IMF Smart Education

MASTER IN DATA SCIENCE AND BUSINESS ANALYTICS ONLINE

In-depth analysis and Machine Learning to establish the right price for your Airbnb

TFM prepared by: Eduard Juan Noordermeer Montoya

TFM tutor: Abel Soriano Márquez

- Seville, June 2024 –

Thanks

I want to take advantage of this small space to thank all the people who have decided to coexist with me while I was completing the master's degree and this master's thesis.

Thanks to my wife for her love while I invested my time in databases.

Thanks to my coworkers for motivating me in the routine.

Thanks to the climbing team for helping me climb literally and metaphorically.

Thanks to Juan Manuel Moreno for his motivation at the beginning of this trip and Abel Soriano Vázquez for his invaluable help to finish it.

And finally, many thanks to all the content creators who have given me hundreds of hours of podcasts to listen to while I learned to be a full-fledged Data Analyst.

Content

1. Project presentation

1.1 Abstract

In the following project, data treatment and analysis is carried out from a Data Scientist perspective in order to create an accurate and reliable price estimator for temporary rental housing.

The data to be processed comes from a large dataset with data obtained via Scraping from the Airbnb website, selection, cleaning and data processing is carried out. This exhaustive collection of data represents the available market offer in a realistic way for correct analysis.

During it, a statistical and graphic analysis of the variables is carried out, with the intention of finding significant patterns and correlations between the characteristics of the properties offered and the rental prices.

Once the data is prepared, an estimation model is established based on Machine Learning techniques, where the model is selected after having analyzed and treated different models.in order to obtain the algorithm with more accurate results and better overall performance. The estimator obtained is capable of predicting the rental price of a home with an accessible, intuitive and easy-to-use user interface, so that no technical knowledge is necessary for its use.

This project offers a valuable tool for potential tenants and owners, also demonstrating the usefulness of Machine Learning techniques in the real market to solve real problems.

1.2 Introduction and background.

Airbnb is a digital platform whose offer is to give individuals the ability to publish and manage accommodations for generally short- and medium-term stays, and in turn, landlord users the ability to rent them in a simple way. Since its founding in 2008, the company has experienced exponential growth, revolutionizing the hospitality and tourism industry. Its innovative business model has transformed the way people travel and stay by providing a viable and attractive alternative to traditional hotel services.

Although there were previously widespread search engines for rentals such as Booking, they focused on collecting offers from lodging businesses. The main attraction of Airbnb lies in its ability to redirect tourism from hotel accommodation to stays in rooms, apartments or private homes, although as we will see in the data analyzed, today they also compile hotel offers in order to cover the largest possible market share. The rentals offered allow clients to have greater independence and privacy than in conventional hotels, hostels, guesthouses and the like.

These particularities allow more personalized experiences for owners, who have access to a large amount of data to choose the best location. Therein lies the potential for data analysis, and that is that as a large amount of information is public, we can discriminate the different factors, understanding their dependencies and correlations.

Since Airbnb is accessible to all individuals who want to get a return on their properties, and to businesses that have different properties to manage, it allows a large housing stock to be available. This adds to the large amount of individual data for each rental

property, the large sum total of properties. The greater the amount of data, the easier it is to establish a reliable algorithm.

In conclusion, Airbnb offers a large amount of data both in its entirety and with respect to the particulars of each offer. Its popularized use around the world and its growth that allows more data to be added to the model over time provides the perfect context to establish a model via Machine Learning, which allows landlords to establish a competitive price for their home.

1.3 Project objectives.

The main objective is to establish a Machine Learning model capable of assimilating and analyzing the factors that influence the rental price of temporary rental housing, in order to obtain accurate predictions, and becominginto a tool accessible to the general public, allowing owners to estimate the appropriate price to publish their properties on rental platforms. To do this, a series of fundamental objectives will have to be met:

1. **Successful data collection and preparation of an Airbnb database.**

To do this we will have to obtain a database with an extensive and varied set of data from the Airbnb platform, ensuring that the data is representative and up-to-date. We will carry out exhaustive data cleaning and treatment of the different variables, both for processing and analysis at a statistical level.

2. **Obtain the keys to understanding the determining factors of the rental price.**

In order to obtain this objective, a variable analysis will have to be completed, which involves identifying and analyzing the key variables that influence the rental price, such as the size of the home, the number of bedrooms and bathrooms, the amenities offered. or the location. It will be necessary to achieve a deep understanding of the interactions of variables, evaluating how they interact with each other through their correlation and how they collectively affect the rental price. These correlations should provide sufficient information to discern the variables truly relevant to the implementation of the model.

3. **Develop the successful Machine Learning model:**

To meet this objective, it will be necessary to evaluate different algorithms to select the most appropriate one for the data presented. Train a model with training and test data, obtaining plausible results and avoiding typical problems such as overfitting. Model evaluation data such as mean square error and coefficient of determination will have to give an acceptable combination of precision and robustness.

4. **Implement an accessible estimator that provides the Machine Learning model with the data necessary for its prediction.**

To do this we will have to develop a user interface that is understandable and in which it is easy for users to enter the characteristics of their property and obtain an estimate of the rental price quickly and easily. This will have to be correctly integrated with the Machine Learning model, providing it with the appropriate input data for a good prediction and ensuring that it gives a corresponding output data.

1.4 Materials and methods.

To complete the aforementioned objectives, Jupyter Notebook will be used as the main software. This is a Python programming environment that will allow us to carry out all the steps of data processing, analysis, visualization and model construction.

This environment is powered by different libraries, their detailed function in the context of the project will be developed as they are used for resolution. The list of the libraries used is as follows:

- **Pandas:**Treatment and analysis of structured data.
- **Numpy**: Operations and management of multidimensional arrays.
- **Ast:**Analysis and manipulation of abstract syntax trees.
- **Counter**(Collections): Count elements in collections.
- **Re**: Treatment of regular expressions.
- **Standardscaler**(sklearn.preprocessing): Data normalization.
- **Matplotlib**: Creating graphs and data visualizations.
- **seaborn**: Improved visualization from matplotlib.
- **Warnings**: Management of warning messages.
- **train_test_split**(sklearn.model_selection): Division of datasets (train and test).
- **LinearRegression**(sklearn.linear_model): Set a linear regression.
- **mean_squared_error**(sklearn.metrics): Calculation of mean square error.
- **ridge**(sklearn.linear_model): Set Linear Ridge Regression.
- **lasso**(sklearn.linear_model): Set Lasso linear regression.
- **DecisionTreeRegressor**(sklearn.tree): Establish regression model based on decision trees.
- **RandomForestRegressor**(sklearn.ensemble): Establish regression model based on random forests.
- **GradientBoostingRegressor**(sklearn.ensemble): Establish regression model based on boosting.
- **ipywidgets**: Creation of interactive widgets.
- **Display**(IPython.display): Display complex output objects.
- **Html**(IPython.display): Create content in HTML format.

In the list of libraries you can see some specialized in establishing models. The models that will be used to later decide the most appropriate one will be the following:

- **Linear regression:**

It is used to predict continuous values by establishing a linear relationship between a dependent variable and one or more independent variables. In our case, we remember, the dependent variable is the price.

- **Ridge and Lasso:**

The Ridge method uses linear regression, but includes an L2 penalty to reduce overfitting and handle multicollinearity.
Lasso works in a similar way but its penalty, known as L1, can reduce coefficients to 0 to eliminate irrelevant features.

- **Decision Tree and Random Forest:**

The Decision Tree model divides the data into subsets according to their common characteristics, developing a structure in the form of a decision tree. On the other hand, Random Forest brings together a large number of decision trees, improving the processing of complex relationships and improving precision with respect to the Decision Tree.

- **Gradient Boosting:**

Creates a model that combines multiple models, performing sequences that correct the previous model, reducing errors and optimizing the algorithm as it iterates repetitions with their respective improvements

Regarding the Dataset.
This comes from the Kaggle platform. This is an open access online platform specialized in Data Science and machine learning.
On this website, users can share datasets for public use, being a very valuable repository for data analysis.

The source directory is titled “Inside Airbnb USA” and its link for consultation is the following: “*https://www.kaggle.com/datasets/konradb/inside-airbnb-usa*”.
The user who shared the dataset is Konrad Banachewicz.
This dataset has been shared under the “ATTRIBUTION 4.0 INTERNATIONAL” license. This license allows you to copy, alter and distribute the material in any medium or format for any purpose, including economic purposes.

This user performed the data scraping using the Inside Airbnb tool, which is also free to use under the same license. “*http://insideairbnb.com/get-the-data/”.*

The data format is given in CSV files, referring to 31 locations in the United States, mainly states and their main cities. For this work the states of Hawaii, Los Angeles, New York, Rhode Island and Seattle will be used.
The variables present will be explained

- **Results.**

The results are presented in the body of the project, through an explanation of data cleaning, study of variables, creation of machine learning models and establishment of the system to enter the data. Indexing each of the fundamental steps. Conclusions will be included to finalize the document.

2. Import of the dataset and initial visualization

2.1 Listing Import

To carry out the analysis, it is necessary to correctly import the files that contain the relevant information for the project. To do this, we establish the file paths of five CSV files that form the data set used. These files contain detailed information about rental properties in different locations.

For clear and efficient organization of data, the imported data set has been set to the name "XY". Although the name itself is not crucial, it helps maintain an orderly structure

and facilitates references during analysis since, within this data set, the sets of variables will later be distinguished:

- Y (Study Variable): This is the target variable that we want to predict, in this case, the rental price of the properties.
- X (Predictor Variables): These are all the other variables that will be used to predict the study variable. They will include the rest of the variables.

First of all, we imported the “pandas” library, a fundamental tool in processing data in DataFrames format in Python. Pandas is especially useful as it allows you to read and write data in various formats, including CSV, facilitating the manipulation and analysis of large volumes of information efficiently:

```
import pandas as pd
```

The file path is established, since the data is downloaded to the personal directory.

```
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 angels 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', }
```

A list is established to store the dataframes, this will be used to combine the mentioned CSVs.

```
Dataframes = []
```

Now we can load each of the dataframes.

