluar si es mejor una habitación de hotel o una habitación compartida. Además, una habitación técnicamente se pude compartir. Al igual una persona puede valorar mejor un medio baño privado que un baño entero compartido.

Dado que en la limpieza de datos hemos eliminado las variables categóricas con un número muy elevado de posibles variables para no saturar el sistema ni añadir en él variables con poca representatividad en los datos, podemos aplicar One-Hot Encoding a las 4 variables presentes sin miedo a crear un gran número de nuevas variables.

Es importante puntualizar que los valores de neighborhood_group_cleansed dependen del valor previo de location, ya que cada uno es exclusivo de una localización concreta. Esto se tendrá en cuenta en la ejecución del modelo.

Tomamos pues dummies. Esto creará una nueva variable para cada posible categoría y establecerá el valor 1 si está presente y el valor 0 si no. Estas son exclusivas las unas de las otras por lo que un dato solo tendrá un valor para un tipo de neighbourhood, room type, bathroom y location.

```
columnas_dummy = ['neighbourhood_group_cleansed', 'room_type',
'bathrooms_text_clean', 'location']
XY = pd.get_dummies(XY, columns=columnas_dummy, drop_first=False)
XY.replace({False: 0, True: 1}, inplace=True)
print(XY.head())
```

Revisamos el tipo de variables, ahora hay un total de 77 debido a la inclusión de las variables dummy, tanto las recientemente creadas como las correspondientes a los 20 amenities más frecuentes creados previamente.
Ya no hay variables objeto, todas tienen el formato int64 o float64, es decir numérico, ya sean números enteros o no.
Podemos echar un nuevo vistazo a los datos generales. Podemos repasar los valores dados y a su vez confirmar que se muestran el total de 77 variables.

4. Tratamiento y limpieza de datos

A continuación, vamos a visualizar los datos numéricos generales del dataset. Esta visualización nos permitirá obtener una idea general de los valores presentes en las distintas variables y detectar posibles anomalías o outliers que puedan afectar nuestro análisis y los resultados del modelo predictivo.

La naturaleza de nuestro dataset implica que los valores anómalos sean relativamente poco comunes debido a que muchos de los valores son continuos y se encuentran dentro de un rango establecido. Además, una de la ventaja en la organización de los datos es que la propia introducción de datos para crear un anuncio en Airbnb está limitada para evitar valores anómalos u irrealistas que puedan tanto confundir al arrendador que mire al anuncio como perjudicar a los propios filtros de búsqueda de la aplicación. Esto hace que, si bien el tratamiento y limpieza de datos sigue siendo fundamental, en primera instancia la eliminación de datos anómalos, al ser reducidos, no implique perder una gran cantidad de datos en el dataset.

Por ejemplo, hay variables con un rango establecido. Variables como baños (bathrooms) y camas (beds) tienen un rango mínimo bien definido. No esperamos encontrar números negativos en estas variables. El número mínimo para los baños podría establecerse en 0, mientras que, para las camas, el mínimo razonable es 1.

Sin embargo, hay variables en nuestro dataset que no tienen un suelo y techo claro, lo que hace que sea más probable encontrar valores anómalos. Un ejemplo clave es nuestra variable de estudio, el precio de alquiler (price), que puede presentar valores anómalos tanto muy elevados como muy reducidos.

Es posible que algunos arrendadores establezcan precios mínimos irrealistas, como 1 unidad monetaria, para promocionar el arrendamiento, aunque este no sea el precio real ni esté disponible por este mismo en la práctica. En ocasiones, por ejemplo, se promocionan así para aparecer los primeros en un filtrado por precios para después pactar un precio en privado.
De manera similar, algunos arrendadores pueden establecer precios máximos extremadamente altos, como 999,999 unidades monetarias, para mantener la antigüedad de la publicación y evitar que la propiedad se alquile, debido a razones relacionadas con el algoritmo de la plataforma.

Estos valores extremos deben ser identificados y tratados adecuadamente para evitar que distorsionen los resultados del modelo predictivo.

Otras variables que pueden presentar outliers significativos son aquellas relacionadas con la capacidad de alojamiento y la duración de la estancia.

Los accommodates, que son la capacidad de alojamiento, aunque tienen como número mínimo de personas 1, su número máximo puede presentar valores inusualmente altos. En nuestra visión general de los valores estadísticos números, observamos que el valor máximo para "accommodates" es 16, lo cual puede ser razonable para propiedades grandes o villas.

Por otro lado, para las variables bedrooms, beds, minimum_nights y maximum_nights encontramos outliers en los valores techo. En los valores mínimo no tiene sentido eliminar los valores más pequeños ya que una sola cama o noche es un valor realista, no obstante, en la visión general hemos podido ver como en contraste, mientras que el número máximo de accomodates era 16, el número máximo de camas es 50. Esto es un indicativo de posibles valores o bien incorrectos, o bien marginales que no sirven para una predicción general realista.

Para asegurar la calidad y fiabilidad de nuestro análisis y modelo predictivo, implementaremos realizamos un tratamiento de outliers en el que eliminamos los valores más reducidos (en las que hemos mencionado, es necesario) y elevados evitando cifras anómalas.

Posteriormente verificaremos de forma gráfica si siguen existiendo outliers que puedan empeorar la calidad del modelo.

Establecemos el decil inferior del precio y el decil tanto inferior como superior del resto de variables.

```
decil_inferior_price = XY['price'].quantile(0.1)
decil_superior = XY[['price', 'accommodates', 'bedrooms', 'beds', 'minimum_nights', 'maximum_nights', 'num_bathrooms']].quantile(0.9)
```

Contamos la cantidad de datos presentes en cada uno.

```
decil_inferior_contar = (XY['price'] < decil_inferior_price).sum()
decil_superior_contar = (XY[['price', 'accommodates', 'bedrooms', 'beds', 'minimum_nights', 'maximum_nights', 'num_bathrooms']] > decil_superior).sum()
total_decil_superior_contar = decil_superior_contar.sum()
XY_prefiltro = XY.copy()
```

Eliminamos los valores.

```
XY = XY[(XY['price'] >= decil_inferior_price) &
        (XY['price'] <= decil_superior['price']) &
        (XY['accommodates'] <= decil_superior['accommodates']) &
        (XY['bedrooms'] <= decil_superior['bedrooms']) &
        (XY['beds'] <= decil_superior['beds']) &
        (XY['minimum_nights'] <= decil_superior['minimum_nights']) &
        (XY['maximum_nights'] <= decil_superior['maximum_nights']) &
        (XY['num_bathrooms'] <= decil_superior['num_bathrooms'])]
```

Contamos el total de valores eliminados.

```
valores_eliminados = len(XY_prefiltro) - len(XY)
print(f'Total de valores eliminados: {valores_eliminados}')
```

Total de valores eliminados: 35852

El dataframe queda tratado, esta es su forma general.

```
print(u'- Número de filas: {}'.format(XY.shape[0]))
print(u'- Número de columnas: {}'.format(XY.shape[1]))
print(u'- Lista de variables: {}'.format(list(XY.columns)))
XY[:2]
```

```
- Número de filas: 77530
- Número de columnas: 77
- Lista de variables: ['accommodates', 'bedrooms', 'beds', 'price', 'minimum_nights', 'maximum_nights', 'Smoke alarm', 'Wifi', 'Kitchen', 'Essentials',
'Hangers', 'Carbon monoxide alarm', 'Hair dryer', 'Iron', 'Hot water', 'Dishes and silverware', 'Refrigerator', 'Shampoo', 'Microwave', 'Cooking basics',
'Bed linens', 'Fire extinguisher', 'Air conditioning', 'Heating', 'First aid kit', 'Self check-in', 'total_amenities', 'num_bathrooms', 'neighbourhood_gr
oup_cleansed_Ballard', 'neighbourhood_group_cleansed_Beacon Hill', 'neighbourhood_group_cleansed_Bristol', 'neighbourhood_group_cleansed_Bronx', 'neighbo
urhood_group_cleansed_Brooklyn', 'neighbourhood_group_cleansed_Capitol Hill', 'neighbourhood_group_cleansed_Cascade', 'neighbourhood_group_cleansed_Centr
al Area', 'neighbourhood_group_cleansed_City of Los Angeles', 'neighbourhood_group_cleansed_Delridge', 'neighbourhood_group_cleansed_Downtown', 'neighbou
rhood_group_cleansed_Hawaii', 'neighbourhood_group_cleansed_Honolulu', 'neighbourhood_group_cleansed_Interbay', 'neighbourhood_group_cleansed_Kauai', 'ne
ighbourhood_group_cleansed_Kent', 'neighbourhood_group_cleansed_Lake City', 'neighbourhood_group_cleansed_Magnolia', 'neighbourhood_group_cleansed_Manhat
tan', 'neighbourhood_group_cleansed_Maui', 'neighbourhood_group_cleansed_Newport', 'neighbourhood_group_cleansed_Northgate', 'neighbourhood_group_cleanse
d_Other Cities', 'neighbourhood_group_cleansed_Other neighborhoods', 'neighbourhood_group_cleansed_Providence', 'neighbourhood_group_cleansed_Queen Ann
e', 'neighbourhood_group_cleansed_Queens', 'neighbourhood_group_cleansed_Rainier Valley', 'neighbourhood_group_cleansed_Seward Park', 'neighbourhood_grou
p_cleansed_Staten Island', 'neighbourhood_group_cleansed_Unincorporated Areas', 'neighbourhood_group_cleansed_University District', 'neighbourhood_group_
cleansed_Washington', 'neighbourhood_group_cleansed_West Seattle', 'room_type_Entire home/apt', 'room_type_Hotel room', 'room_type_Private room', 'room_t
ype_Shared room', 'bathrooms_text_clean_ bath', 'bathrooms_text_clean_ private bath', 'bathrooms_text_clean_ shared bath', 'bathrooms_text_clean_Half-bat
h', 'bathrooms_text_clean_Private half-bath', 'bathrooms_text_clean_Shared half-bath', 'location_Hawaii', 'location_Los Angeles', 'location_New York Cit
y', 'location_Rhode Island', 'location_Seattle']
```

	accommodates	bedrooms	beds	price	minimum_nights	maximum_nights	Smoke alarm	Wifi	Kitchen	Essentials	...	bathrooms_text_clean_ private bath	bathrooms_text_clean_ shared bath	bath
3	6	2.0	3.0	265	3	1125	1	1	1	1	...	0	0	
9	2	1.0	1.0	96	1	1125	1	1	1	1	...	0	0	

Ilustración 17 - Contexto general del Dataframe actual

5. Establecimiento del target

Ahora que hemos tratado y limpiado nuestro dataset, podemos definir la variable objetivo (target) que deseamos que nuestro modelo de Machine Learning prediga. En nuestro caso, esta variable objetivo es el precio de alquiler, representada en el dataset como "price".

Establecer una variable como target, de forma explícita, ayuda al modelo de Machine Learning a definir el objeto a predecir y centrarse en entender las relaciones de las variables del modelo con la variable concreta.

De la misma manera, estos modelos pueden presentar resultados que permiten evaluar su calidad y tener una variable target definida, permite establecer y comparar características como los valores evaluativos de la calidad del modelo como el error cuadrático medio del modelo.

A partir de este punto, estructuraremos nuestro dataset XY de manera que quede claramente dividido en dos partes: X e Y. La parte X contendrá el conjunto de variables predictoras, mientras que Y representará la variable objetivo que deseamos predecir.

Extraemos la columna "price" del DataFrame XY y la almacenamos en un nuevo DataFrame llamado Y.
Esta acción nos permite tener un DataFrame dedicado exclusivamente a la variable objetivo, facilitando así el proceso de modelado y evaluación.

Eliminamos la columna "price" de XY. Esto asegura que nuestro DataFrame X contenga solo las variables predictoras, eliminando cualquier posible fuga de información durante el entrenamiento del modelo.

```
XY['target'] = XY['price']
XY.drop(['price'], axis=1, inplace=True)
X = XY.drop('target', axis=1)
Y = XY['target']
```

Al realizar esta operación, nos quedamos con un DataFrame limpio y estructurado que podemos utilizar para alimentar nuestro modelo de Machine Learning.

También creamos una versión normalizada del dataframe X para su posterior uso en el análisis gráfico. Normalizar la variable establece los valores en una misma escala entre 0 y 1, lo que facilita comparar valores numéricos.

```
X_normalizado = (X-X.mean())/X.std()
```

6. Análisis gráfico y correlación de las variables

6.1 Boxplot

En primer lugar, vamos a proceder con la visualización de un boxplot de los valores numéricos en nuestro modelo. Esta visualización nos ayudará a identificar si aún existen valores anómalos que puedan afectar la precisión y robustez de nuestro modelo. La detección y tratamiento de outliers es esencial para asegurar que nuestro modelo de Machine Learning no se vea influenciado negativamente por datos atípicos.

Para facilitar la comparación de las variables y asegurar que todas estén en una escala común, utilizaremos StandardScaler del módulo sklearn.preprocessing. La normalización ajusta las variables para que tengan una media de 0 y una desviación estándar de 1 y ayuda a su comparación, especialmente a nivel visual.

Para visualizar los resultados de la normalización y detectar posibles valores anómalos, utilizaremos las librerías matplotlib.pyplot y seaborn. Tal y como se ha mencionado en la lista de métodos para el proyecto, son librerías para la creación de gráficos.