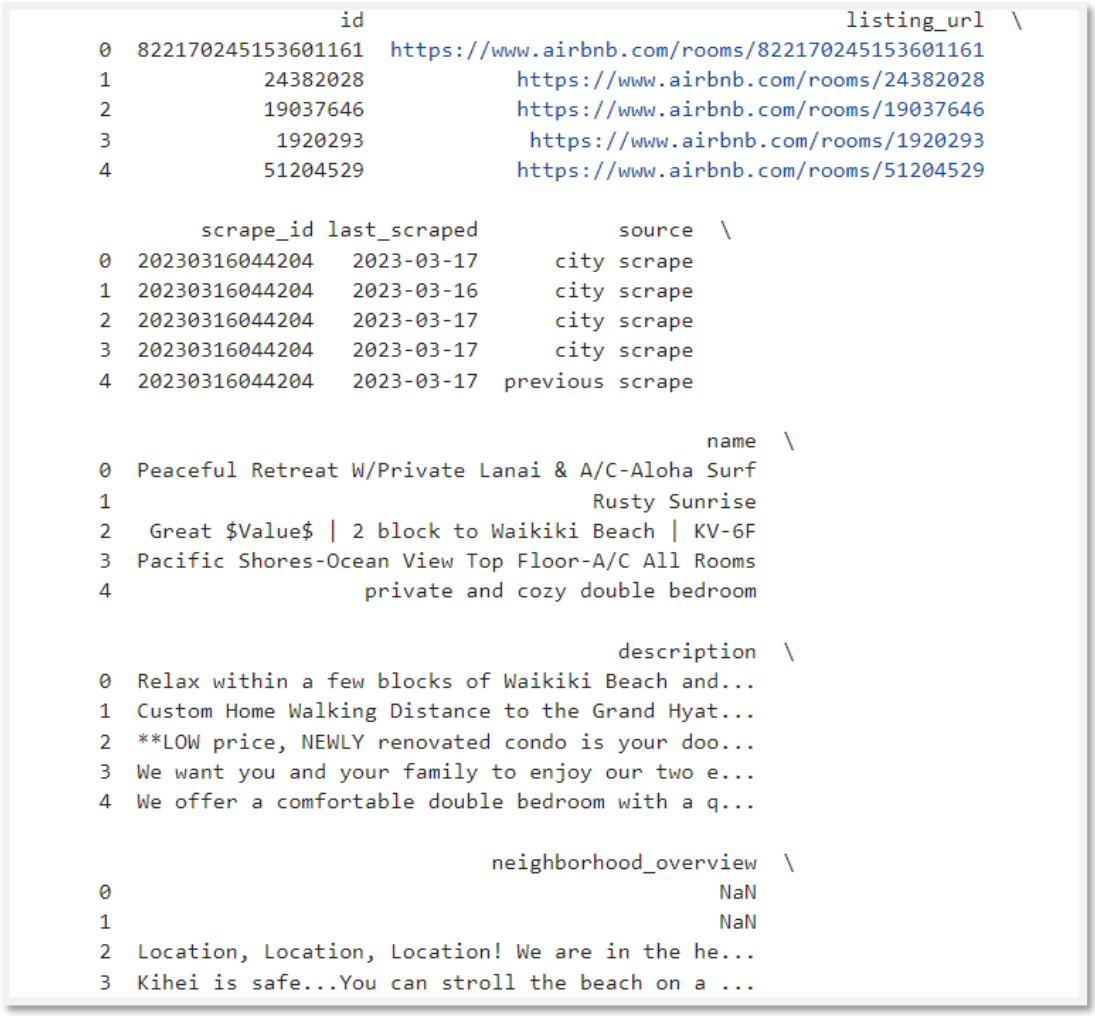
We can also add a column that we are going to call 'location' and it takes into account the place of origin of the data, which has been established when naming the route.

As the files are very large, we add low_memory. This is used to specify to Pandas that it can load files completely at the expense of increased memory usage.

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

Finally, the data is concatenated to establish a single Dataframe with the 5 CSVs included.

```
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 ...	

Illustration1- Initial Display

You can see how the data is imported well in the first instance, but there are many variables in the model so not all of them are shown. We can also see the location column added as the last column of the Dataframe. We are going to analyze the characteristics of the dataframe using “format”. This will allow us to know the number of rows and columns.

```
print(u'- Number of rows: {}'.format(XY.shape[0]))
print(u'- Number of columns: {}'.format(XY.shape[1]))
print(u'- List of variables: {}'.format(list(XY.columns)))
```

- Number of rows: 127260
- Number of columns: 76
- List of variables: ['id', 'listing_url', 'scrape_id', […]]

The DataFrame resulting from the import of the CSV files contains a large amount of data. The quality and usefulness of this data are crucial to the effectiveness of subsequent analysis. A well-structured and accurate data set provides a solid foundation for conducting meaningful analyzes and drawing valid conclusions. However, inaccurate

information, null values, missing data, or inconsistent formats can lead to misdiagnoses and negatively affect analysis results.

Because the DataFrame includes a unique ID column for each record, you can identify and eliminate duplicate records. Duplicates can distort analysis results, so removing them is a critical step to ensure data integrity.

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

Total duplicate IDs: 0

We confirm that the dataframe does not have duplicate data so we proceed to process the variables.

2.2 Initial elimination of variables

Since the dataframe has many variables, it is possible that many of them are not useful for it. The following table presents a very brief description of the reason for discarding both for reasons justifying the model and to understand why a variable may not be relevant due to its characteristics as data.

- **ID:**Once we have verified that they are not duplicates, they are irrelevant since they do not give us information.

- **Scrap data**(listing_url, scrape_id, last_scraped. source): These are scrap data, they help us analysts to know scraping data such as the origin of the data and the date, but it does not provide data for analysis.

- **Descriptions**(name, description, neighborhood_overview): They contain generic and variable text, it is difficult to classify since they use human and subjective writing to describe the property.

- **Host data**(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, neighborhood). They are data about the host, in general they do not have to affect the price of the home, and most of them the client who uses the application would not know how to enter them since they do not have to know the majority of their own characteristics or if they will take actions how to take a profile photo. These variables can be useful to study the type of host with respect to satisfaction to recommend optimizing, for example, response time or using a profile photo, but it is not defining for the price. In a previous model, an attempt was made to take into account, for example, the “host response rate” but its Pearson correlation with the price was very close to 0.

- **Coordinates**(Latitude, longitude) It is given the location, it is a variable that could be useful, however, it is not feasible for the client to use it for the model. The client does not know his coordinates to enter them, so we would have to know how he assigns them according to the street and number of the client, we would need a map with the addresses and coordinates on the map for the transition and that would coincide with how Airbnb presents the cases.

- **Access days**(minimum_minimum_nights, maximum_minimum_nights, minimum_maximum_nights, maximum_maximum_nights, minimum_nights_avg_ntm, maximum_nights_avg_ntm): They are variable with respect to the maximum and minimum number of nights. These are included in the model, the rest is data depending on the history of the home, so the basic values of minimum and maximum days established by the client are sufficient.

- **Availability**(calendar_updated, has_availability, availability_30, availability_60, availability_90, availability_365, calendar_last_scrape). These are the days that the home will be available in the short, medium and long term according to the reservations already made. Since the customer does not yet have reservations, it is not a value that can be entered to determine the price.

- **Characteristic variables**('property_type', 'neighbourhood_cleansed') these variables present a greater categorization regarding the type of property and the neighborhood in which it is located. They are very useful variables that involve the introduction of dozens of "dummy" style variables in the model. These have been applied in the model, but with the amount of data currently available they are not beneficial to the model. This is because, for example, there may be a neighborhood with only 3 homes in it, which vitiates the predictions in a very specific way and in general reduces the predictive capacity of the model, presenting a lower R2 when introduced and reducing its predictive capacity. If we had a broader dataframe, for example, adding new homes over time while taking temporality into account, these variables would improve the model, but this is not the case for this project.

- **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). These are data referring to the site's score, as it is a new home it has not yet been rated by users.

```
clear_columns = ['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_cleanlines s', ' 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'
```

All variables are removed from the list

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

We check the current columns

```
print(XY.head())
print("Columns after deleting:", 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
```

Illustration2- List of variables

We have a total of 11 variables remaining. Let's see what values they present and why they are useful for the model.

- **location:**It is the territory in which the data in the dataframe has been collected. In this case they are Hawaii, Los Angeles, New York, Rhode Island and Seattle. These dataframes not only include the state capital but also other big cities in them.
- **neighborhood_group_cleansed:**Contains the specific city within the location. It is important to point out that in the context of the United States, these cities are adjacent, so they have different characteristics, functioning in a similar way to neighborhoods. For example, New York distinguishes Brooklyn, Staten Island, Bronx, Queens and Manhattan. All adjacent and with a high population density.
- **room_type**: Distinguish between complete houses, private rooms, shared rooms or hotel rooms.
- **accommodates:**It is the number of guests for which the facility is prepared.
- **bathrooms_text:**Includes the number and type of bathrooms present.
- **bedrooms**: It is the sum of available rooms.
- **bed**: They are the number of beds in the site. It should be noted that it does not match the number of accommodations since there may be double beds in the property.
- **amenities:**It is the list of amenities present in the facility.

- **price:**Price per night established for the home.
- **minimum_nights:**It is the minimum number of nights that the rental is allowed.
- **maximum_nights:**It is the maximum number of nights that the rental is allowed.

We review how the variables are categorized for treatment.

```
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
```

Illustration3- Types of variables

Here we can see the first indication that it is necessary to treat the variables for their correct use. For example, our target variable, price, is cataloged as an object, therefore, it is recognized as a column with text and not numeric.
We review the values of the numerical variables in order to obtain a general idea of the possible data and identify possible outliers, a matter that we will discuss soon.

```
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

Illustration4- Statistical values

We look at the values of the object variables.

We get the list of columns and display the unique values for each of them.

```
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']
```

Illustration5– Variables with problematic text

There are certain data that attract attention.
We found problems in the price since for the dollar symbol it is detecting thousands of isolated text values.
The bathrooms are given as text by one part number and another type.
Amenities are text strings that concatenate different amenities in the property.

We are going to treat these variables so that they are correctly usable.
We perform a first cleaning of null values before treating the variables. This is done because, for example, we may want to transform a text variable into a numeric variable, or as dummies, but presenting values breaks the rule used for their treatment.

```
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
```

Illustration6- Null values

We can confirm that there are null values, especially regarding the bedrooms. It is important to eliminate them, since the number of bedrooms is also a fundamental variable to determine the price.

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

We proceed to treat the variables for their correct use.

3. Transformation of variables

3.1 Transform Price

The price is given as an object variable since in addition to the figure, it has the $ symbol, so we must eliminate it and establish the variable as a numeric one, in this case a January number. Prices are generally given as non-whole numbers with decimals, but Airbnb does not allow cents.

To do this we will replace the $ symbol with a blank value, and we will default the type of variable. For this we use “np”, a function from the NumPy release that provides tools for numerical processing and calculation.

```
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
```

Illustration7- Corrected prices

We can verify that the price has been set correctly as an integer.

3.2 Transform Amenities

"Amenities" are the additional comforts and services that a property offers its guests, such as wifi, air conditioning, heating, kitchen utensils, among others. These amenities can have a significant impact on the rental price of a property as they add value and enhance the guest experience. Therefore, it is essential to analyze this variable in depth to understand its influence on prices.

The main challenge when dealing with the "Amenities" variable is the wide variety of possible values. Each property can offer a unique combination of amenities, and the list of all available amenities is extensive. Additionally, the data is usually stored as a concatenation of text in a single column, which complicates its direct analysis.

We will use the “ast” and “counter” modules to facilitate data processing.

Ast allows you to work with abstract syntax trees so you can decompose the list of amenities. On the other hand, counter allows you to count all the elements, in this case each of the independent amenities in the text line.

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

We convert the text string into a list and apply it to the amenities variable.

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

We obtain a list of all the amenities by counting the number of times each one appears.

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

We set the count as a dataframe and sort it in descending order to see the values that appear the most.

```
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("Count of all 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]
```

Illustration8- Types of amenities

We can confirm that there are more than 20,107 unique independent values. Most of these values are individual, that is, they are amenities that appear in very few properties. This high number of unique values presents a challenge for the analysis, as including all amenities in their original form could overload the model and make the results difficult to interpret.

As a solution, two approaches have been taken to process this data:

On the one hand, identify and select the most common amenities. The frequency of each amenity will be analyzed to identify the most common ones, which are also easily recognizable by customers. Only the amenities that appear most frequently will be

selected as individual variables in the model. This allows the number of variables to be significantly reduced, while maintaining a large part of the relevant information.

On the other hand, the total calculation of Amenities since in order not to lose the information provided by the less common amenities we can discern their relevance by the mere fact of being present in the total data. This approach allows you to capture the added value of having more amenities, without needing to analyze each one individually.

This treatment allows us to reduce the complexity of the model while preserving its relevant information.

We establish the 20 most frequent amenities. Clients will be able to individually select which ones are on the property and which ones are not.

```
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} times")
```

```
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
```

Illustration9- Most frequent amenities

To leave these variables correctly established without needing further treatment, we are going to establish them as dummies, so that if the home has comfort it presents a 1 and if it does not have it a 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()
```

Illustration10- Dummies applied

```
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
```

Now we are going to create the total amenities variable, separating the text by its delimiter and counting the number of values

We create the 'total_amenities' column that contains the total number of amenities for each row, obtained using 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
```

Illustration11- Successfully established the total number of amenities

Now we have the most common variables as dummies and the sum of total amenities as a numerical value, so we can eliminate the original amenities column, which is pure text without treatment.

We can delete the original amenities column since as raw text it no longer has any value for the model.

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

3.3 Transform Bathrooms

We study the possible values for your treatment.

```
possible_values = XY['bathrooms_text'].unique()
print("Possible values of bathrooms_text:")
print(possible_values)
```

```
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']
```

Illustration12- Possible bathroom values

The variable bathrooms is presented as a number with text. In general we can refer to:
-bath: Generic bathroom, without specifying.
- shared bath: If shared with more guests.
- Private bath: If it is exclusive for the lessor.
- half-bath: When it is a simple bathroom, generally small with only a toilet and sink.

This explains the existence of decimal values, since on Airbnb the half baths take a decimal value of 0.5.
We can then divide by the variable to obtain two new ones. On the one hand, a categorical object variable that distinguishes the type of bathroom or bathrooms and, on the other, the total number of them.

We import re that works to find patterns in text strings.

```
import re
```

We define the function to extract the text on the one hand and the number on the other. We find numbers in the text, separate them from it and eliminate the numbers in the text.

```
def extract_number_bathrooms(text):
    number = re.findall(r'(\d+(\.\d+)?)', text)
    num_bathrooms = float(number[0][0]) if number else 1
    clean_text = re.sub(r'\d+(\.\d+)?', '', text)
    return num_bathrooms, clean_text
```

We apply the defined function to the column, distinguishing the number of bathrooms and the already cleaned text.
We remove the bathrooms_text column since the raw text is of no interest to us.

```
XY['num_bathrooms'], XY['bathrooms_text_clean'] =
zip(*XY['bathrooms_text'].apply(extract_number_bathrooms))
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]

Illustration13- New bathroom variables

We have created a new numeric variable and object. It is important to review what categories now exist as an object to make sure they are correct and not create too many dummies soon. For this we collect the unique values.

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

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

Illustration14- Text values (Bathrooms)

As we can see, the text distinguishes the type of bathroom in singular and plural. This is not necessary since the number of bathrooms is established by the new variable that quantifies them, so we are going to establish the plural categories as their equivalent in singular.

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

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

Illustration15- Treated text values (Bathrooms)

We confirm that the possible categorical values for bathrooms have been correctly updated.

3.4 Convert categorical variables to numerical variables

Training machine learning models generally requires that the variables used be numerical. This is because most Machine Learning algorithms, including the 4 models that will be used in this project, are designed to process numerical inputs. They do not process categorical data of the “object” type, that is, text entries. This implies that the variables will have to be transformed so that they are understandable for the model by giving them a numerical value.

There are different techniques to carry out this transformation. To understand the reasoning behind the decision to use one mode or another, we are going to consider the different most popular options:

- **Label Encoding**establishes a number for each category, generally giving time or hierarchical order to the variables.
- **One-Hot Encoding**creates a new column for each possible category in a variable. It is assigned a binary value from 0 to 1, with 0 meaning the non-existence of the characteristic and 1 meaning its existence. This transformation does not give hierarchy to the data, but in exchange it increases the dimension of the data and is more difficult to process and understand for various Machine Learning models.
- **Target Encoding**gives a value to the variable according to its average appearance with respect to a target variable.
- **Frequency Encoding**It replaces each of the categories with its frequency, that is, it gives it a numerical value that is equal to the number of times the variable appears among the data. It is a simple and versatile system but it is not good at capturing complex relationships.

First of all, we are going to analyze the possible categories of the object variables. We establish a list of these for analysis, looking at their possible independent values.

Depending on the data categories, it is necessary to decide what type of transformation is most useful to obtain the most useful information possible without overloading the capacity of the model.

```
object_columns = [ 'neighbourhood_group_cleansed', 'room_type',
'bathrooms_text_clean', 'location']
```

```
    for columns in object_columns:
    print(f"Unique values in '{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']
```

Illustration16- Values of dummy variables

We can appreciate how values make sense. Since the objective of the model is to obtain the effect on the price variable of many variables, we anticipate complexity. At the same time, it is difficult to justify a hierarchy in the different variables.

Qualifying the neighborhoods and the location in a hierarchical manner would be inappropriate. This could be done if in an external analysis we could qualify the neighborhoods under a criteria model with a quality level, but it is not appropriate for this model.

In turn, the type of room and bathroom with respect to its size or general social consideration. However, in these cases it is also subjective, since it is difficult to evaluate whether a hotel room or a shared room is better. Also, a room could technically be shared. Likewise, a person may value a private half-bathroom more than an entire shared bathroom.

Given that in data cleaning we have eliminated the categorical variables with a very high number of possible variables so as not to saturate the system or add variables with little representation in the data, we can apply One-Hot Encoding to the 4 variables present without fear to create a large number of new variables.

It is important to point out that the values of neighborhood_group_cleansed depend on the previous value of location, since each one is exclusive to a specific location. This will be taken into account in the execution of the model.

So we take dummies. This will create a new variable for each possible category and set the value to 1 if present and the value to 0 if not. These are exclusive from each other so a piece of data will only have a value for one type of neighborhood, room type, bathroom and location.

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

We reviewed the type of variables, now there are a total of 77 due to the inclusion of dummy variables, both the recently created ones and those corresponding to the 20 most frequent amenities previously created.
There are no longer object variables, they all have the int64 or float64 format, that is, numeric, whether they are integers or not.
We can take a new look at the general data. We can review the given values and in turn confirm that the total of 77 variables are shown.

4. Data processing and cleaning

Next, we are going to visualize the general numerical data of the dataset. This visualization will allow us to obtain a general idea of the values present in the different variables and detect possible anomalies or outliers that may affect our analysis and the results of the predictive model.

The nature of our dataset means that outliers are relatively rare because many of the values are continuous and fall within a set range. Furthermore, one of the advantages in the organization of the data is that the actual entry of data to create an Airbnb listing is limited to avoid anomalous or unrealistic values that could both confuse the landlord looking at the listing and harm the listing filters themselves. search for the application. This means that, although data treatment and cleaning remains essential, in the first instance the elimination of anomalous data, when reduced, does not imply losing a large amount of data in the dataset.

For example, there are variables with a set range. Variables such as bathrooms and beds have a well-defined minimum range. We do not expect to find negative numbers in these variables. The minimum number for bathrooms could be set to 0, while for beds the reasonable minimum is 1.

However, there are variables in our dataset that do not have a clear floor and ceiling, which makes it more likely to find anomalous values. A key example is our study variable, the rental price (price), which can present both very high and very low anomalous values.

Some landlords may set unrealistic minimum prices, such as 1 monetary unit, to promote the rental, even though this is not the actual price or available for it in practice. Sometimes, for example, they promote themselves in this way to appear first in a price filter and then agree on a price privately.
Similarly, some landlords may set extremely high maximum prices, such as 999,999 monetary units, to maintain the age of the listing and prevent the property from being rented, due to reasons related to the platform's algorithm.

These extreme values must be identified and treated appropriately to prevent them from distorting the results of the predictive model.

Other variables that may present significant outliers are those related to accommodation capacity and length of stay.

The accommodates, which are the accommodation capacity, although they have a minimum number of people of 1, their maximum number can present unusually high values. In our overview of the statistical numbers, we note that the maximum value for "accommodates" is 16, which may be reasonable for large properties or villas.

On the other hand, for the variables bedrooms, beds, minimum_nights and maximum_nights we find outliers in the ceiling values. In the minimum values it does not make sense to eliminate the smallest values since a single bed or night is a realistic value, however, in the overview we have been able to see how in contrast, while the maximum number of accommodations was 16, the number maximum number of beds is 50. This is an indication of possible values that are either incorrect or marginal that do not serve for a realistic general prediction.

To ensure the quality and reliability of our analysis and predictive model, we will implement an outlier treatment in which we eliminate the smallest values (in which we have mentioned, it is necessary) and high ones, avoiding anomalous figures.

Later we will graphically verify if there are still outliers that could worsen the quality of the model.

We establish the lower decile of the price and both the lower and upper decile of the rest of the variables.

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

We count the amount of data present in each one.

```
lower_decile_count = (XY['price'] < lower_decile_price).sum()
decil_superior_count = (XY[['price', 'accommodates', 'bedrooms', 'beds',
'minimum_nights', 'maximum_nights', 'num_bathrooms']] > top_decile).sum()
total_decile_superior_count = decile_superior_count.sum()
XY_prefilter = XY.copy()
```

We eliminate the values.

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

We count the total number of values removed.

```
deleted_values = len(XY_prefilter) - len(XY)
print(f'Total deleted values: {deleted_values}')
```

Total values removed: 35852

The dataframe is treated, this is its general form.