Los boxplots son una herramienta útil para visualizar la distribución de las variables numéricas y detectar outliers. A continuación, mostramos cómo crear un boxplot de las variables normalizadas:

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

Establecemos las columnas numéricas

```
columnas_numericas = ['accommodates', 'bedrooms', 'beds', 'minimum_nights',
'maximum_nights', 'total_amenities', 'num_bathrooms', 'target']
XY_seleccion = XY[columnas_numericas].
```

Aplicamos la normalización de datos.

```
scaler = StandardScaler()
XY_normalizado = scaler.fit_transform(XY_seleccion)
XY_normalizado = pd.DataFrame(XY_normalizado,
columns=columnas_numericas)
```

Creamos el gráfico de cajas

```
plt.figure(figsize=(15,7))
ax = sns.boxplot(data=XY_normalizado)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.title(u'Boxplot con las variables numéricas normalizadas')
plt.ylabel('Valor')
plt.show()
```

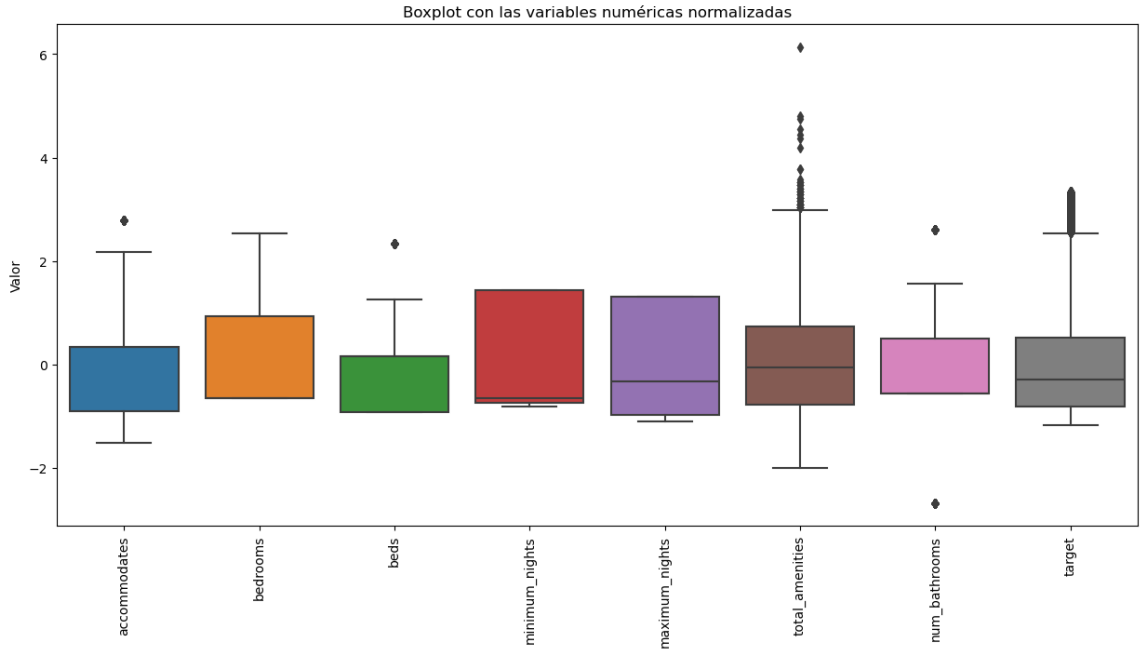

Ilustración 18 – Boxplot

El boxplot nos proporciona una visualización clara y efectiva que nos permite confirmar que el número de outliers presentes en nuestro conjunto de datos es reducido y residual. Esta herramienta gráfica nos facilita la identificación visual de cualquier valor atípico, mostrando que la mayoría de los datos se encuentran dentro de los parámetros esperados y formando un bloque consistente. La ausencia de numerosos outliers sugiere que nuestros procesos de limpieza y normalización de datos han sido exitosos y eficientes.

6.2 Histograma de distribuciones

Después de haber comprobado que la eliminación de outliers ha sido efectiva y que tiene sentido en nuestro análisis, el siguiente paso consiste en visualizar la distribución gráfica de las variables para entender mejor su comportamiento y características.

Para ello, utilizaremos histogramas, que nos permite observar cómo se distribuyen los valores de las variables y la frecuencia con la que ocurren.
También permite detectar asimetrías, aunque, con el tratamiento de datos realizado, estas no deberían es un problema.

Importamos y aplicamos “warnings” ya que el sistema que utilizaremos se dejará de usar en futuras versiones de Python vía Jupyter Notebook, pero en la actualidad funcionan correctamente.

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning,
module="seaborn._oldcore")
```

Establecemos la figura e iteramos las columnas de interés. Ya establecimos la lista de columnas numéricas en la gráfica anterior.

```
plt.figure(figsize=(20, 15))
for i, column in enumerate(columnas_numericas):
    plt.subplot(4, 2, i + 1)
    sns.histplot(XY[column].dropna(), bins=30)
    plt.title(f'Distribución {column}')
plt.tight_layout()
plt.show()
```

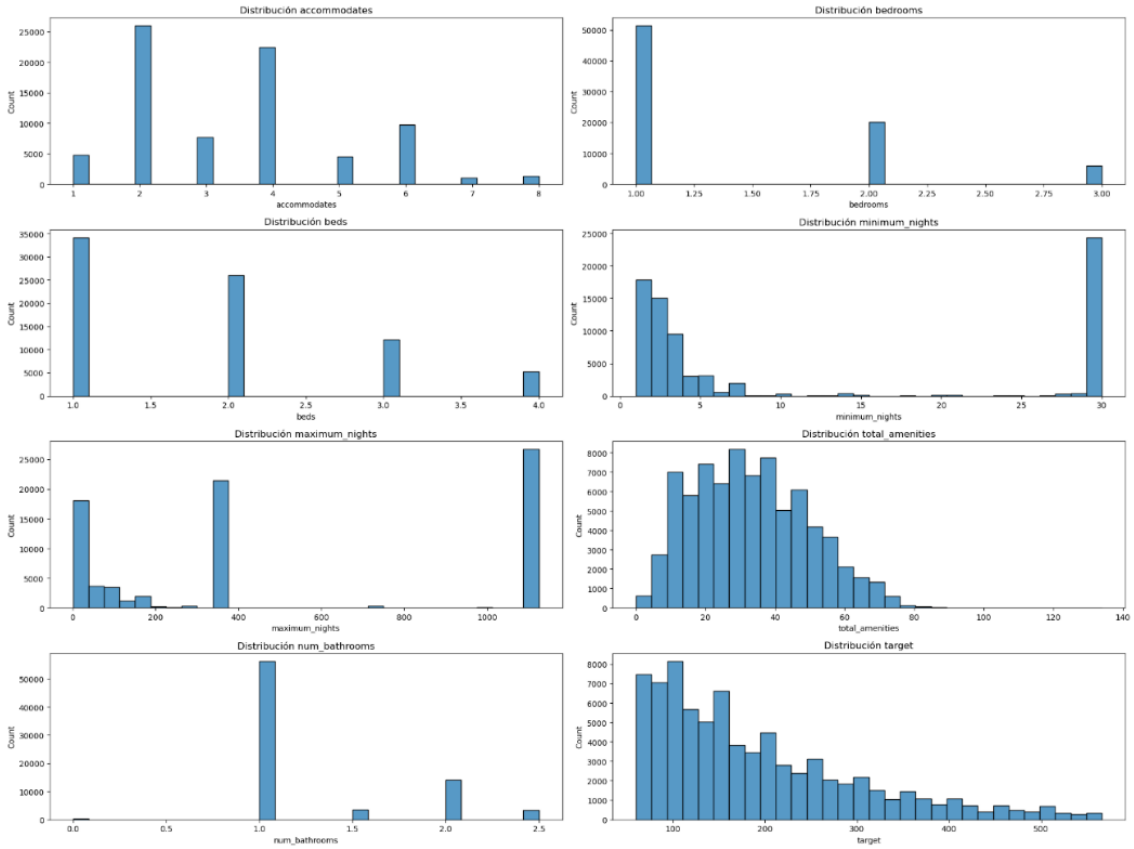

Ilustración 19 - Histograma de distribución

Las variables tienen una distribución con sentido para sus características.

“Accomodates” tiene como valores más frecuentes 2 o 4 inquilinos.

Habitaciones tiene como más frecuente 1, ya que los alquileres de habitación privada, compartida o de hotel suman a esta cifra.

El máximo y mínimo de noches tienen una progresión normal con la peculiaridad de que el máximo y mínimo de noches que permite la aplicación, es decir establecer un mínimo de 30 días (1 mes) de alquiler o el máximo de 365 o 1255 días es decir sin límite son valores muy presentes.

Por último, el valor más común de baños es solo uno, y el total de amenities y el precio total tienden a presentar cada vez menos valores ya que hay más propiedades de tamaños y precios reducidos y pocos de grandes propiedades con valores de lujo.

6.3 Correlación de Pearson

La correlación de las variables permite comprender cómo una variable puede influir en otra. En términos conceptuales, la correlación mide la fuerza y la dirección de la relación lineal entre dos variables. Este análisis es crucial para identificar relaciones significativas y entender cómo los cambios en una variable pueden impactar en otra.

Una correlación positiva significa que a medida que una variable aumenta, la otra también lo hace. Una correlación negativa implica que a medida que una variable aumenta, la otra disminuye.

En primer lugar, vamos a analizar las variables numéricas.
Volvamos a señalar las columnas numéricas y calculamos el matriz de correlación a través de Pearson.

```
XY_numerico = XY[columnas_numericas]
matriz_correlaciones = XY_numerico.corr(method='pearson')
```

Extremos la correlación numérica exacta con target y la mostramos.

```
correlacion_target =
matriz_correlaciones['target'].drop('target').sort_values(ascending=False)
print("Correlación numérica con respecto a 'target' (ordenada de mayor a menor):")
print(correlacion_target)
```

Realizamos también una tabla de correlación que nos ayuda a visualizar no solo la correlación con target si no entre las variables.

```
n_ticks = len(columnas_numericas)
plt.figure(figsize=(12, 10))
plt.xticks(range(n_ticks), columnas_numericas, rotation='vertical')
plt.yticks(range(n_ticks), columnas_numericas)
plt.colorbar(plt.imshow(matriz_correlaciones, interpolation='nearest', vmin=-1., vmax=1., cmap='coolwarm'))
plt.title('Matriz de correlaciones de Pearson')
plt.show()
```

```
Correlación numérica con respecto a 'target' (ordenada de mayor a menor):
accommodates       0.466888
beds               0.387433
bedrooms           0.361056
num_bathrooms      0.323091
total_amenities    0.119476
maximum_nights    -0.039710
minimum_nights    -0.229024
Name: target, dtype: float64
```

Ilustración 20 - Correlación de las variables numéricas

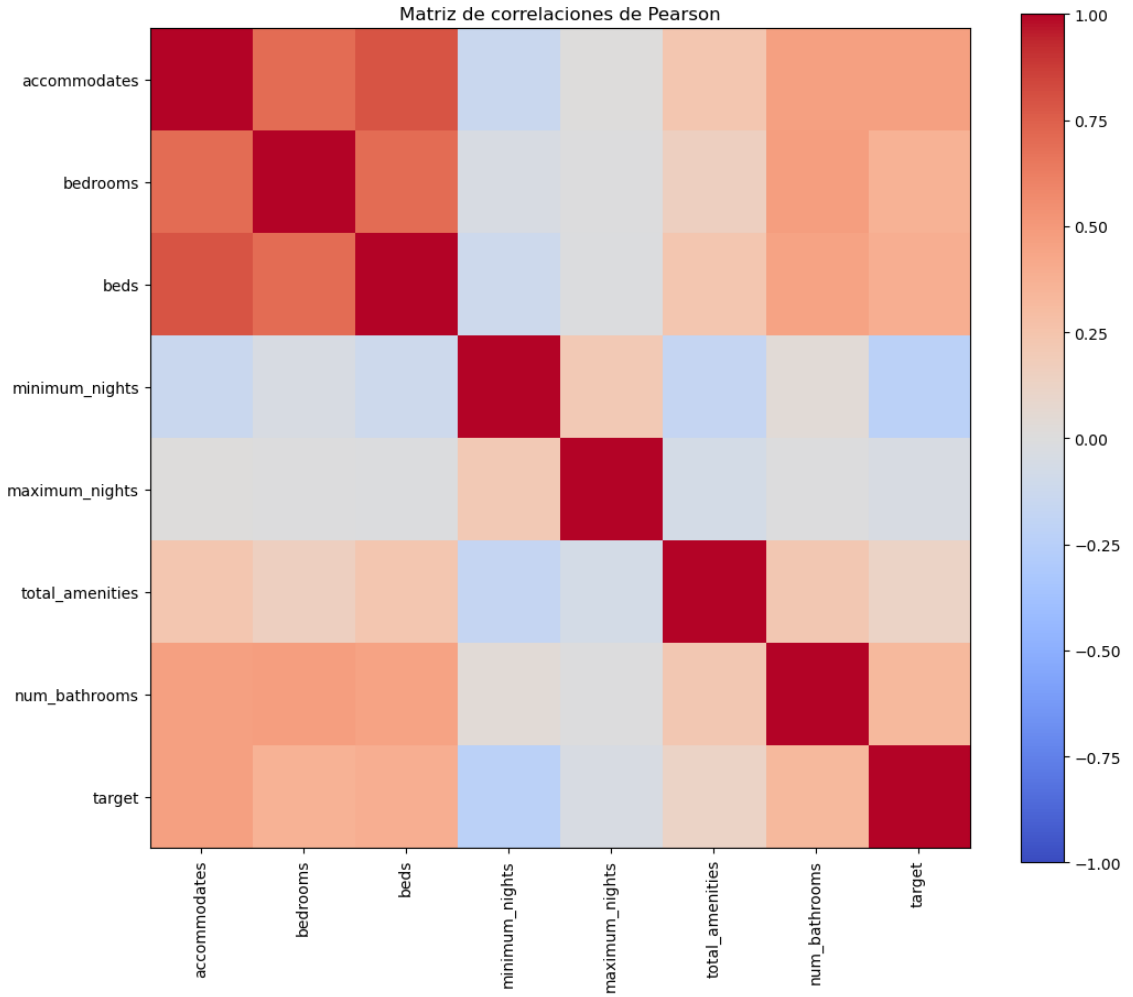

Ilustración 21 - Heatmap correlativo

Al realizar el análisis de correlación, encontramos una tendencia lógica en la relación entre nuestras variables predictoras y la variable objetivo, el precio. Observamos que el aumento en la cantidad de habitaciones, camas, baños, inquilinos o comodidades tiende a incrementar el precio de alquiler. Esta tendencia es intuitiva, ya que las propiedades con más habitaciones y comodidades suelen ofrecer mayor valor y, por lo tanto, pueden justificar precios más altos.

Por otro lado, variables como la cantidad de noches mínimas o máximas requeridas para alquilar la propiedad muestran una correlación inversa con el precio. Específicamente, establecer una cantidad superior de noches mínimas o máximas tiende a reducir el precio. Esto se debe a que un mayor número de noches mínimas impone una restricción en la flexibilidad del alquiler, lo cual puede desincentivar a algunos potenciales inquilinos que buscan estancias más cortas. De la misma manera, un límite superior en el número de noches de estancia puede limitar la demanda de la propiedad, ya que los inquilinos pueden preferir opciones con mayor flexibilidad en la duración de la estancia.

Además de las variables numéricas, las variables categóricas que hemos transformado en dummies también presentan correlaciones con el precio. Para estudiar su impacto vamos a calcular la correlación respecto a estas variables.

Establecemos las columnas dummies con el target (precio) y filtramos la columna.