```
print(u'- Number of rows: {}'.format(XY.shape[0]))
print(u'- Number of columns: {}'.format(XY.shape[1]))
print(u'- List of 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	

Illustration17- General context of the current Dataframe

5. Establishment of the target

Now that we have treated and cleaned our dataset, we can define the target variable that we want our Machine Learning model to predict. In our case, this target variable is the rental price, represented in the dataset as "price".

Establishing a variable as a target, explicitly, helps the Machine Learning model to define the object to predict and focus on understanding the relationships of the model variables with the specific variable.

In the same way, these models can present results that allow their quality to be evaluated and having a defined target variable, allows establishing and comparing characteristics such as the evaluative values of the quality of the model such as the mean square error of the model.

From this point, we will structure our XY dataset so that it is clearly divided into two parts: X and Y. Part X will contain the set of predictor variables, while Y will represent the target variable that we want to predict.

We extract the "price" column from DataFrame XY and store it in a new DataFrame called Y.
This action allows us to have a DataFrame dedicated exclusively to the target variable, thus facilitating the modeling and evaluation process.

We remove the "price" column from XY. This ensures that our DataFrame X contains only the predictor variables, eliminating any possible information leakage during model training.

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

By performing this operation, we are left with a clean and structured DataFrame that we can use to feed our Machine Learning model.

We also created a normalized version of the X dataframe for later use in graphical analysis. Normalizing the variable sets the values on the same scale between 0 and 1, making it easier to compare numerical values.

```
X_normalized = (XX.mean())/X.std()
```

6. Graphic analysis and correlation of variables

6.1 Boxplot

First, let's proceed with displaying a boxplot of the numerical values in our model. This visualization will help us identify if there are still anomalous values that could affect the accuracy and robustness of our model. The detection and treatment of outliers is essential to ensure that our Machine Learning model is not negatively influenced by outlier data.

To make it easier to compare the variables and ensure that they are all on a common scale, we will use the StandardScaler from the sklearn.preprocessing module. Normalization adjusts the variables so that they have a mean of 0 and a standard deviation of 1 and helps with their comparison, especially at a visual level.

To visualize the results of the normalization and detect possible anomalous values, we will use the matplotlib.pyplot and seaborn libraries. As mentioned in the list of methods for the project, they are libraries for creating graphics.

Boxplots are a useful tool to visualize the distribution of numerical variables and detect outliers. Here's how to create a boxplot of the normalized variables:

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

We establish the numerical columns

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

We apply data normalization.

```
scaler = StandardScaler()
XY_normalized = scaler.fit_transform(XY_selection)
XY_normalized = pd.DataFrame(XY_normalized,
columns=numeric_columns)
```

We create the box plot

```
plt.figure(figsize=(15,7))
ax = sns.boxplot(data=XY_normalized)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.title(u'Boxplot with normalized numeric variables')
plt.ylabel('Value')
plt.show()
```

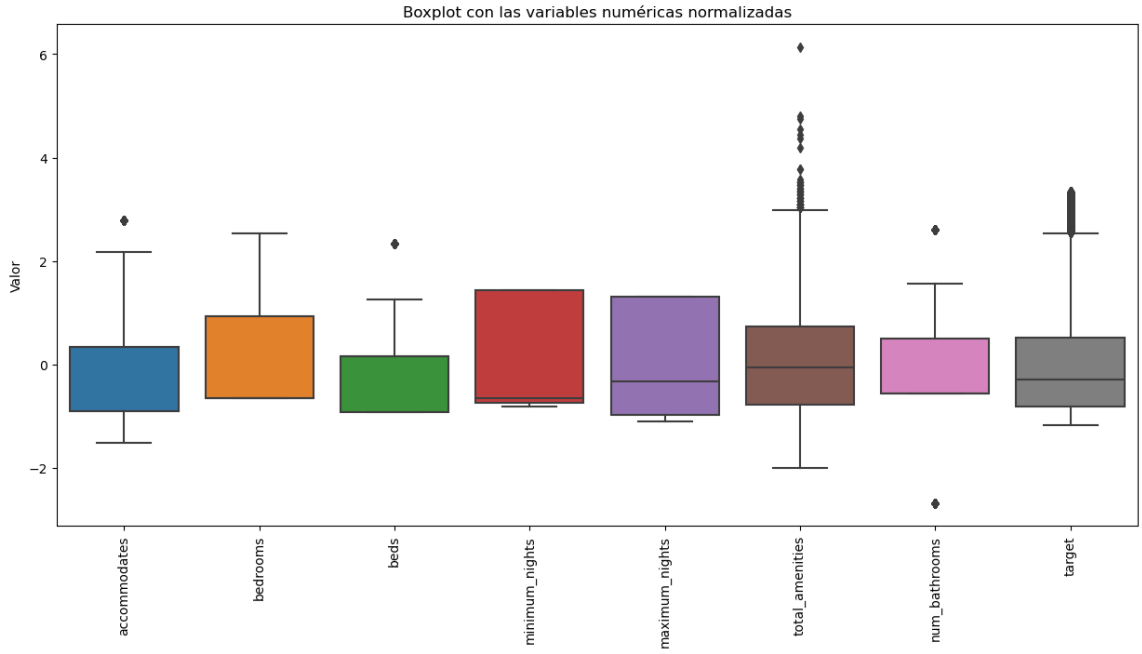


Illustration18– Boxplot

The boxplot provides us with a clear and effective visualization that allows us to confirm that the number of outliers present in our data set is reduced and residual. This graphical tool makes it easier for us to visually identify any outliers, showing that the majority of the data is within the expected parameters and forming a consistent block. The absence of numerous outliers suggests that our data cleaning and normalization processes have been successful and efficient.

6.2 Histogram of distributions

After having verified that the elimination of outliers has been effective and that it makes sense in our analysis, the next step is to visualize the graphic distribution of the variables to better understand their behavior and characteristics.

To do this, we will use histograms, which allow us to observe how the values of the variables are distributed and the frequency with which they occur.
It also allows asymmetries to be detected, although, with the data processing carried out, these should not be a problem.

We import and apply “warnings” since the system we will use will be discontinued in future versions of Python via Jupyter Notebook, but currently they work correctly.

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

We establish the figure and iterate the columns of interest. We already established the list of numerical columns in the previous graph.

```
plt.figure(figsize=(20, 15))
```

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

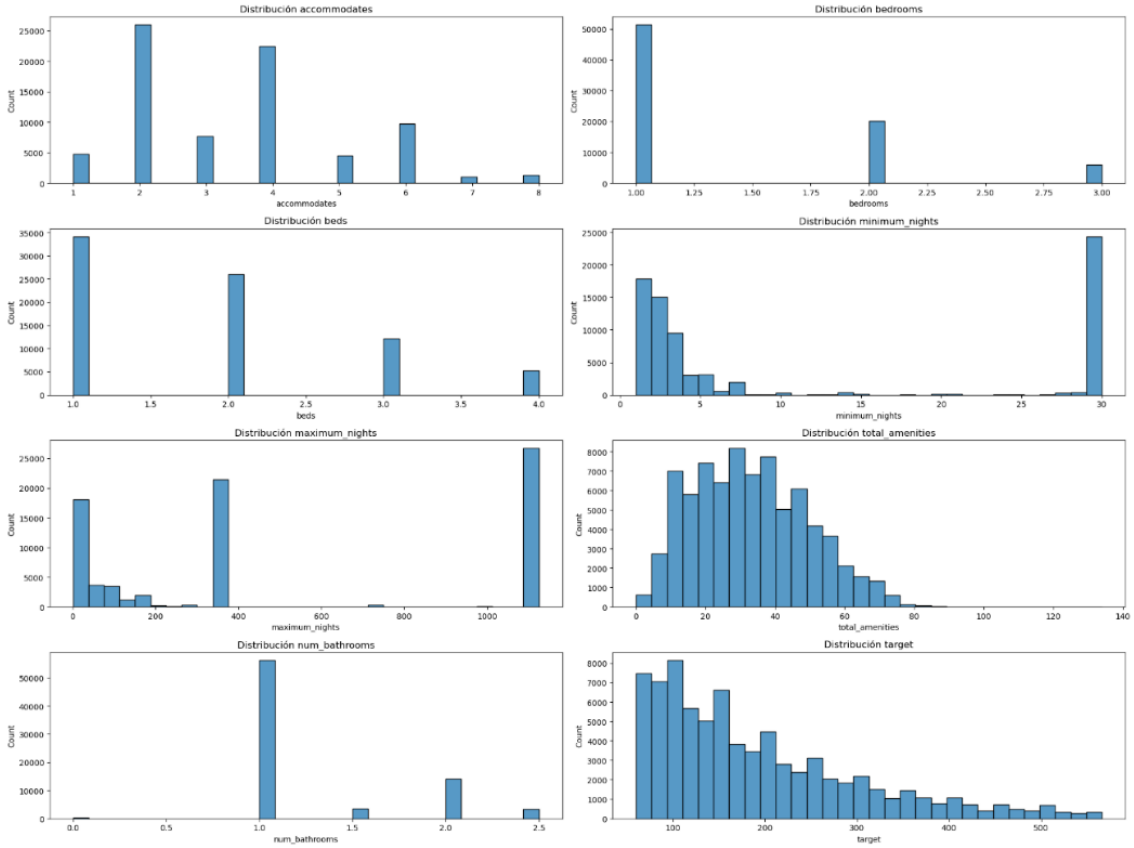


Illustration19- Distribution histogram

The variables have a distribution that makes sense for their characteristics.

“Accomodates” has the most frequent values of 2 or 4 tenants.

Rooms is most frequently 1, since private, shared or hotel room rentals add to this figure.

The maximum and minimum number of nights have a normal progression with the peculiarity that the maximum and minimum number of nights that the application allows, that is, establishing a minimum of 30 days (1 month) of rental or the maximum of 365 or 1255 days, that is, without limits are very present values.

Finally, the most common value for bathrooms is only one, and the total amenities and the total price tend to present fewer and fewer values since there are more properties of reduced sizes and prices and few large properties with luxury values.

6.3 Pearson correlation

The correlation of variables allows us to understand how one variable can influence another. Conceptually, correlation measures the strength and direction of the linear

relationship between two variables. This analysis is crucial to identifying meaningful relationships and understanding how changes in one variable can impact another.

A positive correlation means that as one variable increases, the other does too. A negative correlation implies that as one variable increases, the other decreases.