```
columnas_dummies = ['location_Hawaii', 'location_Los Angeles', 'location_New
York City', 'location_Rhode Island', 'location_Seattle', 'room_type_Entire
home/apt', 'room_type_Hotel room', 'room_type_Private room',
'room_type_Shared room', 'Smoke alarm', 'Wifi', 'Kitchen', 'Essentials',
'Hangers', 'Carbon monoxide alarm', 'Hair dryer', 'Iron', 'Hot water', 'Dishes and
silverware', 'Refrigerator', 'Shampoo',          'Microwave', 'Cooking basics', 'Bed
linens', 'Fire extinguisher', 'Air conditioning', 'Heating', 'First aid kit', 'Self check-
in', 'neighbourhood_group_cleansed_Ballard',
'neighbourhood_group_cleansed_Beacon Hill',
'neighbourhood_group_cleansed_Bristol',
'neighbourhood_group_cleansed_Bronx',
'neighbourhood_group_cleansed_Brooklyn',
'neighbourhood_group_cleansed_Capitol Hill',
'neighbourhood_group_cleansed_Cascade',
'neighbourhood_group_cleansed_Central Area',
'neighbourhood_group_cleansed_City of Los Angeles',
'neighbourhood_group_cleansed_Delridge',
'neighbourhood_group_cleansed_Downtown',
'neighbourhood_group_cleansed_Hawaii',
'neighbourhood_group_cleansed_Honolulu',
'neighbourhood_group_cleansed_Interbay',
'neighbourhood_group_cleansed_Kauai',
'neighbourhood_group_cleansed_Kent',
'neighbourhood_group_cleansed_Lake City',
'neighbourhood_group_cleansed_Magnolia',
'neighbourhood_group_cleansed_Manhattan',
'neighbourhood_group_cleansed_Maui',
'neighbourhood_group_cleansed_Newport',
'neighbourhood_group_cleansed_Northgate',
'neighbourhood_group_cleansed_Other Cities',
'neighbourhood_group_cleansed_Other neighborhoods',
'neighbourhood_group_cleansed_Providence',
'neighbourhood_group_cleansed_Queen Anne',
'neighbourhood_group_cleansed_Queens',
'neighbourhood_group_cleansed_Rainier Valley',
'neighbourhood_group_cleansed_Seward Park',
'neighbourhood_group_cleansed_Staten Island',
'neighbourhood_group_cleansed_Unincorporated Areas',
'neighbourhood_group_cleansed_University District',
'neighbourhood_group_cleansed_Washington',
'neighbourhood_group_cleansed_West Seattle', 'target']

XY_dummies = XY[columnas_dummies]
```

Calculamos la correlación de pearson, esta vez son muchos valores así que vamos a mostrar en display solo los más influyentes.

```
matriz_correlaciones = XY_dummies.corr(method='pearson')
correlacion_target =
matriz_correlaciones['target'].drop('target').sort_values(ascending=False)
print("Alta correlación positiva con el precio:")
print(correlacion_target.head(2))
print("Alta correlación negativa con el precio:")
print(correlacion_target.tail(2))
```

```
Alta correlación positiva con el precio:
location_Hawaii              0.395048
room_type_Entire home/apt    0.360759
Name: target, dtype: float64
Alta correlación negativa con el precio:
location_New York City   -0.192913
room_type_Private room   -0.360218
Name: target, dtype: float64
```

Ilustración 22 - Funcionamiento de las correlaciones

Podemos apreciar cómo las variables dummy también presentan una correlación significativa con la variable objetivo, el precio de alquiler. Este análisis nos permite identificar tendencias específicas en función de características categóricas, que hemos transformado en variables dummy para incluirlas en nuestro modelo.

Por ejemplo, respecto a la localización, podemos observar que la localización de las propiedades tiene un impacto directo y considerable en el precio de alquiler. Las propiedades ubicadas en Hawái tienden a tener precios significativamente más altos. Esto puede deberse a la alta demanda turística. Por otro lado, Nueva York tiende a mostrar precios más bajos en comparación con Hawái.
Este análisis confirma que la localización influye en el precio de alquiler. Diferentes localizaciones presentan diferentes niveles de demanda y valores percibidos, lo cual se refleja directamente en los precios.

Además, el tipo de propiedad también muestra una correlación significativa con el precio. Tal y como podemos ver en los resultados de la correlación, una casa entera tiende a tener precios de alquiler más altos. Algo lógico debido a la cantidad de espacio e independencia que ofrece una vivienda al completo. Por otro lado, las propiedades que ofrecen solo una habitación privada tienden a tener precios más bajos precisamente por el mismo motivo, menor independencia, espacio y normalmente, comodidades.

Estas observaciones son coherentes con las expectativas y confirman que nuestro modelo está capturando de manera efectiva las relaciones lógicas entre las características de las propiedades y sus precios de alquiler.

7. Test & Train para Machine Learning.

Para evaluar adecuadamente el rendimiento de nuestro modelo de aprendizaje automático, tenemos que dividir nuestros datos en dos conjuntos distintos: el conjunto de entrenamiento (train) y el conjunto de prueba (test).

El conjunto de prueba (test) representará los datos que se utilizarán para que el modelo realice las predicciones y se compare con los resultados reales. Por otro lado, el conjunto de entrenamiento (train) está compuesto por los datos que se emplearán para nutrir y ajustar el modelo de Machine Learning.

Ambas partes de la división utilizarán los dos Dataframes el referente al conjunto de variables explicativas (X) y el referente a la variable objetivo (Y).

He decidido utilizar una división del 15% para el conjunto de prueba y el 85% para el conjunto de entrenamiento. Esta proporción ha sido seleccionada después de probar varias divisiones y determinar que esta configuración ofrecía un modelo con mejores resultados en términos de precisión, al mismo tiempo que evitaba el sobreajuste (overfitting). Esto significa que el 85% de los datos se utilizarán para entrenar el modelo, mientras que el 15% restante se empleará para evaluar su precisión.

Para realizar esta división, utilizaremos la función train_test_split de la librería sklearn.model_selection, que está diseñada específicamente para dividir los datos de manera eficiente entre los conjuntos de entrenamiento y prueba.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=0)
```

Ahora que hemos limpiado y preparado los datos, y que hemos definido nuestra variable objetivo (target), estamos listos para aplicar varios modelos de aprendizaje automático. Nuestro objetivo es identificar el modelo más eficaz, capaz de realizar predicciones precisas y confiables sobre el precio de las propiedades.

El Machine Learning establece un algoritmo que se entrena con datos (en este caso nuestro dataframe X e Y para aprender patrones y relaciones entre las variables. Una vez entrenado, el modelo utiliza estos patrones para hacer predicciones sobre valores desconocidos de la variable objetivo. Y este es precisamente el resultado que queremos obtener, un algoritmo con la capacidad de predecir un precio desconocido según los patrones que siguen el resto de las variables.

Al probar diferentes algoritmos de Machine Learning, ajustaremos sus parámetros y evaluaremos su rendimiento utilizando los conjuntos de entrenamiento y prueba. Esto nos permitirá seleccionar el modelo que ofrezca los mejores resultados y que sea más adecuado para nuestras necesidades específicas.

8. Modelos de Machine Learning

8.1 Modelo - Regresión Lineal

La regresión lineal es un enfoque para tomar en el Machine Learning, que se utiliza para modelar la relación entre una o más variables independientes y una variable dependiente (en este caso, el precio) de forma lineal. Este método asume que la relación entre las variables puede ser aproximada por una línea recta.

Este modelo tiene potencial para usarse en el modelo debido a que hemos observado una relación lineal entre las variables independientes y la variable dependiente.

Para establecer el modelo utilizamos Linear Regression, que como su nombre indica permite establecer una regresión lineal. Creamos el modelo de regresión señalando los tramos de entrenamiento (train) de los datos.

```
from sklearn.linear_model import LinearRegression
modelo_regresion = LinearRegression()
modelo_regresion.fit(X_train, Y_train)
```

Aplicamos la predicción de Y (precio) según los datos de entrenamiento y test.

```
Y_pred_train = modelo_regresion.predict(X_train)
Y_pred_test = modelo_regresion.predict(X_test)
```

Importamos los recursos para calcular el error cuadrático medio y el coeficiente determinación. A partir de ahora referidos como MSE y R^2 en su forma abreviada.

El error cuadrático medie cual es el promedio de los errores al cuadrado entre el valor de un estimador y el de su estimación. Es un parámetro muy útil para realizar predicciones como en este caso. A la hora de comparar modelos, aquel que presente un menor MSE será representativo de una mejor calidad de modelo.

El coeficiente de determinación (R^2) es un estadístico establece entre 0 y 1 según la capacidad de predicción del modelo. Siendo 0 la total incapacidad de predicción y 1 una predicción perfecta en todos los casos. Un modelo con un alto R^2 generalmente es el mejor modelo para elegir (Excepto si bajo escenarios de sobreajuste).

Una vez establecido los valores de MSE y R^2 realizaremos también la representación gráfica de las predicciones

```
from sklearn.metrics import mean_squared_error, r2_score
mse_train = mean_squared_error(Y_train, Y_pred_train)
r2_train = r2_score(Y_train, Y_pred_train)
mse_test = mean_squared_error(Y_test, Y_pred_test)
r2_test = r2_score(Y_test, Y_pred_test)
print(f'Evaluación del modelo:')
print(f'MSE: {mse_test}')
print(f'R²: {r2_test}')
```

Gráfica con los resultados

```
plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_test, color='blue', edgecolor='w', alpha=0.6)
plt.plot([Y.min(), Y.max()], [Y.min(), Y.max()], 'r--', linewidth=2)
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Predicciones del modelo de regresión')
plt.show()
```

Evaluación del modelo del modelo de regresión lineal:
MSE: 6718.749406826796
R^2: 0.4658384175652456

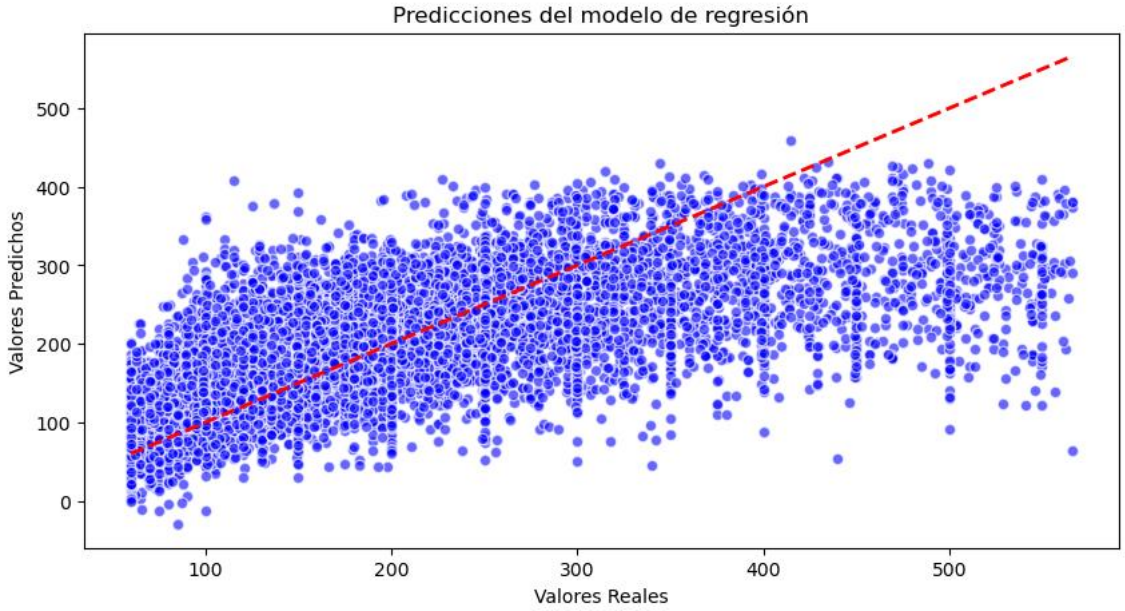

Ilustración 23- Regresión lineal

El valor del Error Cuadrático Medio (MSE) por sí solo es insuficiente para evaluar la calidad del modelo de regresión lineal sin compararse con otros modelos. De la misma forma, aunque el Coeficiente de Determinación (R^2) es una métrica útil para evaluar la capacidad predictiva del modelo, el valor actual de R^2 es inferior a 0.5. Esto sugiere que el modelo explica menos del 50% de la variabilidad observada en los datos, lo cual es indicativo de una capacidad predictiva subóptima.

Al analizar la dispersión entre los valores reales y los valores predichos, se observa que, aunque existe una tendencia coherente entre ellos, la dispersión es considerablemente alta. Esta alta dispersión sugiere que el modelo no está capturando adecuadamente la relación entre las variables independientes y el precio, lo que resulta en predicciones menos precisas.

Dado el desempeño del modelo actual puede ser suficiente pero mejorable, vamos a otros modelos con el objetivo de mejorar la precisión predictiva.

8.2 Modelos - Ridge y Lasso

En el ámbito del Machine Learning, existen técnicas avanzadas como Ridge y Lasso, que son populares para mejorar los modelos de predicción.
Estas técnicas, aunque independientes, comparten características similares y a menudo se presentan conjuntamente debido a sus propiedades complementarias.

Ridge y Lasso son particularmente eficaces para evitar el problema del sobreajuste.
El sobreajuste es un fenómeno que se da cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, con datos muy concretos ligados a sus datos. Cuando se introducen datos nuevos ajenos al entrenamiento, el modelo es incapaz de hacer predicciones precisas ya que le falta flexibilidad para interpretar correctamente los nuevos datos.

Evitar el sobreajuste es importante en nuestro modelo predictivo donde hay variables categóricas muy concretas y con mucha influencia como la localización que pueden estancar los precios en cifras muy concretas.

Ambos métodos introducen una penalización al modelo con el objetivo de regularizar los coeficientes de la regresión, minimizando el impacto de las variables más irrelevantes y mejorando la generalización del modelo.

Estos métodos agregan una penalización al modelo a fin de reducir los coeficientes y eliminar predictores irrelevantes.