First, let's analyze the numerical variables.
Let us again point out the numerical columns and calculate the correlation matrix through Pearson.

```
XY_numeric = XY[numeric_columns]
correlation_matrix = XY_numeric.corr(method='pearson')
```

We extreme the exact numerical correlation with target and show it.

```
target_correlation =
correlation_array['target'].drop('target').sort_values(ascending=False)
print("Numerical correlation with respect to 'target' (ordered from greatest to minor):")
print(target_correlation)
```

We also made a correlation table that helps us visualize not only the correlation with target but also between the variables.

```
n_ticks = len(numeric_columns)
plt.figure(figsize=(12, 10))
plt.xticks(range(n_ticks), numeric_columns, rotation='vertical')
plt.yticks(range(n_ticks), numeric_columns)
plt.colorbar(plt.imshow(matrix_correlations, interpolation='nearest', vmin=-1., vmax=1., cmap='coolwarm'))
plt.title('Pearson correlation matrix')
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
```

Illustration20- Correlation of numerical variables

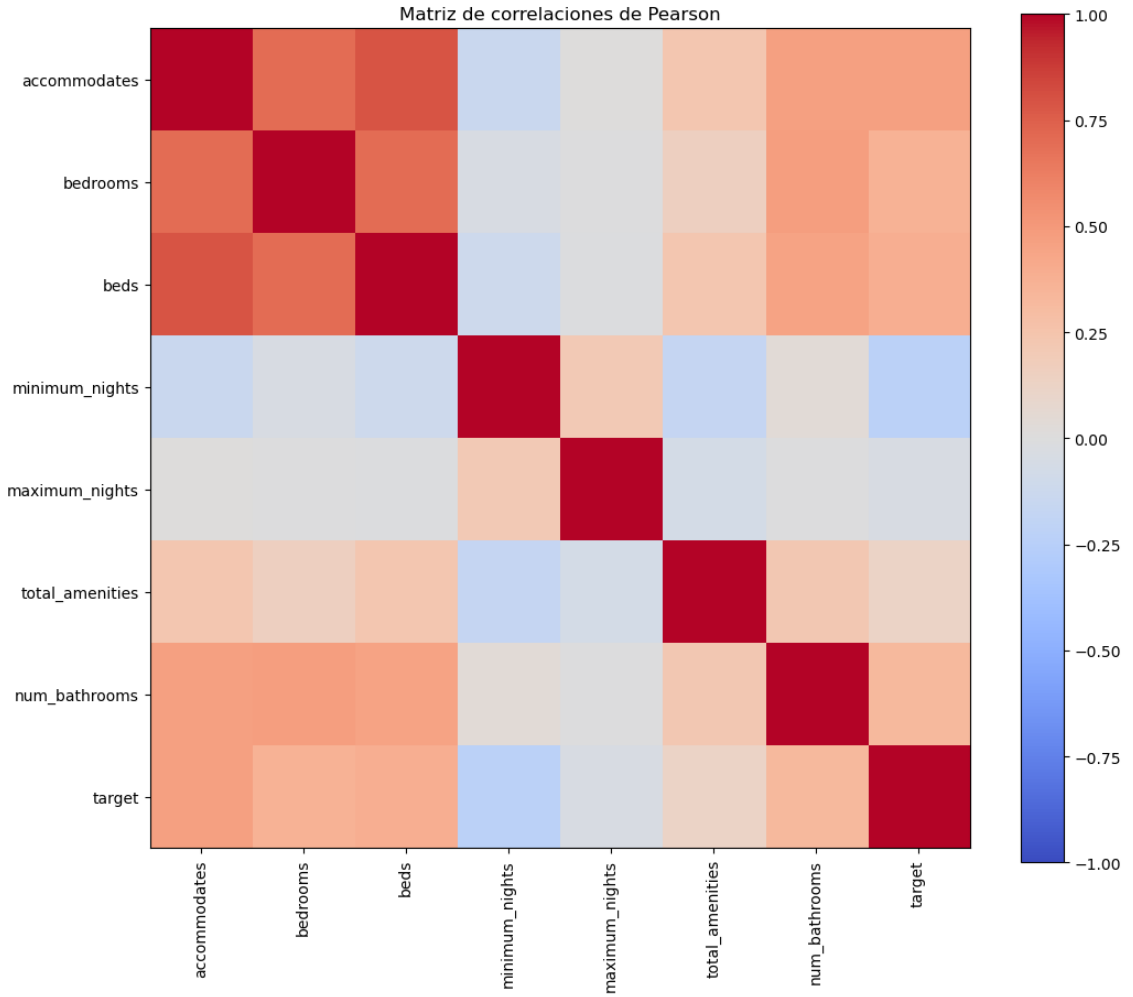


Illustration21- Correlative heatmap

By performing correlation analysis, we found a logical trend in the relationship between our predictor variables and the target variable, price. We observe that increases in the number of rooms, beds, bathrooms, tenants or amenities tend to increase the rental price. This trend is intuitive, as properties with more rooms and amenities typically offer greater value and can therefore justify higher prices.

On the other hand, variables such as the number of minimum or maximum nights required to rent the property show an inverse correlation with the price. Specifically, setting a higher number of minimum or maximum nights tends to reduce the price. This is because a greater number of minimum nights imposes a restriction on rental flexibility, which may discourage some potential tenants looking for shorter stays. Likewise, an upper limit on the number of nights stayed can limit demand for the property, as renters may prefer options with greater flexibility in length of stay.

In addition to the numerical variables, the categorical variables that we have transformed into dummies also present correlations with the price. To study its impact we are going to calculate the correlation with respect to these variables.

We establish the dummy columns with the target (price) and filter the column.

```
columns_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[dummies_columns]
```

We calculate the Pearson correlation, this time there are many values so we are going to show only the most influential ones on display.

```
correlation_matrix = XY_dummies.corr(method='pearson')
target_correlation =
correlation_array['target'].drop('target').sort_values(ascending=False)
print("High positive correlation with price:")
```

```
print(target_correlation.head(2))
print("High negative correlation with price:")
print(target_correlation.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
```

Illustration22- How correlations work

We can see how the dummy variables also present a significant correlation with the objective variable, the rental price. This analysis allows us to identify specific trends based on categorical characteristics, which we have transformed into dummy variables to include in our model.

For example, regarding location, we can observe that the location of properties has a direct and considerable impact on the rental price. Properties located in Hawaii tend to have significantly higher prices. This may be due to high tourist demand. On the other hand, New York tends to show lower prices compared to Hawaii.
This analysis confirms that location influences the rental price. Different locations present different levels of demand and perceived values, which is directly reflected in prices.

Furthermore, property type also shows a significant correlation with price. As we can see from the correlation results, an entire house tends to have higher rental prices. Something logical due to the amount of space and independence that a complete home offers. On the other hand, properties that offer only a private room tend to have lower prices for precisely the same reason, less independence, space and usually, amenities.

These observations are consistent with expectations and confirm that our model is effectively capturing the logical relationships between property characteristics and their rental prices.

7. Test & Train for Machine Learning.

To properly evaluate the performance of our machine learning model, we have to split our data into two distinct sets: the training set (train) and the test set (test).

The test set will represent the data that will be used for the model to make predictions and compare with the actual results. On the other hand, the training set (train) is made up of the data that will be used to nourish and adjust the Machine Learning model.

Both parts of the division will use the two Dataframes, the one referring to the set of explanatory variables (X) and the one referring to the objective variable (Y).

I have decided to use a split of 15% for the test set and 85% for the training set. This proportion has been selected after testing several splits and determining that this configuration offered a model with better results in terms of precision, while avoiding overfitting. This means that 85% of the data will be used to train the model, while the remaining 15% will be used to evaluate its accuracy.

To perform this split, we will use the train_test_split function from the sklearn.model_selection library, which is specifically designed to efficiently split the data between the training and test sets.

```
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)
```

Now that we have cleaned and prepared the data, and defined our target variable, we are ready to apply various machine learning models. Our goal is to identify the most effective model, capable of making accurate and reliable property price predictions.

Machine Learning establishes an algorithm that is trained with data (in this case our dataframe X and Y to learn patterns and relationships between variables. Once trained, the model uses these patterns to make predictions about unknown values of the target variable. Y This is precisely the result we want to obtain, an algorithm with the ability to predict an unknown price according to the patterns followed by the rest of the variables.

When testing different Machine Learning algorithms, we will tune their parameters and evaluate their performance using the training and test sets. This will allow us to select the model that offers the best results and is most suitable for our specific needs.

8. Machine Learning Models

8.1 Model - Linear Regression

Linear regression is an approach to take in Machine Learning, which is used to model the relationship between one or more independent variables and a dependent variable (in this case, price) in a linear way. This method assumes that the relationship between variables can be approximated by a straight line.

This model has potential to be used in the model because we have observed a linear relationship between the independent variables and the dependent variable.

To establish the model we use Linear Regression, which, as its name indicates, allows us to establish a linear regression. We create the regression model indicating the training sections of the data.

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

We apply the prediction of Y (price) according to the training and test data.

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

We import the resources to calculate the mean square error and the coefficient of determination. Hereinafter referred to as MSE and R^2 in its abbreviated form.

The squared error measures what is the average of the squared errors between the value of an estimator and that of its estimate. It is a very useful parameter to make predictions as in this case. When comparing models, the one with a lower MSE will be representative of a better model quality.

The coefficient of determination (R^2) is a statistic set between 0 and 1 depending on the prediction capacity of the model. With 0 being the total inability to predict and 1 being perfect prediction in all cases. A model with a high R^2 is generally the best model to choose (except under overfitting scenarios).

Once the MSE and R^2 values have been established, we will also carry out the graphical representation of the predictions

```
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'Model evaluation:')
print(f'MSE: {mse_test}')
print(f'R²: {r2_test}')
```

Graph with the results

```
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('Real Values')
plt.ylabel('Predicted Values')
plt.title('Regression model predictions')
plt.show()
```

Model evaluation of linear regression model:
MSE: 6718.749406826796

R^2: 0.4658384175652456

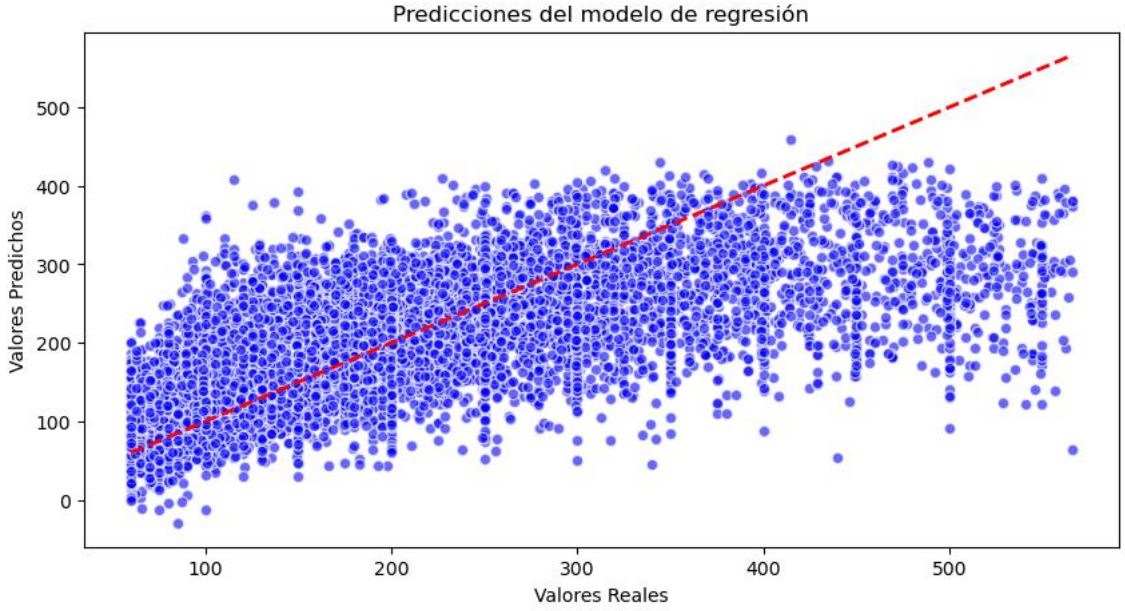


Illustration23- Linear regression

The Mean Square Error (MSE) value alone is insufficient to evaluate the quality of the linear regression model without comparing it with other models. Likewise, although the Coefficient of Determination (R^2) is a useful metric to evaluate the predictive capacity of the model, the current value of R^2 is less than 0.5. This suggests that the model explains less than 50% of the variability observed in the data, which is indicative of suboptimal predictive ability.