La Regresión de Ridge es una técnica que agrega una penalización de tipo “L2”. Esta es la suma de los cuadrados de los coeficientes a la función de costo. Esta penalización reduce la magnitud de los coeficientes, pero no los reduce a cero. Es eficaz en situaciones donde todos los predictores son relevantes y se busca evitar coeficientes extremadamente grandes.
Importamos el modelo para su aplicación.

```
from sklearn.linear_model import Ridge
```

Establecemos el modelo, el valor alpha 1 indica la penalización. El valor 1 se utiliza para una penalización moderada para evitar que el impacto de los coeficientes sea demasiado grande.

```
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, Y_train)
```

Realizamos las predicciones y evaluamos el modelo.

```
Y_pred_ridge = ridge_model.predict(X_test)
mse_ridge = mean_squared_error(Y_test, Y_pred_ridge)
r2_ridge = r2_score(Y_test, Y_pred_ridge)

print(f'MSE: {mse_ridge}')
print(f'R²: {r2_ridge}')
```

Observamos la representación gráfica

```
plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_ridge, color='violet', alpha=0.5, label='Predicciones Ridge')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red', lw=2, label='Línea de Igualdad')
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Regresión de Ridge')
plt.legend()
plt.show()
```

Evaluación del Modelo de Regresión de Ridge
MSE: 6718.778884659101
R^2: 0.4658360739854218

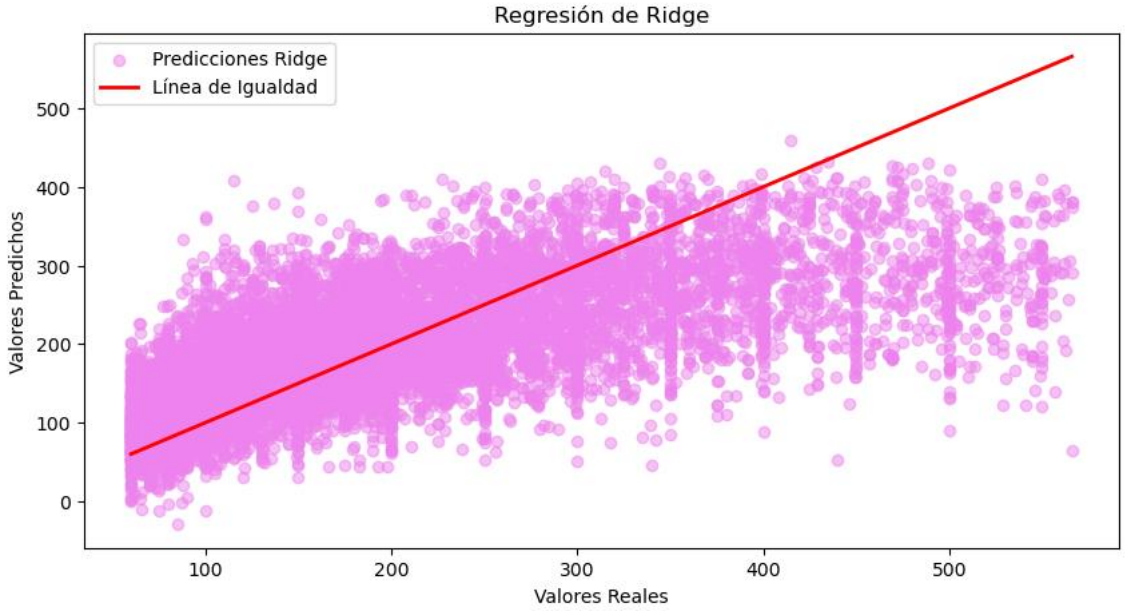

Ilustración 24 - Regresión de Ridge

Podemos ver poca variación respecto al modelo de regresión lineal.
Vamos a aplicar Lasso antes de extraer conclusiones.

Lasso emplea una penalización de tipo “L1”, que es la suma de los valores absolutos de los coeficientes. Esta penalización tiene la propiedad de forzar los coeficientes a ser exactamente cero. Esto implica que realiza una selección de variables e ignora aquellas que detecta como irrelevantes.

El efecto de Lasso pues es a efectos prácticos simplificar el modelo utilizando solo las variables más relevantes y significativas para el mismo. Deshacerse de variables más allá de la eliminación que hemos hecho de forma manual desde el análisis personal es una estrategia que puede funcionar, ya que el modelo puede detectar irrelevancias con su algoritmo que podemos no haber apreciado a simple vista desde en análisis estadístico básico que se hace sobre los datos.

Importamos el modelo para ejecutarlo con nuestros datos.

```
from sklearn.linear_model import Lasso
```

Establecemos el modelo y predicción.

```
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, Y_train)
Y_pred_lasso = lasso_model.predict(X_test)
```

Evaluamos el modelo

```
mse_lasso = mean_squared_error(Y_test, Y_pred_lasso)
r2_lasso = r2_score(Y_test, Y_pred_lasso)
print(f'MSE: {mse_lasso}')
print(f'R²: {r2_lasso}')
```

Establecemos la gráfica

```
plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_lasso, color='purple', alpha=0.5, label='Predicciones Lasso')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red', lw=2, label='Línea de Igualdad')
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Regresión de Lasso')
plt.legend()
plt.show()
```

Evaluación del Modelo de Regresión de Ridge

MSE: 6762.1387506757765

R²: 0.4623888290826586

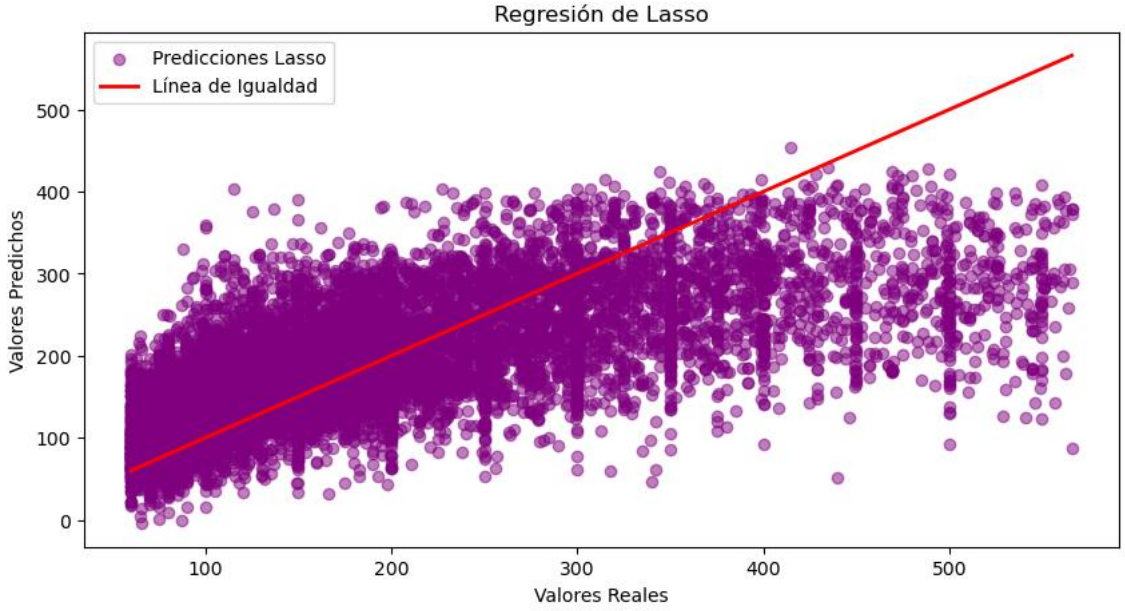

Ilustración 25 - Regresión de Lasso

Podemos apreciar como con respecto Ridge, el MSE y R^2 de Lasso ha aumentado levemente, esto implica que la calidad del modelo cuanto más se reducen los coeficientes de las variables que considera irrelevantes más calidad pierde, por lo que, en general, todas las variables son, aunque sea en poca medida, relevantes y significativas.

Cuando comparamos los valores obtenidos en los 3 modelos podemos apreciar que son muy similares:

Linear - MSE: 6718.715740020915 / R^2: 0.4658410941815645
Ridge - MSE: 6718.778884659101 / R^2: 0.465836073985421
Lasso - MSE: 6762.1387506757765 / R^2: 0.4623888290826586

El mejor modelo por muy poca diferencia en este momento es el lineal, ya que es el que tiene menor MSE y mayor R^2. La poca variación en los resultados del modelo implica que es un modelo robusto, sin sobreajuste y con variables concretas con mucha relevancia. Por lo que, si bien la capacidad predictiva del mismo es mejorable, en líneas generales los datos parecen bien tratados. Por lo tanto, vamos a seguir con el objetivo de encontrar el modelo con la mayor capacidad predictiva posible.

8.3 Modelos - Decision Tree y Random Forest

Con la información que tenemos sobre las características del modelo y las variables, y sabiendo el alto impacto de las variables categóricas, otros métodos de aprendizaje supervisado para el machine learning que podemos emplear son Decision Tree y Random Forest.

Decision Tree, o en español, Árbol de decisión, es un algoritmo que emplea clasificación y regresión para establecer su algoritmo. El meto divide las características de los datos en subconjuntos homogéneos.
El árbol empieza por una base compuesta por todos los datos del modelo y selecciona las características para iterar entre los subconjuntos, creando clasificaciones cada vez más específicas y siendo los resultados finales de la iteración la predicción obtenida.

Este es un modelo sencillo y con capacidad de interpretar de forma eficaz las variables categóricas. Esto es positivo para nuestro modelo debido a que hay un número elevado de las mismas. No obstante, es un modelo con propensión al sobreajuste y que muestra dificultades para establecer relaciones complejas.

Importamos el regresor para realizar el Decision Tree a fin de estudiar si, aunque parece contraintuitivo con la información que hemos analizado hasta ahora, en su simplicidad está la clave de una estimación adecuada. Posteriormente abordaremos el modelo de Random Forest

```
from sklearn.tree import DecisionTreeRegressor
```

Establecemos el modelo y predecimos.

```
tree_model = DecisionTreeRegressor(random_state=0)
tree_model.fit(X_train, Y_train)
Y_pred_tree = tree_model.predict(X_test)
```

Evaluamos el modelo.

```
mse_tree = mean_squared_error(Y_test, Y_pred_tree)
r2_tree = r2_score(Y_test, Y_pred_tree)
print(f'MSE: {mse_tree}')
print(f'R²: {r2_tree}')
```

Visualizamos de los resultados

```
plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_tree, color='limegreen', alpha=0.5,
label='Predicciones Árbol de Decisión')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red',
lw=2, label='Línea de Igualdad')
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Decision Tree')
plt.legend()
plt.show()
```

Evaluación del Modelo Decision Tree:

MSE: 10159.382809402248

R²: 0.19229730572824688

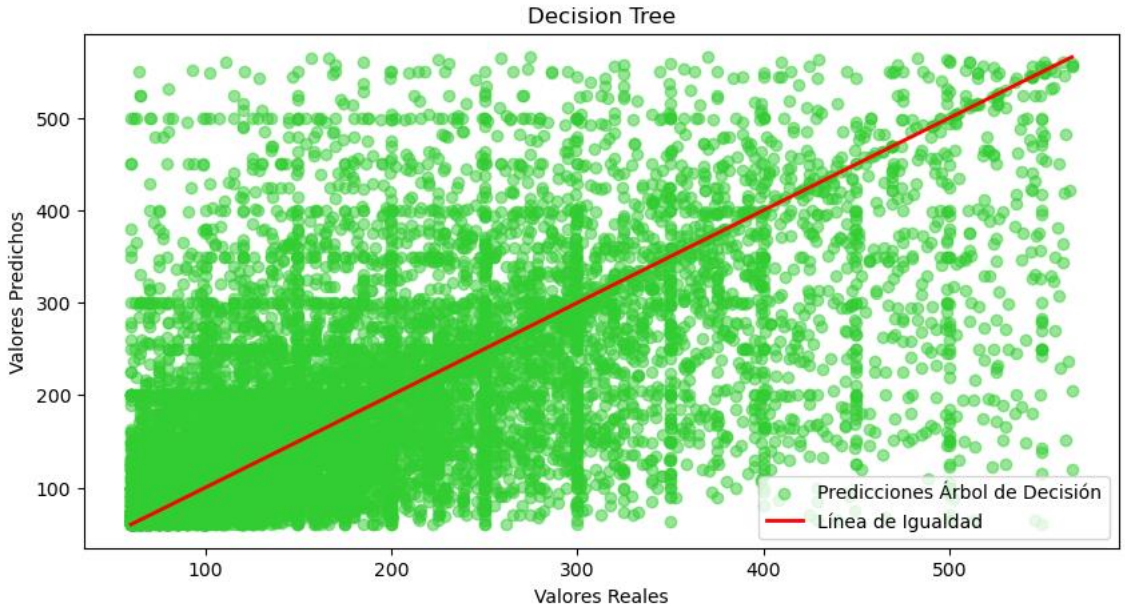

Ilustración 26 - Decision Tree

Los resultados del modelo son mucho peores a los modelos previos. Esto puede ser debido a que tenemos un amplio número de variables con poca linealidad entre ellas. Como se ha explicado previamente este modelo no es bueno trabajando relaciones complejas entre muchas variables.

A pesar de que la primera impresión sea que este tipo de modelo no es útil para los datos utilizados, el modelo derivado conocido como Random Forest puede dar grandes resultados al ser capaz de tratar con mejor resultado las relaciones complejas del modelo.

Random Forest, o Bosque Aleatorio, es una extensión de Decision Tree que busca mejorar el modelo, incrementando robustez y precisión. Esto se consigue mediante la unión de una gran cantidad de árboles de decisión. El método utilizado se conoce como "bootstrap aggregating" y crear múltiples subconjuntos de datos de entrenamiento mediante un muestreo con reemplazo. Cada árbol en el bosque se entrena en un subconjunto aleatorio de las características, y las predicciones finales se obtienen agregando los resultados obtenidos con los promedios que mejor resultado han mostrado.

Es un modelo que requiere más tiempo y recursos para procesar, pero permite establecer muchas más relaciones entre las muchas variables del modelo aun teniendo relaciones complejas.

Importamos el regresor, establecemos el modelo y realizamos las predicciones.

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=0)
rf_model.fit(X_train, Y_train)
Y_pred_rf = rf_model.predict(X_test)
```

Evaluamos y visualizamos el modelo.