When analyzing the dispersion between the actual values and the predicted values, it is observed that, although there is a coherent trend between them, the dispersion is considerably high. This high dispersion suggests that the model is not adequately capturing the relationship between the independent variables and price, resulting in less accurate predictions.

Given the performance of the current model may be sufficient but could be improved, we turn to other models with the aim of improving predictive accuracy.

8.2 Models - Ridge and Lasso

In the area of Machine Learning, there are advanced techniques such as Ridge and Lasso, which are popular for improving prediction models.
These techniques, although independent, share similar characteristics and are often presented together due to their complementary properties.

Ridge and Lasso are particularly effective in avoiding the overfitting problem.
Overfitting is a phenomenon that occurs when a model fits the training data too well, with very specific data tied to its data. When new data from outside the training is introduced,

the model is unable to make accurate predictions as it lacks the flexibility to correctly interpret the new data.

Avoiding overfitting is important in our predictive model where there are very specific and highly influential categorical variables such as location that can stagnate prices at very specific figures.

Both methods introduce a penalty to the model with the objective of regularizing the regression coefficients, minimizing the impact of the most irrelevant variables and improving the generalization of the model.

These methods add a penalty to the model in order to reduce the coefficients and eliminate irrelevant predictors.

Ridge Regression is a technique that adds an “L2” type penalty. This is the sum of the squares of the coefficients to the cost function. This penalty reduces the magnitude of the coefficients, but does not reduce them to zero. It is effective in situations where all predictors are relevant and extremely large coefficients are sought to be avoided.
We import the model for your application.

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

We establish the model, the alpha value 1 indicates the penalty. The value 1 is used for a moderate penalty to prevent the impact of the coefficients from being too large.

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

We make the predictions and evaluate the model.

```
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}')
```

We look at the graphic representation

```
plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_ridge, color='violet', alpha=0.5, label='Predictions
Ridge')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red',
lw=2, label='Line of Equality')
plt.xlabel('Real Values')
plt.ylabel('Predicted Values')
plt.title('Ridge Regression')
plt.legend()
plt.show()
```

Evaluation of the Ridge Regression Model
MSE: 6718.778884659101
R^2: 0.4658360739854218

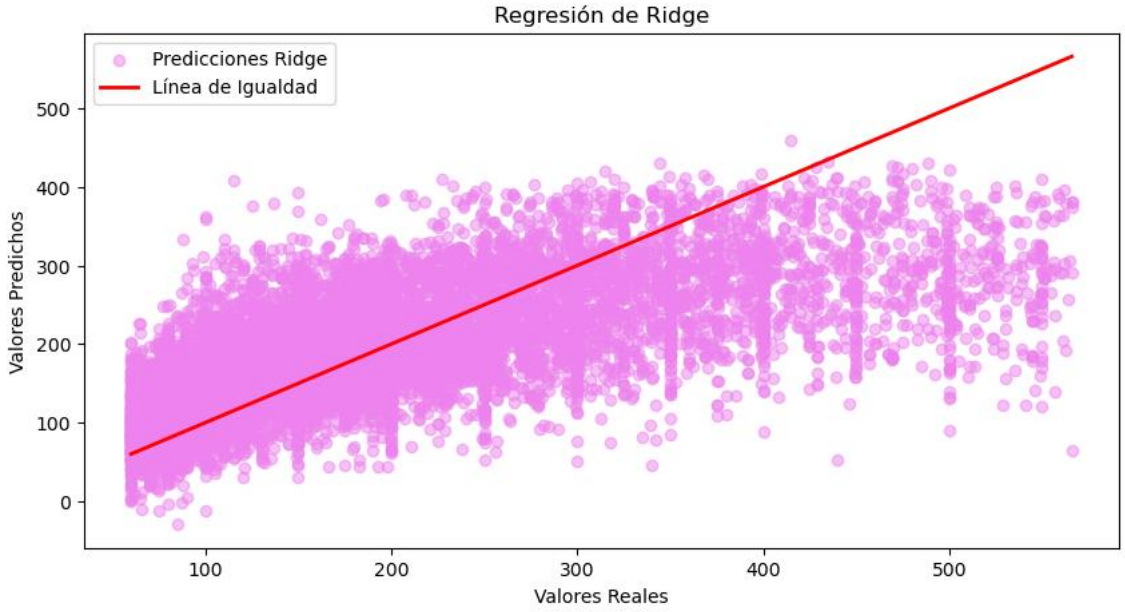


Illustration24- Ridge Regression

We can see little variation with respect to the linear regression model.
Let's apply Lasso before drawing conclusions.

Lasso uses a penalty of type “L1”, which is the sum of the absolute values of the coefficients. This penalty has the property of forcing the coefficients to be exactly zero. This implies that it performs a selection of variables and ignores those that it detects as irrelevant.

The Lasso effect is, for practical purposes, simplifying the model using only the most relevant and significant variables for it. Getting rid of variables beyond the elimination that we have done manually from the personal analysis is a strategy that can work, since the model can detect irrelevances with its algorithm that we may not have appreciated with the naked eye from the basic statistical analysis that is does about the data.
We import the model to run it with our data.

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

We establish the model and prediction.

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

We evaluate the model

```
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}')
```

We establish the graph

```
plt.figure(figsize=(10, 5))
plt.scatter(Y_test, Y_pred_lasso, color='purple', alpha=0.5, label='Predictions Lasso')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red', lw=2, label='Line of Equality')
plt.xlabel('Real Values')
plt.ylabel('Predicted Values')
plt.title('Lasso regression')
plt.legend()
plt.show()
```

Evaluation of the Ridge Regression Model

MSE: 6762.1387506757765

R²: 0.4623888290826586

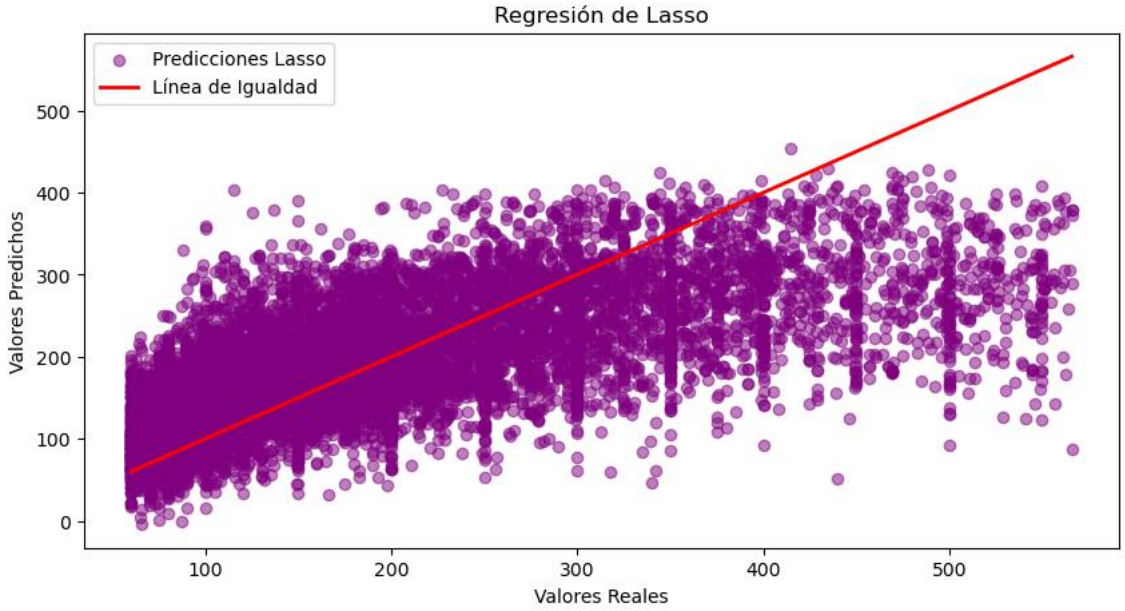


Illustration25- Lasso regression

We can see how with respect to Ridge, Lasso's MSE and R² have increased slightly, this implies that the quality of the model the more the coefficients of the variables that are considered irrelevant are reduced, the more quality it loses, so, in general, all the variables are, even to a small extent, relevant and significant.

When we compare the values obtained in the 3 models we can see that they are very similar:

Linear - MSE: 6718.715740020915 / R²: 0.4658410941815645

Ridge - MSE: 6718.778884659101 / R²: 0.465836073985421

Lasso - MSE: 6762.1387506757765 / R²: 0.4623888290826586

The best model by very little difference at this time is the linear one, since it is the one with the lowest MSE and the highest R². The little variation in the results of the model implies that it is a robust model, without overfitting and with specific variables with great relevance. Therefore, although its predictive capacity can be improved, in general terms the data seems well treated. Therefore, we will continue with the objective of finding the model with the greatest possible predictive capacity.

8.3 Models - Decision Tree and Random Forest

With the information we have about the characteristics of the model and the variables, and knowing the high impact of categorical variables, other supervised learning methods for machine learning that we can use are Decision Tree and Random Forest.

Decision Tree, or in Spanish, Decision Tree, is an algorithm that uses classification and regression to establish its algorithm. The method divides the characteristics of the data into homogeneous subsets.
The tree starts with a base composed of all the model data and selects the characteristics to iterate between the subsets, creating increasingly specific classifications and the final results of the iteration being the prediction obtained.

This is a simple model with the capacity to effectively interpret categorical variables. This is positive for our model because there are a high number of them. However, it is a model with a propensity for overfitting and shows difficulties in establishing complex relationships.

We import the regressor to carry out the Decision Tree in order to study whether, although it seems counterintuitive with the information we have analyzed so far, its simplicity lies the key to an adequate estimation. Later we will address the Random Forest model

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

We establish the model and predict.