```
mse_rf = mean_squared_error(Y_test, Y_pred_rf)
r2_rf = r2_score(Y_test, Y_pred_rf)
print(f'MSE: {mse_rf}')
print(f'R²: {r2_rf}')
plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_rf, color='green', alpha=0.5, label='Predicciones Random Forest')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red', lw=2, label='Línea de Igualdad')
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Random Forest')
plt.legend()
plt.show()
```

Evaluación del Modelo Random Forest
MSE: 5238.500219330838
R^2: 0.5835228556226009

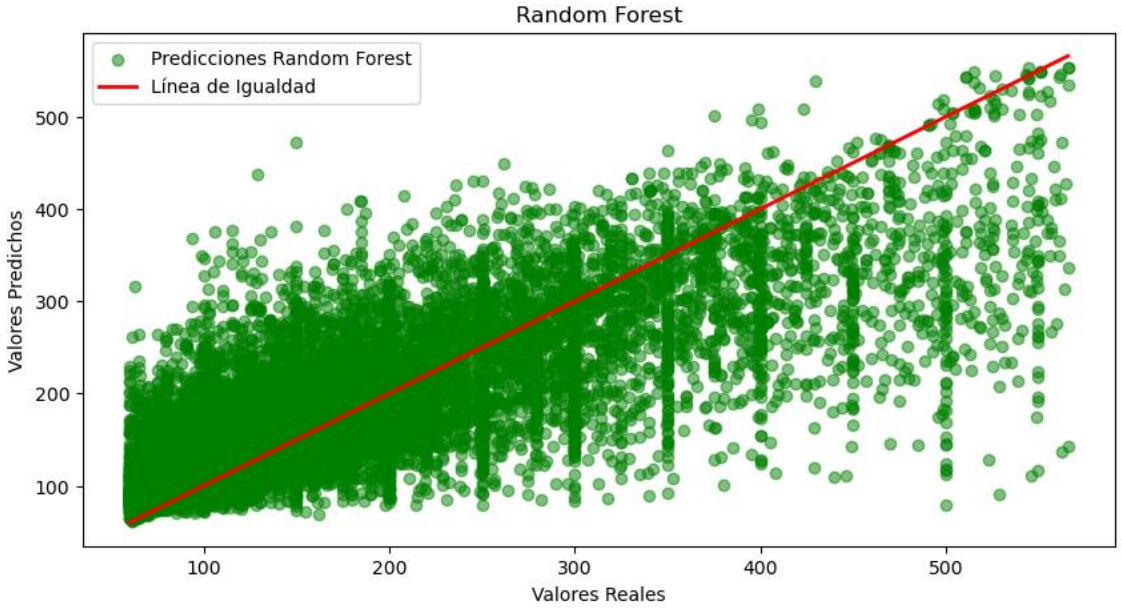

Ilustración 27 - Random Forest

El modelo ha funcionado muy bien, presenta el mejor R^2 hasta el momento y el menor MSE. Esto implica que la potencia del modelo ha funcionado correctamente al ser capaz de relacionar un número amplio de variables con relaciones complejas.

Entendiendo las características de los datos del modelo y su comportamiento respecto a la correlación entre las variables independientes y el precio, podemos deducir que este modelo, que además presenta unos datos de evaluación estadísticos aceptable, es el mejor posible. No obstante, vamos a aplicar por último un modelo de Gradient Boosting, asegurándonos de haber probado todos los potenciales candidatos a modelo antes de tomar una decisión definitiva.

8.4 Modelo - Gradient Boosting

Gradient Boosting es un modelo con una gran capacidad de clasificación. Este construye modelos de forma secuencial, corrigiendo los errores cometidos en cada secuencia, obteniendo cada vez modelos más precisos. Esto lo hace a través de gradiente de error que evoluciona a medida que se obtienen resultados.

Este modelo es interesante de plantear ya que es especialmente bueno para capturar relaciones no lineales y complejas. Si bien hemos detectado cierta linealidad en las variables, algo que tiene sentido sobre todo para las variables numéricas, como es el número de camas, puede que el impacto de la existencia o no de un amenities concretos haya condicionado gravemente el modelo.

Importamos el regresor, establecemos el modelo y realizamos la predicción.

```
from sklearn.ensemble import GradientBoostingRegressor
gb_model = GradientBoostingRegressor(random_state=0)
gb_model.fit(X_train, Y_train)
Y_pred_gb = gb_model.predict(X_test)
```

Evaluamos el modelo y vemos su gráfica correspondiente

```
mse_gb = mean_squared_error(Y_test, Y_pred_gb)
r2_gb = r2_score(Y_test, Y_pred_gb)
print(f'Gradient Boosting Mean Squared Error: {mse_gb}')
print(f'Gradient Boosting R^2 Score: {r2_gb}')

plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_gb, color='gold', alpha=0.5, label='Predicciones Gradient Boosting')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red', lw=2, label='Línea de Igualdad')
plt.xlabel('Valores Reales')
plt.ylabel('Valores Predichos')
plt.title('Gradient Boosting')
plt.legend()
plt.show()
```

Evaluación del Modelo Gradient Boosting.

MSE: 6235.593245363433

R^2: 0.5042508428758454

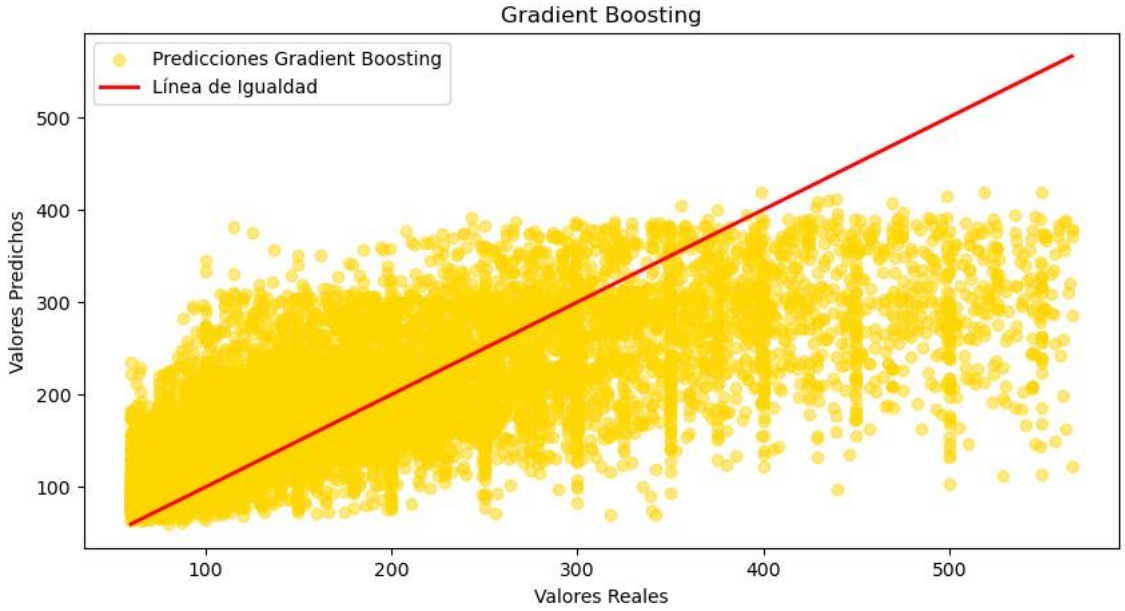

Ilustración 28 - Gradient Boosting

El modelo funciona relativamente bien, de hecho, tiene mayor R2 y menor MSE que otros modelos. Esto puede implicar como hemos mencionado, que hay variables con relación no linear que pueden tener un alto impacto en el modelo.

8.5 Conclusiones respecto a los modelos

La evaluación de los modelos mediante Error Cuadrático Medio y Coeficiente de Determinación es la siguiente:

- Linear - MSE: 6718.71 / R^2: 0.46584
- Ridge - MSE: 6718.77 / R^2: 0.46583
- Lasso - MSE: 6762.13 / R^2: 0.462388
- Decision Tree - MSE: 10159.38 / R^2: 0.19229
- Random Forest - MSE: 5238.50 / R^2: 0.58359
- Gradient Boosting - MSE: 6235.59 / R^2: 0.50425

Un menor error cuadrático medio implica que, en general, las predicciones del modelo son más cercanas a los valores reales, el modelo comete menos errores en sus predicciones.
Por otro lado, un mayor coeficiente de determinación, que mide la variabilidad de la variable precio respecto al resto de variables, significa que el modelo captura mejor la relación entre las mismas.

De acuerdo con estas métricas, el modelo de Random Forest es el que presenta el mejor desempeño, ya que tiene el MSE más bajo (5238.50) y el R^2 más alto (0.58359).

Recapitulando las conclusiones que hemos obtenido en el análisis de cada modelo, podemos determinar que el modelo:

- Es robusto.
- No presenta sobreajuste.
- El alto número de datos y variables requiere de un modelo con capacidad de procesar relaciones complejas.
- La reducción de coeficientes no otorga un mejor modelo por lo que las variables sutilizadas se plantean relevantes.
- Su capacidad predictiva es buena y con tendencias claras que son adaptables a algoritmos, pero una mayor cantidad de datos permitiría predicciones más precisas.

Con estas conclusiones, y comparado las evaluaciones de calidad de los modelos, Random Forest logra una mejor precisión en las predicciones y ajusta mejor los datos observados. Por lo tanto, podemos concluir que, basado en estas métricas, el modelo de Random Forest es el mejor para predecir dada nuestra base de datos y va a ser sobre el que apliquemos las predicciones.

Haber construido con éxito un modelo vía Machine Learning capaz de estimar las variables precios es la base técnica necesaria y un objetivo fundamental para cumplir los objetivos del proyecto.

9. Estimador de precios

9.1 Estimador Manual

Por último, vamos a establecer el estimador de precios.
En primer lugar, vamos a introducir un dato de forma manual para asegurarnos de que el modelo lo procesa correctamente y es capaz de predecir el valor del precio correctamente.

Para ello, tenemos que crear un dato estableciendo el valor de las distintas variables presentes en el modelo. Es decir, todas las variables independientes que componen el dataframe X.

Vamos a establecer como dato un ejemplo de una propiedad genérica con cantidades adecuadas al modelo. Una casa en Brooklyn, Nueva York. Permite el alojamiento a 4 personas, con dos habitaciones, tres camas, un baño y distintas comodidades seleccionadas aleatoriamente. Al haber transformado todas las variables como numéricas, las variables dummies tendrán un valor 1 o 0 según su existencia o ausencia.
Establecemos el dato que queremos predecir y el valor de cada una de sus variables.

```
dato_prueba = {
    'accommodates': 4,
    'bedrooms': 2,
    'beds': 3,
    'minimum_nights': 1,
    'maximum_nights': 30,
    'Smoke alarm': 1,
    'Wifi': 1,
    'Kitchen': 1,
    'Essentials': 1,
    'Hangers': 0,
    'Carbon monoxide alarm': 0,
    'Hair dryer': 1,
    'Iron': 1,
    'Hot water': 0,
    'Dishes and silverware': 0,
    'Refrigerator': 1,
    'Shampoo': 0,
    'Microwave': 1,
    'Cooking basics': 0,
    'Bed linens': 1,
    'Fire extinguisher': 0,
    'Air conditioning': 1,
    'Heating': 1,
    'First aid kit': 1,
    'Self check-in': 0,
    'total_amenities': 15,
    'num_bathrooms': 1.0,
```

'neighbourhood_group_cleansed_Ballard': 0,
'neighbourhood_group_cleansed_Beacon Hill': 0,
'neighbourhood_group_cleansed_Bristol': 0,
'neighbourhood_group_cleansed_Bronx': 0,
'neighbourhood_group_cleansed_Brooklyn': 1,
'neighbourhood_group_cleansed_Capitol Hill': 0,
'neighbourhood_group_cleansed_Cascade': 0,
'neighbourhood_group_cleansed_Central Area': 0,
'neighbourhood_group_cleansed_City of Los Angeles': 0,
'neighbourhood_group_cleansed_Delridge': 0,
'neighbourhood_group_cleansed_Downtown': 0,
'neighbourhood_group_cleansed_Hawaii': 0,
'neighbourhood_group_cleansed_Honolulu': 0,
'neighbourhood_group_cleansed_Interbay': 0,
'neighbourhood_group_cleansed_Kauai': 0,
'neighbourhood_group_cleansed_Kent': 0,
'neighbourhood_group_cleansed_Lake City': 0,
'neighbourhood_group_cleansed_Magnolia': 0,
'neighbourhood_group_cleansed_Manhattan': 0,
'neighbourhood_group_cleansed_Maui': 0,
'neighbourhood_group_cleansed_Newport': 0,
'neighbourhood_group_cleansed_Northgate': 0,
'neighbourhood_group_cleansed_Other Cities': 0,
'neighbourhood_group_cleansed_Other neighborhoods': 0,
'neighbourhood_group_cleansed_Providence': 0,
'neighbourhood_group_cleansed_Queen Anne': 0,
'neighbourhood_group_cleansed_Queens': 0,
'neighbourhood_group_cleansed_Rainier Valley': 0,
'neighbourhood_group_cleansed_Seward Park': 0,
'neighbourhood_group_cleansed_Staten Island': 0,
'neighbourhood_group_cleansed_Unincorporated Areas': 0,
'neighbourhood_group_cleansed_University District': 0,
'neighbourhood_group_cleansed_Washington': 0,
'neighbourhood_group_cleansed_West Seattle': 0,
'room_type_Entire home/apt': 1,
'room_type_Hotel room': 0,
'room_type_Private room': 0,
'room_type_Shared room': 0,
'bathrooms_text_clean_ bath': 1,
'bathrooms_text_clean_ private bath': 0,
'bathrooms_text_clean_ shared bath': 0,
'bathrooms_text_clean_Half-bath': 0,
'bathrooms_text_clean_Private half-bath': 0,
'bathrooms_text_clean_Shared half-bath': 0,
'location_Hawaii': 0,
'location_Los Angeles': 0,
'location_New York City': 1,
'location_Rhode Island': 0,
'location_Seattle': 0,}

Convertimos los datos del diccionario a un DataFrame para garantizar que las columnas estén en el mismo orden que el DataFrame de entrenamiento del modelo.

```
dataframe_prueba = pd.DataFrame([dato_prueba])
```

Convertimos el DataFrame a un array de numpy, ya que es el formato esperado por el modelo de Random Forest.

```
nuevo_array = dataframe_prueba.to_numpy()
```

Utilizamos el modelo de Random Forest previamente entrenado (rf_model) para predecir el precio.