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

We evaluate the model.

```
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}')
```

We view the results

```
plt.figure(figsize=(10, 5))
```

```
plt.scatter(Y_test, Y_pred_tree, color='limegreen', alpha=0.5,
label='Decision Tree Predictions')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red',
lw=2, label='Line of Equality')
plt.xlabel('Real Values')
plt.ylabel('Predicted Values')
plt.title('Decision Tree')
plt.legend()
plt.show()
```

Evaluation of the Decision Tree Model:

MSE: 10159.382809402248

R^2: 0.19229730572824688

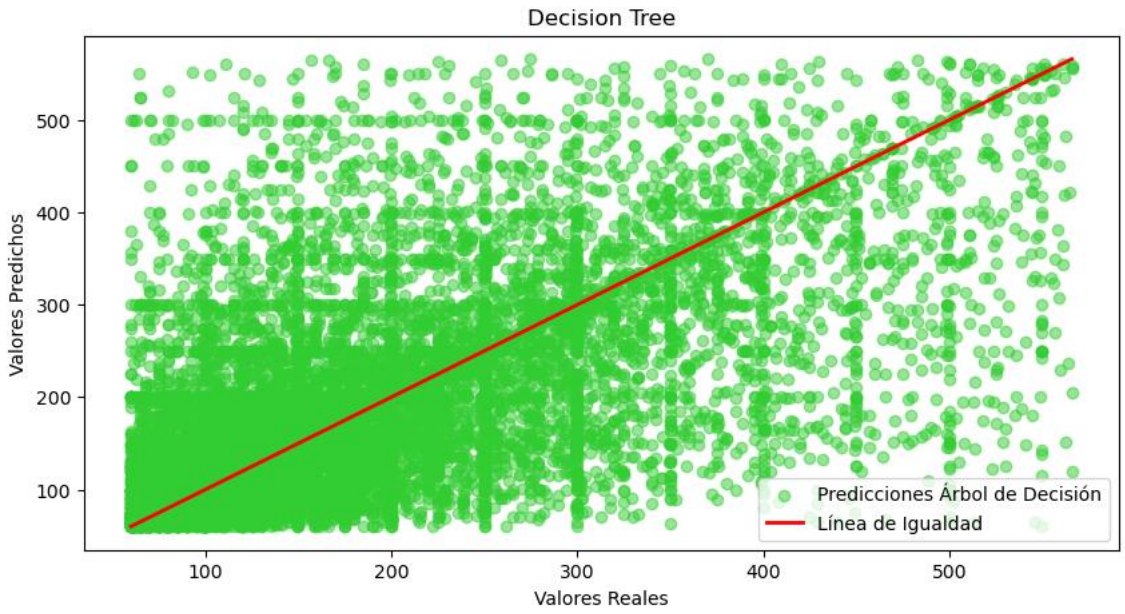


Illustration26-Decision Tree

The results of the model are much worse than previous models. This may be because we have a large number of variables with little linearity between them. As previously explained, this model is not good at working with complex relationships between many variables.

Although the first impression is that this type of model is not useful for the data used, the derived model known as Random Forest can give great results as it is able to better deal with the complex relationships of the model.

Random Forest, or Random Forest, is an extension of Decision Tree that seeks to improve the model, increasing robustness and precision. This is achieved by joining together a large number of decision trees. The method used is known as “bootstrap aggregating” and creates multiple subsets of training data using sampling with replacement. Each tree in the forest is trained on a random subset of the features, and

the final predictions are obtained by aggregating the results obtained with the averages that have shown the best results.

It is a model that requires more time and resources to process, but allows many more relationships to be established between the many variables in the model even though it has complex relationships.

We import the regressor, establish the model and make the predictions.

```
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)
```

We evaluate and visualize the model.

```
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='Predictions Random Forest')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red', lw=2, label='Line of Equality')
plt.xlabel('Real Values')
plt.ylabel('Predicted Values')
plt.title('Random Forest')
plt.legend()
plt.show()
```

Random Forest Model Evaluation

MSE: 5238.500219330838

R²: 0.5835228556226009

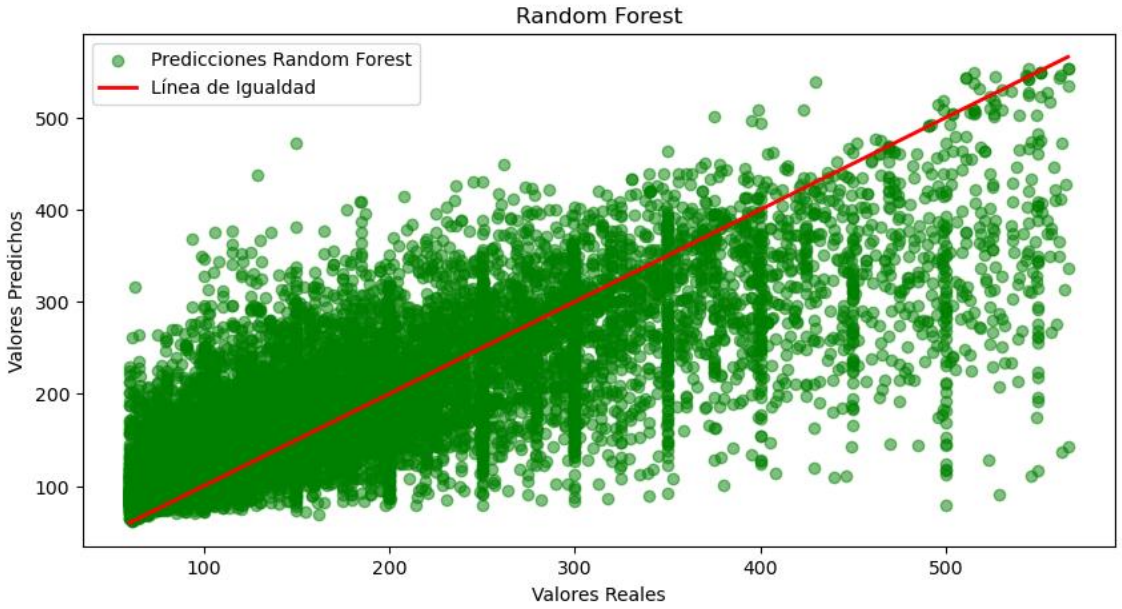


Illustration27-Random Forest

The model has performed very well, presenting the best R^2 so far and the lowest MSE. This implies that the power of the model has worked correctly by being able to relate a large number of variables with complex relationships.

Understanding the characteristics of the model data and its behavior with respect to the correlation between the independent variables and the price, we can deduce that this model, which also presents acceptable statistical evaluation data, is the best possible. However, we will finally apply a Gradient Boosting model, making sure we have tested all potential model candidates before making a final decision.

8.4 Model - Gradient Boosting

Gradient Boosting is a model with strong classification ability. It builds models sequentially, correcting errors made in each sequence, obtaining more accurate models each time. It does this through an error gradient that evolves as results are obtained.

This model is interesting to propose since it is especially good for capturing non-linear and complex relationships. Although we have detected a certain linearity in the variables, something that makes sense especially for numerical variables, such as the number of beds, it may be that the impact of the existence or not of a specific amenities has seriously conditioned the model.

We import the regressor, establish the model and perform the prediction.

```
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)
```

We evaluate the model and see its corresponding graph

```
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='Predictions Gradient Boosting')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], color='red', lw=2, label='Line of Equality')
plt.xlabel('Real Values')
plt.ylabel('Predicted Values')
plt.title('Gradient Boosting')
plt.legend()
plt.show()
```

Evaluation of the Gradient Boosting Model.
MSE: 6235.593245363433
R^2: 0.5042508428758454

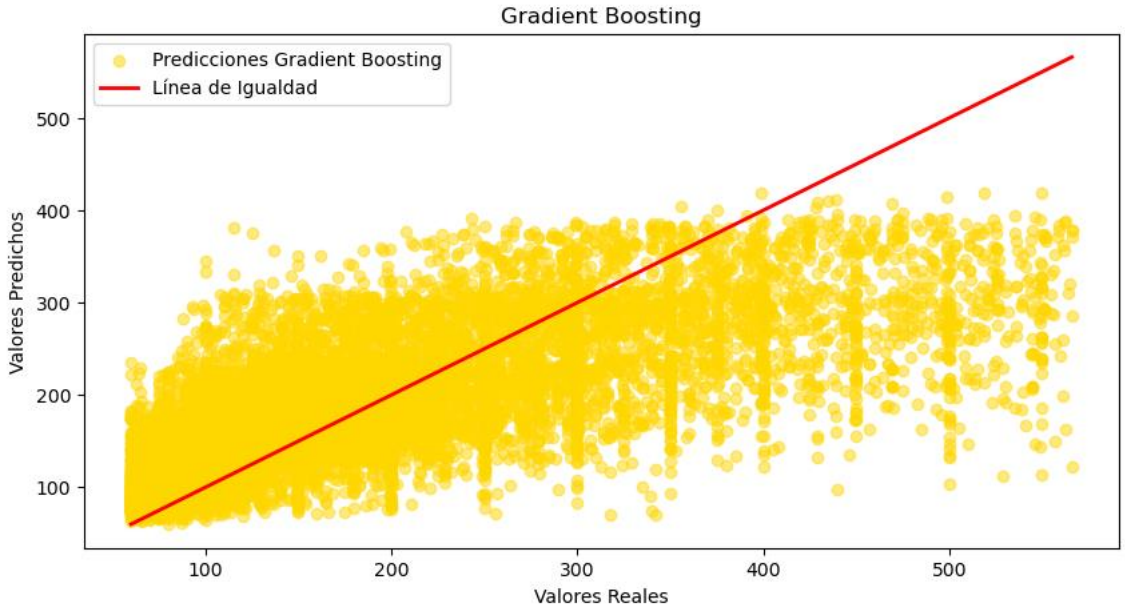


Illustration28- Gradient Boosting

The model works relatively well, in fact, it has higher R2 and lower MSE than other models. This may imply, as we have mentioned, that there are variables with a non-linear relationship that can have a high impact on the model.

8.5 Conclusions regarding the models

The evaluation of the models using Mean Square Error and Coefficient of Determination is as follows:

- 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

A lower mean square error implies that, in general, the model's predictions are closer to the real values, the model makes fewer errors in its predictions.
On the other hand, a higher coefficient of determination, which measures the variability of the price variable with respect to the rest of the variables, means that the model better captures the relationship between them.

According to these metrics, the Random Forest model is the one with the best performance, since it has the lowest MSE (5238.50) and the highest R^2 (0.58359).

Recapitulating the conclusions that we have obtained in the analysis of each model, we can determine that the model:

- It is robust.
- It does not present overfitting.
- The high number of data and variables requires a model with the capacity to process complex relationships.
- Reducing coefficients does not provide a better model, so the refined variables appear relevant.
- Its predictive capacity is good and with clear trends that are adaptable to algorithms, but a greater amount of data would allow more precise predictions.