```
prediccion_random_forest = np.array([tree.predict(nuevo_array) for tree in rf_model.estimators_])
```

En lugar de proporcionar un valor exacto, establecemos un rango estimado basado en los percentiles 45 y 55 de las predicciones obtenidas. Esto proporciona una estimación más robusta y manejable para el cliente.

Esto no solo asegura que la predicción sea en carácter generales más precisas, al tener un mayor margen de para establecer el precio y por lo tanto menos probabilidad de error. Si no que además le da al cliente la capacidad de terminar de decidir el precio según sus criterios no recogidos en el algoritmo que le ofrece la herramienta.

```
estimacion_inferior = np.percentile(prediccion_random_forest, 45)
estimacion_superior = np.percentile(prediccion_random_forest, 55)
```

Finalmente, mostramos el rango estimado del precio en inglés, ya que el público objetivo de esta herramienta habla este idioma.

```
print(f"The recommended price per night is: ${estimacion_inferior:.2f} - ${estimacion_superior:.2f}")
```

The recommended price per night is: $153.30 - $180.00.

El precio recomendado por noche de esta casa completa para 4 personas sería de entre $153.30 y $180

El precio dado es plausible y tiene sentido con respecto a la media de los datos presentes en el modelo. En esta situación lo óptimo es introducir más posibles datos para ver si el modelo es capaz de dar resultados óptimos para viviendas con otras características. Esto se va a efectuar a través del estimador como herramienta final, que invita a una introducción de datos cómoda.

9.2 Estimador con Widgets

Para cumplir el último objetivo del proyecto, que es crear un estimador accesible para el usuario promedio que pueda utilizar directamente en forma de aplicación, tenemos que plantear el método de introducción de los datos.

Obviamente, el usuario no va a escribir código para añadir un dato nuevo, en este caso las características de la vivienda cuyo precio quiere estimar, por lo que tenemos que crear un entorno con un diseño amigable y visualmente entendible para que pueda introducir estos datos con normalidad, siendo el programa quién interprete las entradas de datos y la traduzca en un formato de código compatible con as variables del Dataframe X y por lo tanto, para que pueda aplicarlo de manera automatizada a nuestro de Random Forest.

Para crear esta interfaz de usuario, utilizaremos “ipywidgets”, una biblioteca de Python que permite crear widgets interactivos en Jupyter Notebooks y otras interfaces HTML. Además, utilizaremos IPython.display para mejorar la presentación visual de los widgets y el resultado final.

Establecer un formato HTML permite ciertas ventajas, y es que lo hace que sea fácilmente exportable a formato web, pudiendo llevar a la implementación y uso real del modelo con relativa facilidad.

```
import ipywidgets as widgets
from IPython.display import display, HTML
```

Creamos una serie de widgets para capturar las diferentes características de la propiedad. Los widgets “IntSlider” se utilizarán para las variables numéricas y los “Checkbox” para las variables dummies.

Tal y como se ha explicado en el aparato referente a los outliers, el dataframe presenta máximos y mínimos coherentes con las características disponibles de una vivienda. Es decir, permitir que se pueda añadir un dato donde se establecen 1000 camas, por un lado, crea una interfaz menos exacta y profesional, y por otro dará una estimación errónea ya que los datos de entrenamiento no incluían nada remotamente similar, teniendo el impacto del coeficiente de 1000 camas una influencia irrealista en el precio.

Añadimos los posibles valores para cada uno de los widgets.

Por lo que, al introducir las variables numéricas como IntSlider, establecemos un mínimo y máximo atendiendo los datos máximos y mínimos que hemos apreciado en el análisis estadístico de los datos.

```
accommodates = widgets.IntSlider(min=1, max=10, value=0, description='Accommodates')
bedrooms = widgets.IntSlider(min=0, max=10, value=0, description='Bedrooms')
beds = widgets.IntSlider(min=0, max=10, value=0, description='Beds')
minimum_nights = widgets.IntSlider(min=1, max=30, value=1, description='Min Nights')
maximum_nights = widgets.IntSlider(min=1, max=365, value=30, description='Max Nights')
smoke_alarm = widgets.Checkbox(value=True, description='Smoke Alarm')
wifi = widgets.Checkbox(value=True, description='Wifi')
kitchen = widgets.Checkbox(value=True, description='Kitchen')
essentials = widgets.Checkbox(value=True, description='Essentials')
hangers = widgets.Checkbox(value=False, description='Hangers')
carbon_monoxide_alarm = widgets.Checkbox(value=False, description='CO Alarm')
```

```
hair_dryer = widgets.Checkbox(value=True, description='Hair Dryer')
iron = widgets.Checkbox(value=True, description='Iron')
hot_water = widgets.Checkbox(value=False, description='Hot Water')
dishes_and_silverware = widgets.Checkbox(value=False, description='Dishes')
refrigerator = widgets.Checkbox(value=True, description='Refrigerator')
shampoo = widgets.Checkbox(value=False, description='Shampoo')
microwave = widgets.Checkbox(value=True, description='Microwave')
cooking_basics = widgets.Checkbox(value=False, description='Cooking Basics')
bed_linens = widgets.Checkbox(value=True, description='Bed Linens')
fire_extinguisher = widgets.Checkbox(value=False, description='Fire Extinguisher')
air_conditioning = widgets.Checkbox(value=True, description='AC')
heating = widgets.Checkbox(value=True, description='Heating')
first_aid_kit = widgets.Checkbox(value=True, description='First Aid')
self_check_in = widgets.Checkbox(value=False, description='Self Check-in')
num_bathrooms = widgets.IntSlider(min=0, max=10, value=0, description='Bathrooms')
```

Para las variables con distintas opciones únicas utilizaremos un dropdown.

La localización además tiene jerarquía ya que las zonas / ciudades son exclusivas de ciertos estados.

Por esta razón establecernos primero una lista para la localización y sus opciones posibles en ella.

```
location_options = {
 'Hawaii': ['Kauai', 'Maui', 'Honolulu', 'Hawaii'],
 'Los Angeles': ['Unincorporated Areas', 'Other Cities', 'City of Los Angeles'],
          'New York City': ['Brooklyn', 'Staten Island', 'Bronx', 'Queens',
          'Manhattan'],
 'Rhode Island': ['Providence', 'Newport', 'Kent', 'Washington', 'Bristol'],
 'Seattle': ['Other neighborhoods', 'West Seattle', 'Ballard', 'Cascade', 'Capitol
        Hill', 'Queen Anne', 'Rainier Valley', 'Magnolia', 'Beacon Hill', 'Downtown',
        'Lake  City', 'Central Area', 'University District', 'Delridge', 'Northgate',
        'Interbay', 'Seward Park']
        }
```

Establecemos el dropdown de localización.

```
location_dropdown = widgets.Dropdown(
   options=list(location_options.keys()),
   description='Location'
)
```

Establecemos el dropdown de zona (ciudad).

```
neighborhood_dropdown = widgets.Dropdown(
   options=[],
          description='Zone'
)
```

Definimos la correlación entre la localización y la zona, de modo que, al seleccionar una localización, las opciones de la zona se actualicen automáticamente.

```
def actualizador_ubicacion(*args):
    selected_location = location_dropdown.value
    neighborhood_dropdown.options = location_options[selected_location]
```

Establecemos el evento entre ambas variables.

```
location_dropdown.observe(actualizador_ubicacion, 'value')
```

Desplegamos los widgets creados ligando la actualización automática entre los dropdown de localización y zona.

```
display(location_dropdown, neighborhood_dropdown)
actualizador_ubicacion()
```

Creamos los dropdowns de las otras variables con valores únicos como son el tipo de propiedad y el tipo de baño.

```
room_type = widgets.Dropdown(
    options=['Entire home/apt', 'Hotel room', 'Private room', 'Shared room'],
    description='Place Type'
)

bathrooms_text = widgets.Dropdown(
options=['Bath', 'Private bath', 'Shared bath', 'Half-bath', 'Private half-bath',
'Shared half-bath'],
    description='Main Type'
)
```

Ahora que hemos establecido todos los widgets establecemos la definición de precio, que depende de todas las variables del modelo.

```
def predictor_precio(accommodates, bedrooms, beds, minimum_nights,
maximum_nights, smoke_alarm, wifi, kitchen, essentials,
hangers, carbon_monoxide_alarm, hair_dryer, iron, hot_water,
dishes_and_silverware, refrigerator,
shampoo, microwave, cooking_basics, bed_linens, fire_extinguisher,
air_conditioning, heating,
first_aid_kit, self_check_in, num_bathrooms, neighborhood_dropdown,
room_type, bathrooms_text, location_dropdown):
```

Establecemos los checkboxes que hemos creado como int (1 está presenta, 0 no lo está) ya que es el formato dummie que le hemos dado.

```
    smoke_alarm = int(smoke_alarm)
    wifi = int(wifi)
    kitchen = int(kitchen)
    essentials = int(essentials)
    hangers = int(hangers)
    carbon_monoxide_alarm = int(carbon_monoxide_alarm)
    hair_dryer = int(hair_dryer)
    iron = int(iron)
```

```
hot_water = int(hot_water)
dishes_and_silverware = int(dishes_and_silverware)
refrigerator = int(refrigerator)
shampoo = int(shampoo)
microwave = int(microwave)
cooking_basics = int(cooking_basics)
bed_linens = int(bed_linens)
fire_extinguisher = int(fire_extinguisher)
air_conditioning = int(air_conditioning)
heating = int(heating)
first_aid_kit = int(first_aid_kit)
self_check_in = int(self_check_in)
```

Establecemos un diccionario para los datos nuevos, de forma que el dato introducido se condicione a las variables del modelo.

Este paso es fundamental ya que como hemos explicado, el modelo introduciendo un nuevo dato mediante código ha funcionado correctamente, pero para trasladar la entrada de datos desde los widgets a este formato, hay que relacionar los datos de entrada con los datos que espera encontrar el modelo para su procesamiento.

```
dato_prueba = {
    'accommodates': accommodates,
    'bedrooms': bedrooms,
    'beds': beds,
    'minimum_nights': minimum_nights,
    'maximum_nights': maximum_nights,
    'Smoke alarm': smoke_alarm,
    'Wifi': wifi,
    'Kitchen': kitchen,
    'Essentials': essentials,
    'Hangers': hangers,
    'Carbon monoxide alarm': carbon_monoxide_alarm,
    'Hair dryer': hair_dryer,
    'Iron': iron,
    'Hot water': hot_water,
    'Dishes and silverware': dishes_and_silverware,
    'Refrigerator': refrigerator,
    'Shampoo': shampoo,
    'Microwave': microwave,
    'Cooking basics': cooking_basics,
    'Bed linens': bed_linens,
    'Fire extinguisher': fire_extinguisher,
    'Air conditioning': air_conditioning,
    'Heating': heating,
    'First aid kit': first_aid_kit,
    'Self check-in': self_check_in,
    'num_bathrooms': num_bathrooms,
```

```
        'neighbourhood_group_cleansed_Ballard': 1 if neighborhood_dropdown == 'Ballard' else 0,
        'neighbourhood_group_cleansed_Beacon Hill': 1 if neighborhood_dropdown == 'Beacon Hill' else 0,
        'neighbourhood_group_cleansed_Bristol': 1 if neighborhood_dropdown == 'Bristol' else 0,
        'neighbourhood_group_cleansed_Bronx': 1 if neighborhood_dropdown == 'Bronx' else 0,
        'neighbourhood_group_cleansed_Brooklyn': 1 if neighborhood_dropdown == 'Brooklyn' else 0,
        'neighbourhood_group_cleansed_Capitol Hill': 1 if neighborhood_dropdown == 'Capitol Hill' else 0,
        'neighbourhood_group_cleansed_Cascade': 1 if neighborhood_dropdown == 'Cascade' else 0,
        'neighbourhood_group_cleansed_Central Area': 1 if neighborhood_dropdown == 'Central Area' else 0,
        'neighbourhood_group_cleansed_City of Los Angeles': 1 if neighborhood_dropdown == 'City of Los Angeles' else 0,
        'neighbourhood_group_cleansed_Delridge': 1 if neighborhood_dropdown == 'Delridge' else 0,
        'neighbourhood_group_cleansed_Downtown': 1 if neighborhood_dropdown == 'Downtown' else 0,
        'neighbourhood_group_cleansed_Hawaii': 1 if neighborhood_dropdown == 'Hawaii' else 0,
        'neighbourhood_group_cleansed_Honolulu': 1 if neighborhood_dropdown == 'Honolulu' else 0,
        'neighbourhood_group_cleansed_Interbay': 1 if neighborhood_dropdown == 'Interbay' else 0,
        'neighbourhood_group_cleansed_Kauai': 1 if neighborhood_dropdown == 'Kauai' else 0,
        'neighbourhood_group_cleansed_Kent': 1 if neighborhood_dropdown == 'Kent' else 0,
        'neighbourhood_group_cleansed_Lake City': 1 if neighborhood_dropdown == 'Lake City' else 0,
        'neighbourhood_group_cleansed_Magnolia': 1 if neighborhood_dropdown == 'Magnolia' else 0,
        'neighbourhood_group_cleansed_Manhattan': 1 if neighborhood_dropdown == 'Manhattan' else 0,
        'neighbourhood_group_cleansed_Maui': 1 if neighborhood_dropdown == 'Maui' else 0,
        'neighbourhood_group_cleansed_Newport': 1 if neighborhood_dropdown == 'Newport' else 0,
        'neighbourhood_group_cleansed_Northgate': 1 if neighborhood_dropdown == 'Northgate' else 0,
        'neighbourhood_group_cleansed_Other Cities': 1 if neighborhood_dropdown == 'Other Cities' else 0,
        'neighbourhood_group_cleansed_Other neighborhoods': 1 if neighborhood_dropdown == 'Other neighborhoods' else 0,
        'neighbourhood_group_cleansed_Providence': 1 if neighborhood_dropdown == 'Providence' else 0,
        'neighbourhood_group_cleansed_Queen Anne': 1 if neighborhood_dropdown == 'Queen Anne' else 0,
```

```
        'neighbourhood_group_cleansed_Queens': 1 if neighborhood_dropdown
    == 'Queens' else 0,
        'neighbourhood_group_cleansed_Rainier Valley': 1 if
    neighborhood_dropdown == 'Rainier Valley' else 0,
        'neighbourhood_group_cleansed_Seward Park': 1 if
    neighborhood_dropdown == 'Seward Park' else 0,
        'neighbourhood_group_cleansed_Staten Island': 1 if
    neighborhood_dropdown == 'Staten Island' else 0,
        'neighbourhood_group_cleansed_Unincorporated Areas': 1 if
    neighborhood_dropdown == 'Unincorporated Areas' else 0,
        'neighbourhood_group_cleansed_University District': 1 if
    neighborhood_dropdown == 'University District' else 0,
        'neighbourhood_group_cleansed_Washington': 1 if
    neighborhood_dropdown == 'Washington' else 0,
        'neighbourhood_group_cleansed_West Seattle': 1 if
    neighborhood_dropdown == 'West Seattle' else 0,
        'room_type_Entire home/apt': 1 if room_type == 'Entire home/apt' else 0,
        'room_type_Hotel room': 1 if room_type == 'Hotel room' else 0,
        'room_type_Private room': 1 if room_type == 'Private room' else 0,
        'room_type_Shared room': 1 if room_type == 'Shared room' else 0,
        'bathrooms_text_clean_ bath': 1 if bathrooms_text == 'Bath' else 0,
        'bathrooms_text_clean_ Private bath': 1 if bathrooms_text == 'Private bath'
    else 0,
        'bathrooms_text_clean_ shared bath': 1 if bathrooms_text == 'Shared bath'
    else 0,
        'bathrooms_text_clean_Half-bath': 1 if bathrooms_text == 'Half-bath' else 0,
        'bathrooms_text_clean_Private half-bath': 1 if bathrooms_text == 'Private
    half-bath' else 0,
        'bathrooms_text_clean_Shared half-bath': 1 if bathrooms_text == 'Shared
    half-bath' else 0,
        'location_Hawaii': 1 if location == 'Hawaii' else 0,
        'location_Los Angeles': 1 if location == 'Los Angeles' else 0,
        'location_New York City': 1 if location == 'New York City' else 0,
        'location_Rhode Island': 1 if location == 'Rhode Island' else 0,
        'location_Seattle': 1 if location == 'Seattle' else 0,
      }
```

Añadimos a cualquier columna faltante el valor 0, por si el cliente deja alguna opción en blanco, por ejemplo, al no introducir zona, asegurarnos de que no se produce ningún error.

```
    for col in X.columns:
      if col not in dato_prueba:
        dato_prueba[col] = 0
```

Establecemos el dataframe de prueba par a la predicción, que está compuesto por el dato introducido de forma manual mediante el widget.

```
    dataframe_prueba = pd.DataFrame([dato_prueba])
    dataframe_prueba = dataframe_prueba[X.columns]
    nuevo_array = dataframe_prueba.to_numpy()
```

Aplicamos la predicción al modelo random forest creado (rf_model)

En esta ocasión establecemos el resultado como int (número entero) ya que la aplicación de Airbnb no permite establecer decimales en el precio.

```
    prediccion_random_forest = np.array([tree.predict(nuevo_array) for tree in rf_model.estimators_])
    estimacion_inferior = np.percentile(prediccion_random_forest, 45)
    estimacion_superior = np.percentile(prediccion_random_forest, 55)
    estimacion_inferior_int = int(estimacion_inferior)
    estimacion_superior_int = int(estimacion_superior)
    resultado_prediccion = f"Recommended price per night: From \${estimacion_inferior_int} to ${estimacion_superior_int}"
    return resultado_prediccion
```

Establecemos un widget como botón para aplicar la estimación. La aplicación de forma independiente funcionaría sin este botón, simplemente actualizaría constantemente el precio a medida que se alteren datos de la predicción.

Esto implica que se realice una nueva predicción cada vez que el cliente cambie un dato, lo que lo hace subóptimo, tanto a nivel de gasto en procesamiento como para el propio cliente que vería un precio de salida cambiando constantemente de cifras que son irrelevantes para el mismo ya que hasta que no haya introducido todos sus datos son precios sin utilidad y simplemente distraerían al cliente.

```
boton_prediccion = widgets.Button(description="¡Estimate your ideal price!")
output = widgets.Output()
```

Definimos la función que se llevará a cabo cuando se pulse el botón, es decir mostrar el resultado de la predicción.

```
def boton_prediccion_activar (button):
    with output:
        output.clear_output()
        resultado_prediccion = predictor_precio(accommodates.value, bedrooms.value, beds.value, minimum_nights.value, maximum_nights.value, smoke_alarm.value,   wifi.value, kitchen.value, essentials.value, hangers.value, carbon_monoxide_alarm.value,       hair_dryer.value, iron.value,  hot_water.value, dishes_and_silverware.value,      refrigerator.value, shampoo.value, microwave.value, cooking_basics.value,    bed_linens.value, fire_extinguisher.value, air_conditioning.value, heating.value, first_aid_kit.value,      self_check_in.value, num_bathrooms.value, neighborhood_dropdown.value, room_type.value, bathrooms_text.value, location_dropdown.value)

        display(HTML(f'<p style="font-family:Verdana; font-size:20px;">{resultado_prediccion}</p>'))
```

Creamos el evento y establecemos sus características visuales.

```
boton_prediccion.on_click(boton_prediccion_activar)
boton_prediccion.style.button_color = 'lightgreen'
boton_prediccion.style.font_weight = 'bold'
```

```
boton_prediccion.layout.width = '200px'
boton_prediccion.layout.margin = '10px'
```

Señalamos los widgets correlacionados el botón de “predict” y ejecutamos el código.

```
introducir_widgets = widgets.VBox([
    room_type, accommodates, bedrooms, beds, num_bathrooms, bathrooms_text, minimum_nights, maximum_nights, smoke_alarm,
    wifi, kitchen, essentials, hangers, carbon_monoxide_alarm, hair_dryer, iron, hot_water, dishes_and_silverware,
    refrigerator, shampoo, microwave, cooking_basics, bed_linens, fire_extinguisher, air_conditioning, heating,
    first_aid_kit, self_check_in
])
ui = widgets.VBox([introducir_widgets, boton_prediccion])
display(ui, output)
```

El display inicial que verá el cliente, y a su vez el definitivo que buscábamos como objetivo definitivo del proyecto, es el siguiente:

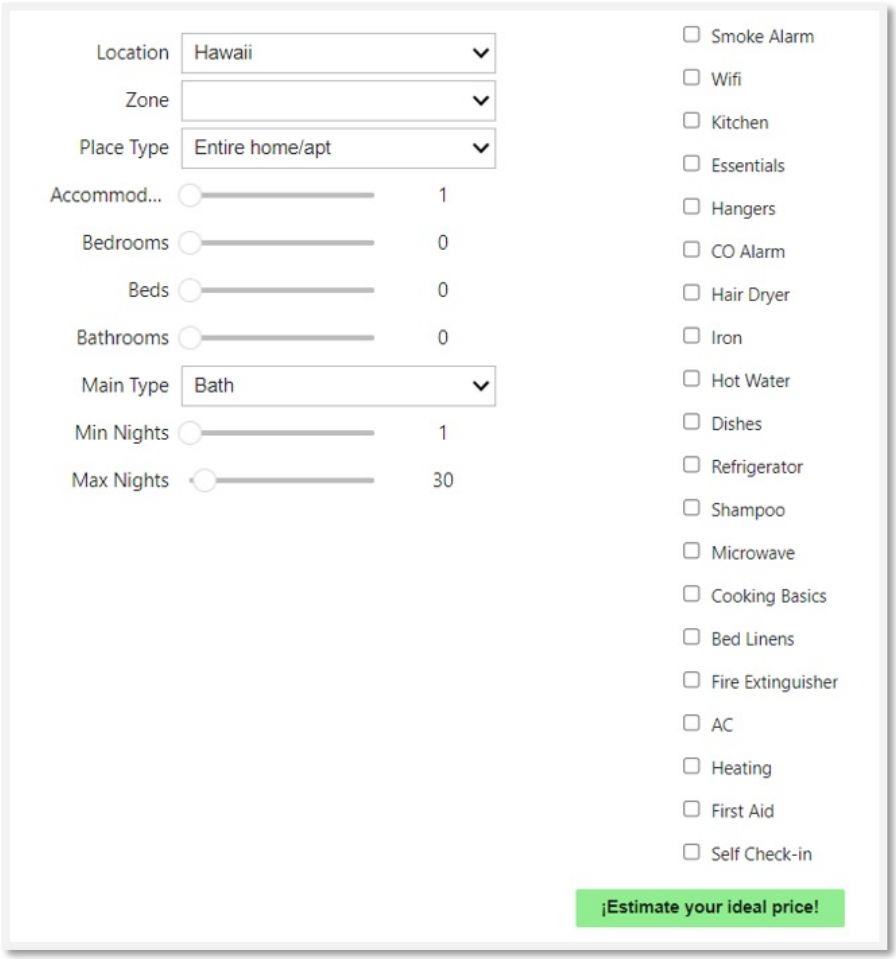

Ilustración 29 - Display del estimador

Con esto queda creado con éxito el estimador de precios. Vamos a realizar distintas pruebas para corroborar que funciona correctamente y que los datos que arroja el modelo tienen sentido.

9.3 Demostración y resultados.

A continuación, podemos visualizar el sistema listo para su uso operativo.

En la introducción manual de datos anterior hemos calculado el precio de una vivienda completa para cuatro personas en Brooklyn – Nueva York. Recibido un precio por noche de $153 a $180.

En este caso vamos a utilizar otro dato con características distintas. con un tipo de vivienda distinto como es un cuarto privado para dos personas en Ciudad de los Ángeles. La expectativa es recibir un precio menor a al obtenido en la predicción anterior.

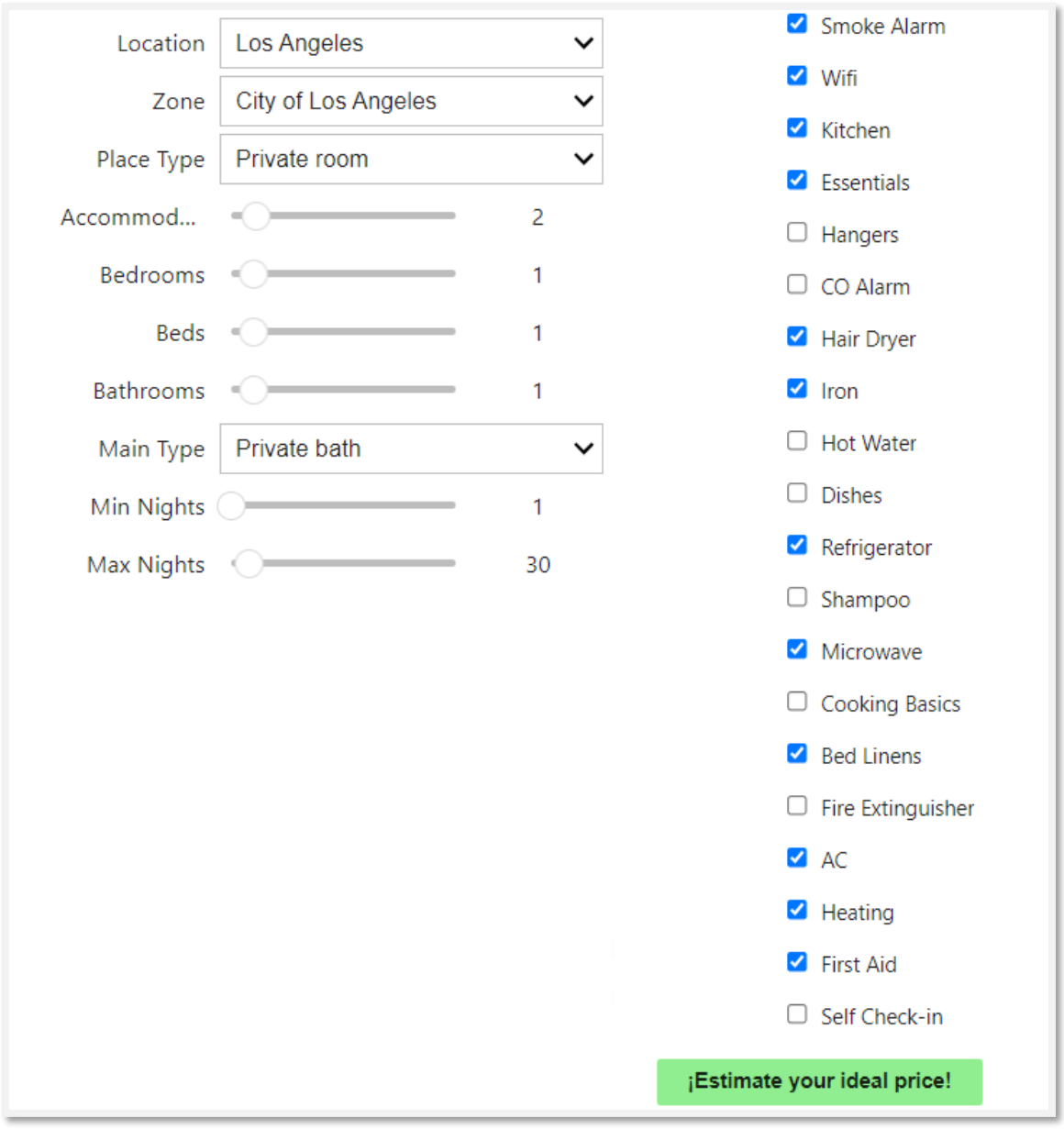

Ilustración 30- Ejemplo estimación – Habitación en Los Ángeles

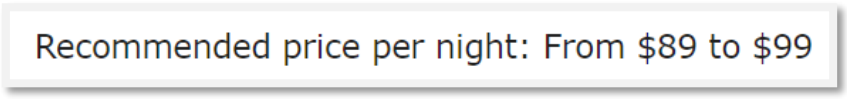

Ilustración 31 - Display de la estimación obtenida al usar el botón para estimar aplicando el dato al modelo.

Hemos obtenido un precio estimado por noche de $89 a $99. De nuevo un dato con sentido, es una buena señal. Vamos a cambiar algunas variables para asegurarnos de que los widgets transmiten la información correctamente al modelo y las estimaciones tienen sentido con respecto a las variables aplicadas.

Introducimos un dato con exactamente las mismas características, pero localizado en Rhode Island – Kent. Con esto comprobaremos que el modelo interpreta correctamente el cambio de ubicación.

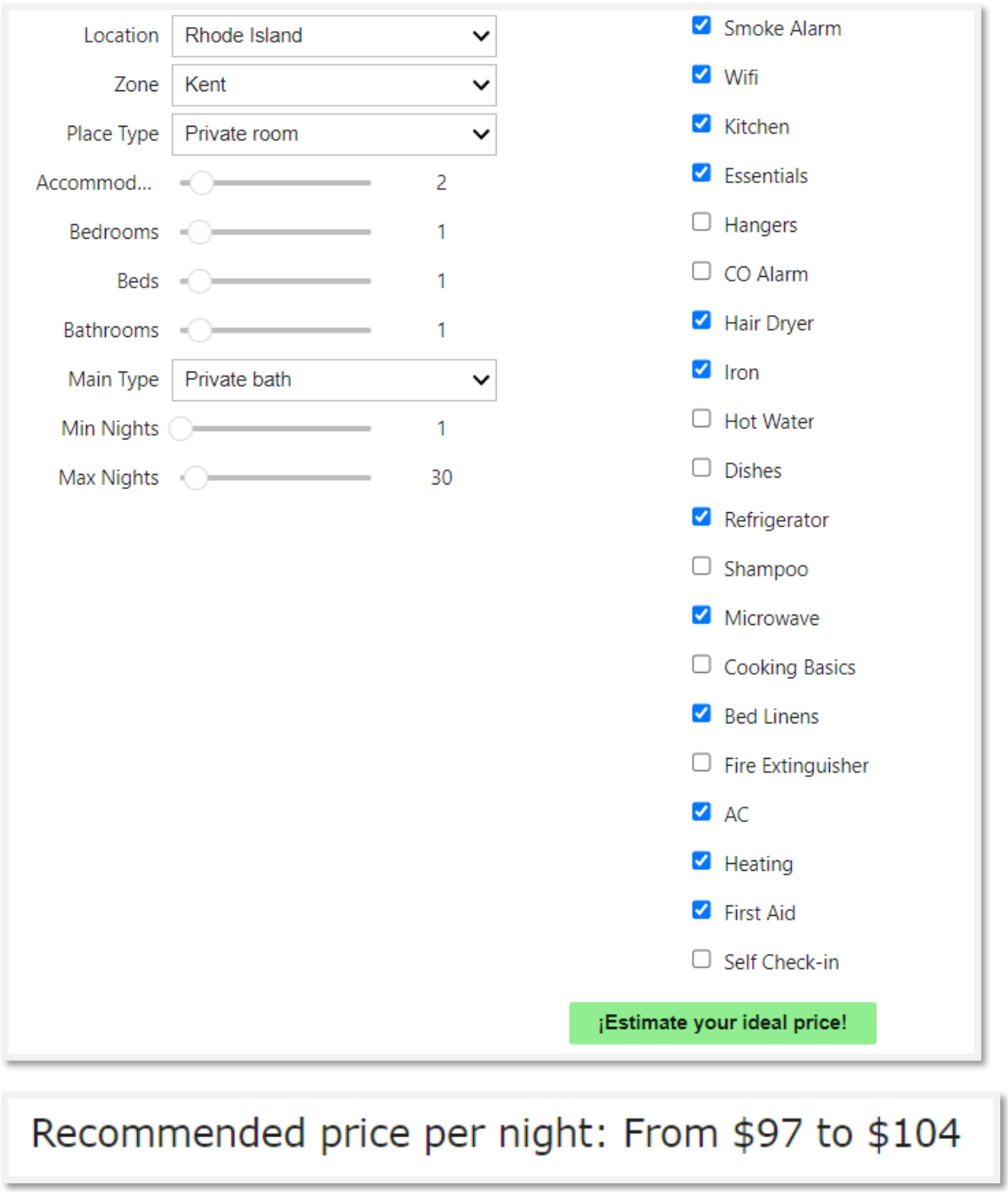

Ilustración 32 - Estimación para habitación en Kent (Rhode Island)

Podemos ver como el precio se ha alterado con éxito. Para una habitación de estas características, los precios son un poco más caros para esta ciudad. Esta estimación de nuevo es satisfactoria ya que la variación de precio es existente, pero se encuentra en unos márgenes de variación esperables para que, a pesar de haber cambiado de ubicación, siga siendo una habitación con exactamente las mismas características para el mismo número de personas.

Haciendo una comprobación del impacto de las variables numéricas. Repetimos el dato de Los Ángeles, pero en esta ocasión en lugar de introducir una habitación para dos personas, introduciremos una casa completa para cuatro personas, dos habitaciones, cuatro camas y dos baños.

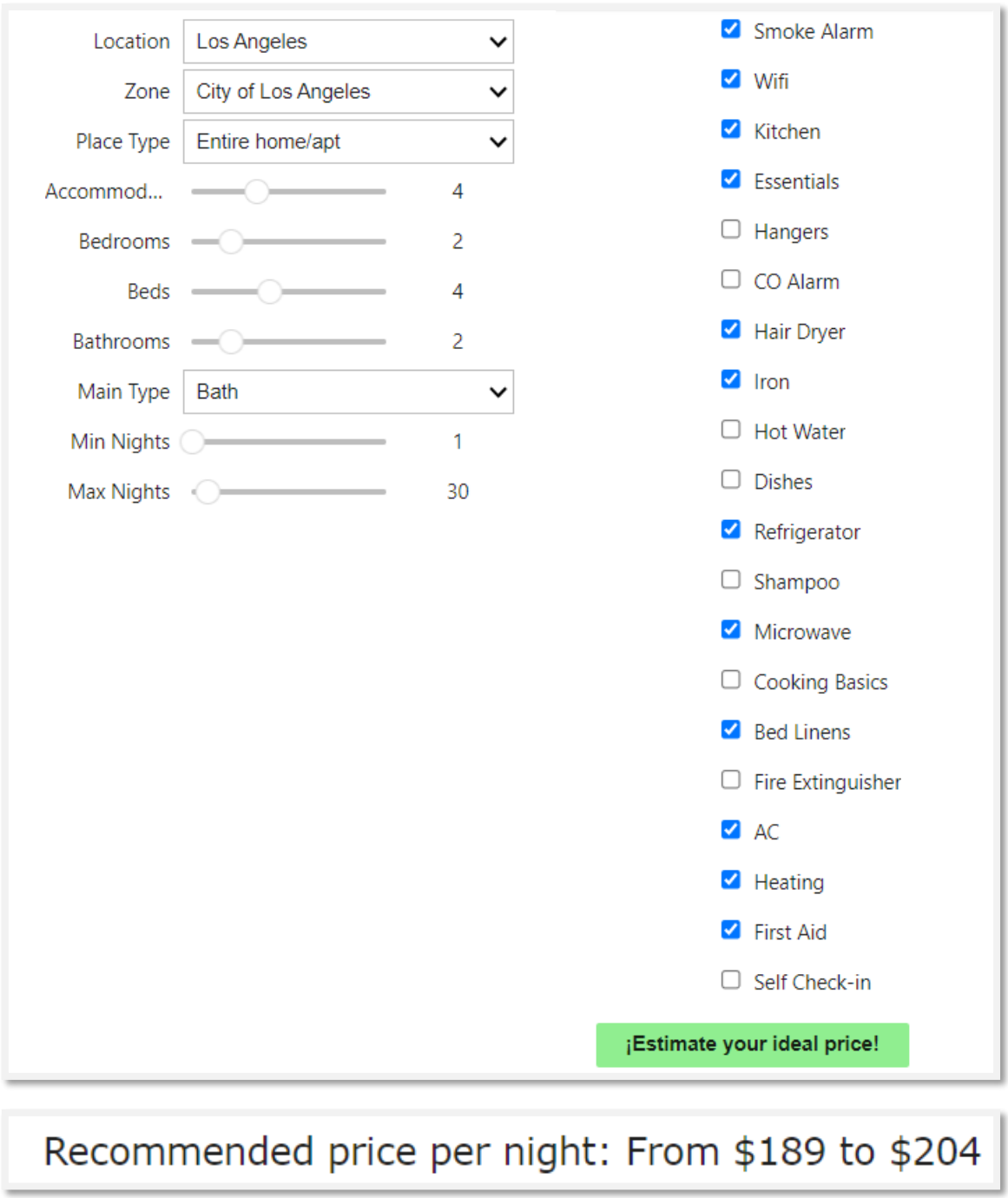

Ilustración 33- Estimación en Los Ángeles, pero con vivienda completa.

El precio ha pasado de $89 a $99 dólares la noche a $189 a $204 dólares la noche que, de nuevo, es el comportamiento que esperábamos dado los datos introducidos.

El estimador se muestra funcional, no obstante, hasta ahora hemos mantenido el valor de las comodidades (amenities) constante, estableciendo la existencia o ausencia de estas de forma aleatoria.

Al establecer los modelos hemos analizado como estas pueden tener un fuerte impacto en la capacidad predictiva del modelo por lo que introducimos exactamente el mismo dato que el anterior, pero cambiando un amenitie.

Vamos a retirar la nevera manteniendo el resto de las variables con el mismo valor. Esto implica que para el modelo se trata de exactamente una propiedad con las mismas características y en la misma zona, pero ahora la propiedad no dispone de nevera para uso personal.

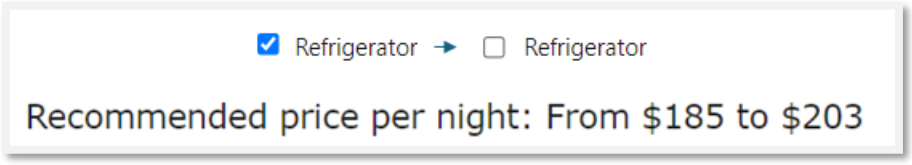

Ilustración 34 – Repetir estimación cambiando una checkbox

El precio mínimo recomendado por noche ha bajado en 4$ y el máximo en 1$. Por lo que retirar la nevera reduce el precio, lo cual tiene sentido.

Hemos establecido pues estimaciones condicionadas a:

- Distintas localizaciones.
- Viviendas de varios tipos.
- Distinta cantidad de inquilinos y habitaciones.
- Diferentes comodidades disponibles.

La variación en el precio al alterar cada uno de estos datos es la esperada, por lo que el modelo funciona correctamente y el estimador lo ejecuta con la fidelidad que estábamos buscando.

Podemos confirmar que resultado es un éxito, por lo que, ahora que hemos cumplido con el último objetivo fundamental del proyecto.
A continuación, vamos a recopilar los resultados obtenidos en cada una de las fases del proyecto para obtener establecer la conclusión definitiva al mismo.

10. Conclusiones

Este proyecto presentaba cuatro grandes objetivos que hemos cumplido durante el desarrollo de este.

- **Recopilación y preparación de datos satisfactoria de una base de datos de Airbnb.**

Se ha importado correctamente el dataframe a emplear, se han tratado y limpiado los datos, habiéndose eliminado variables irrelevantes, al igual que valores nulos y outliers. También se han creado nuevas variables a partir de variables cuyos datos no eran adecuadas para el tratamiento de datos vía Machine Leanring.

- **Obtener las claves para comprender los factores determinantes del precio de alquiler.**

Hemos realizado un análisis estadístico de las variables, entendiendo su naturaleza y estudiando su correlación. Tanto entre las mismas variables como y especialmente con respecto a la variable dependiente precio.

- **Desarrollar del modelo exitoso de Machine Learning.**

Se han establecido 6 posibles modelos (Linear, Ridge, Lasso, Decision Tree, Random Forest y Gradient Boosintg), explicando sus potenciales y obteniendo conclusiones de sus predicciones. Tomando finalmente el modelo Random Forest al ser el que mejor calidad presentaba.

- **Implementar un estimador accesible que proporcione al modelo de Machine Learning los datos necesarios para su predicción.**

Hemos desarrollado un estimador con una interfaz amigable vía HTML en la que el usuario promedio puede introducir las características de su vivienda que son transformadas en un dato que el modelo Random Forest puede procesar para obtener una recomendación de precio.

Por supuesto, el modelo tiene margen de mejora. Tal y como hemos podido comprobar al revisar la salud del modelo y con respecto al coeficiente determinación. Debido a esto hay ciertas combinaciones de amenities que tienen un impacto muy elevado en el precio que podría no considerarse realista, también hay algunas que para ciertas ciudades el hecho de su presencia reduce ligeramente el precio en lugar de aumentarlo.

Para hacer más robusto el modelo y también más universal, se recomienda incorporar más datos en futuras iteraciones. Aunque el scraping de datos de una fecha específica ha proporcionado resultados satisfactorios, estos no son perfectos. Obtener datos adicionales y de diferentes momentos de tiempo podría mejorar sustancialmente la precisión del modelo.

Estimar el precio de una propiedad privada puede ser una tarea compleja ya que hay muchos detalles que no son fáciles de reflejar en los datos básicos de un anuncio. Es decir, hay variables que no son se reflejan en el modelo estadístico, ya sea porque no se requieren para la creación del anuncio o por ser abstractas.

Por ejemplo, pueden existir dos apartamentos en el mismo bloque con cuatro camas y mientras que uno es un piso renovado con cuatro camas en buen estado el otro puede ser un piso sin renovar y mal tratado por los años que puntualiza tener cuatro camas paro dos de esos puestos son de un sofá cama desplegado.

En este ejemplo no solo se incluiría la calidad y localización de las camas si no también el estado de la propiedad. A la hora de alquiler una propiedad, el precio puede condicionarse por las fotos mostradas, que pueden mostrar una casa gris y antigua o una casa lujosa y colorida.

Es crucial disponer del mayor número de muestras posibles de las variables disponibles. Esto ayuda a minimizar el impacto de los factores no contemplados. Un gran volumen de datos permite que el modelo capture mejor las variaciones y tendencias, proporcionando así estimaciones más precisas. Por lo que ampliar el dataset constantemente aplicando las transformaciones para el modelo, debería mejorar su capacidad predictiva a largo plazo.

Aun así, hay variables con un impacto muy claro que proporcionan estimaciones asequibles como es el número de inquilinos y permiten crear modelos como este, con unos resultados satisfactorios.

Este modelo satisface una necesidad real en el mercado y puede ser realmente útil tanto para el uso de particulares como para especialistas que se dediquen a la gestión u asesoramiento.

El tratamiento masivo de datos otorga unos resultados mucho más satisfactorios, precisos y veloces que comparar de forma manual distintas viviendas y, por lo tanto, este estimador, es un activo con unas características muy valiosas. Por lo que, este proyecto, es un indicativo del potencial de la ciencia de datos y su aplicación real para satisfacer necesidades existentes en nuestra sociedad.