With these conclusions, and compared the quality evaluations of the models, Random Forest achieves better precision in the predictions and better fits the observed data. Therefore, we can conclude that, based on these metrics, the Random Forest model is the best to predict given our database and it will be the one on which we apply the predictions.

Having successfully built a model via Machine Learning capable of estimating price variables is the necessary technical basis and a fundamental objective to meet the project objectives.

9. Price estimator

9.1 Manual Estimator

Lastly, we are going to establish the price estimator.
First of all, we are going to enter data manually to ensure that the model processes it correctly and is able to predict the price value correctly.

To do this, we have to create data by establishing the value of the different variables present in the model. That is, all the independent variables that make up dataframe X.

We are going to establish as data an example of a generic property with quantities appropriate to the model. A house in Brooklyn, New York. It allows accommodation for 4 people, with two rooms, three beds, a bathroom and different randomly selected amenities. Having transformed all the variables as numerical, the dummy variables will have a value of 1 or 0 depending on their existence or absence.
We establish the data that we want to predict and the value of each of its variables.

```
test_data = {
'accommodates': 4,
'bedrooms': 2,
'beds': 3,
'minimum_nights': 1,
'maximum_nights': 30,
'Smoke alarm': 1,
'Wi-Fi': 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,}
```

We convert the dictionary data to a DataFrame to ensure that the columns are in the same order as the model training DataFrame.

```
    test_dataframe = pd.DataFrame([test_data])
```

We convert the DataFrame to a numpy array, since it is the format expected by the Random Forest model.

```
    new_array = dataframe_test.to_numpy()
```

We use the pre-trained Random Forest model (rf_model) to predict the price.

```
prediction_random_forest = np.array([tree.predict(new_array) for tree in rf_model.estimators_])
```

Instead of providing an exact value, we establish an estimated range based on the 45th and 55th percentiles of the obtained predictions. This provides a more robust and manageable estimate for the client.
This not only ensures that the prediction is generally more accurate, having a greater margin to establish the price and therefore less probability of error. If not, it also gives the client the ability to finish deciding the price according to their criteria not included in the algorithm offered by the tool.

```
lower_estimate = np.percentile(random_forest_prediction, 45)
upper_estimate = np.percentile(random_forest_prediction, 55)
```

Finally, we show the estimated price range in English, since the target audience of this tool speaks this language.

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

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

The recommended price per night for this complete house for 4 people would be between $153.30 and $180

The price given is plausible and makes sense with respect to the mean of the data present in the model. In this situation, the optimal thing is to introduce more possible data to see if the model is capable of giving optimal results for homes with other characteristics. This will be done through the estimator as a final tool, which invites easy data entry.

9.2 Estimator with Widgets

To meet the last objective of the project, which is to create an estimator accessible to the average user that can be used directly in the form of an application, we have to consider the method of data entry.

Obviously, the user is not going to write code to add new data, in this case the characteristics of the house whose price he wants to estimate, so we have to create an environment with a friendly and visually understandable design so that he can enter this data with normality, the program being the one that interprets the data entries and translates it into a code format compatible with the variables of Dataframe

To create this user interface, we will use “ipywidgets”, a Python library that allows you to create interactive widgets in Jupyter Notebooks and other HTML interfaces. Additionally, we will use IPython.display to improve the visual presentation of the widgets and the final result.

Establishing an HTML format allows certain advantages, and that is that it makes it easily exportable to web format, being able to lead to the implementation and real use of the model with relative ease.

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

We created a series of widgets to capture the different characteristics of the property. The “IntSlider” widgets will be used for numerical variables and the “Checkbox” widgets will be used for dummy variables.

As explained in the device referring to outliers, the dataframe presents maximums and minimums consistent with the available characteristics of a home. That is, allowing data to be added where 1000 beds are established, on the one hand, creates a less accurate and professional interface, and on the other hand, it will give an erroneous estimate since the training data did not include anything remotely similar, having the impact of the 1000 bed coefficient an unrealistic influence on the price.

We add the possible values for each of the widgets.

Therefore, when introducing numerical variables as IntSlider, we establish a minimum and maximum based on the maximum and minimum data that we have seen in the statistical analysis of the data.

```
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')
```

For variables with different unique options we will use a dropdown.
Location also has hierarchy since zones/cities are exclusive to certain states.
For this reason, we first establish a list for the location and its possible options in it.

```
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']
        }
```

We establish the location dropdown.

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

We establish the zone (city) dropdown.

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

We define the correlation between the location and the zone, so that when you select a location, the zone options are updated automatically.

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

We establish the event between both variables.

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

We deploy the created widgets by linking the automatic update between the location and zone dropdowns.

```
display(location_dropdown, neighborhood_dropdown)
location_update()
```

We create the dropdowns of the other variables with unique values such as the property type and the bathroom type.

```
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'
)
```

Now that we have established all the widgets we establish the price definition, which depends on all the variables in the model.

```
def predictor_price(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):
```

We set the checkboxes that we have created as int (1 is present, 0 is not) since it is the dummie format that we have given it.

```
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)
```

We establish a dictionary for the new data, so that the data entered is conditioned on the variables of the model.

This step is essential since, as we have explained, the model introducing new data through code has worked correctly, but to transfer the data entry from the widgets to this format, the input data must be related to the data that the user expects to find. model for processing.

```
test_data = {
'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 ifneighborhood_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,
}
```

We add the value 0 to any missing column, in case the client leaves an option blank, for example, by not entering a zone, to ensure that no error occurs.

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

We establish the test dataframe for the prediction, which is made up of the data entered manually using the widget.

```
test_dataframe = pd.DataFrame([test_data])
test_dataframe = test_dataframe[X.columns]
new_array = dataframe_test.to_numpy()
```

We apply the prediction to the created random forest model (rf_model)
This time we set the result as int (integer) since the Airbnb application does not allow setting decimals in the price.

```
prediction_random_forest = np.array([tree.predict(new_array) for tree in
rf_model.estimators_])
lower_estimate = np.percentile(random_forest_prediction, 45)
upper_estimate = np.percentile(random_forest_prediction, 55)
lower_estimate_int = int(lower_estimate)
upper_estimate_int = int(higher_estimate)
prediction_result = f"Recommended price per night: From
\${lower_int_estimate} to ${int_upper_estimate}"
return prediction_result
```

We set a widget as a button to apply the estimate. The app would independently work without this button, it would simply constantly update the price as prediction data changes.

This implies that a new prediction is made every time the client changes a piece of information, which makes it suboptimal, both at the level of processing expense and for the client himself who would see a constantly changing output price of figures that are irrelevant to the client. the same since until you have entered all your data, the prices are useless and would simply distract the customer.

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

We define the function that will be carried out when the button is pressed, that is, display the result of the prediction.

```
def button_prediction_activate (button):
with output:
output.clear_output()
prediction_result = price_predictor(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_lines.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;">{prediction_result}</p>'))
```

We create the event and establish its visual characteristics.

```
prediction_button.on_click (prediction_button_activate)
prediction_button.style.button_color = 'lightgreen'
prediction_button.style.font_weight = 'bold'
prediction_button.layout.width = '200px'
prediction_button.layout.margin = '10px'
```

We point the correlated widgets to the “predict” button and execute the code.

```
introduce_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([enter_widgets, prediction_button])
display(ui, output)
```

The initial display that the client will see, and at the same time the definitive one that we were looking for as the definitive objective of the project, is the following:

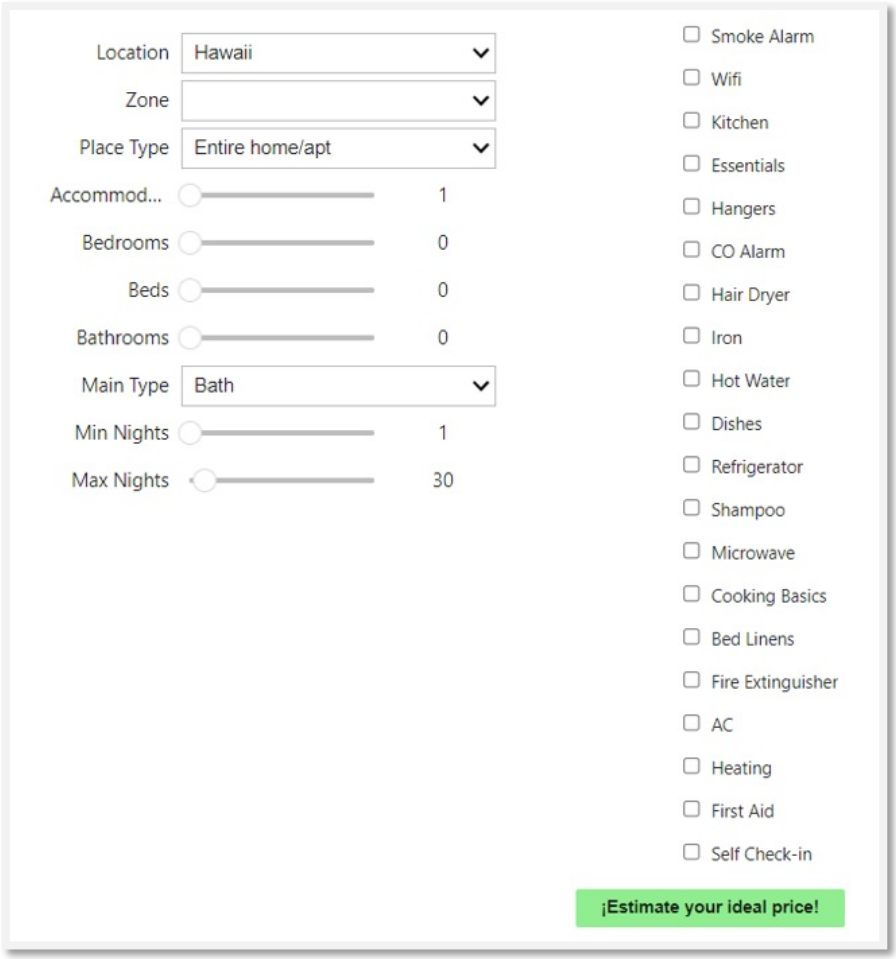


Illustration29- Estimator display

With this, the price estimator is successfully created. We are going to carry out different tests to confirm that it works correctly and that the data returned by the model make sense.

9.3 Demonstration and results.

Next, we can view the system ready for operational use.
In the previous manual data entry we have calculated the price of a complete home for four people in Brooklyn – New York. Received a price per night of $153 to $180.
In this case we are going to use other data with different characteristics. with a different type of housing such as a private room for two people in Ciudad de los Ángeles. The expectation is to receive a price lower than that obtained in the previous prediction.

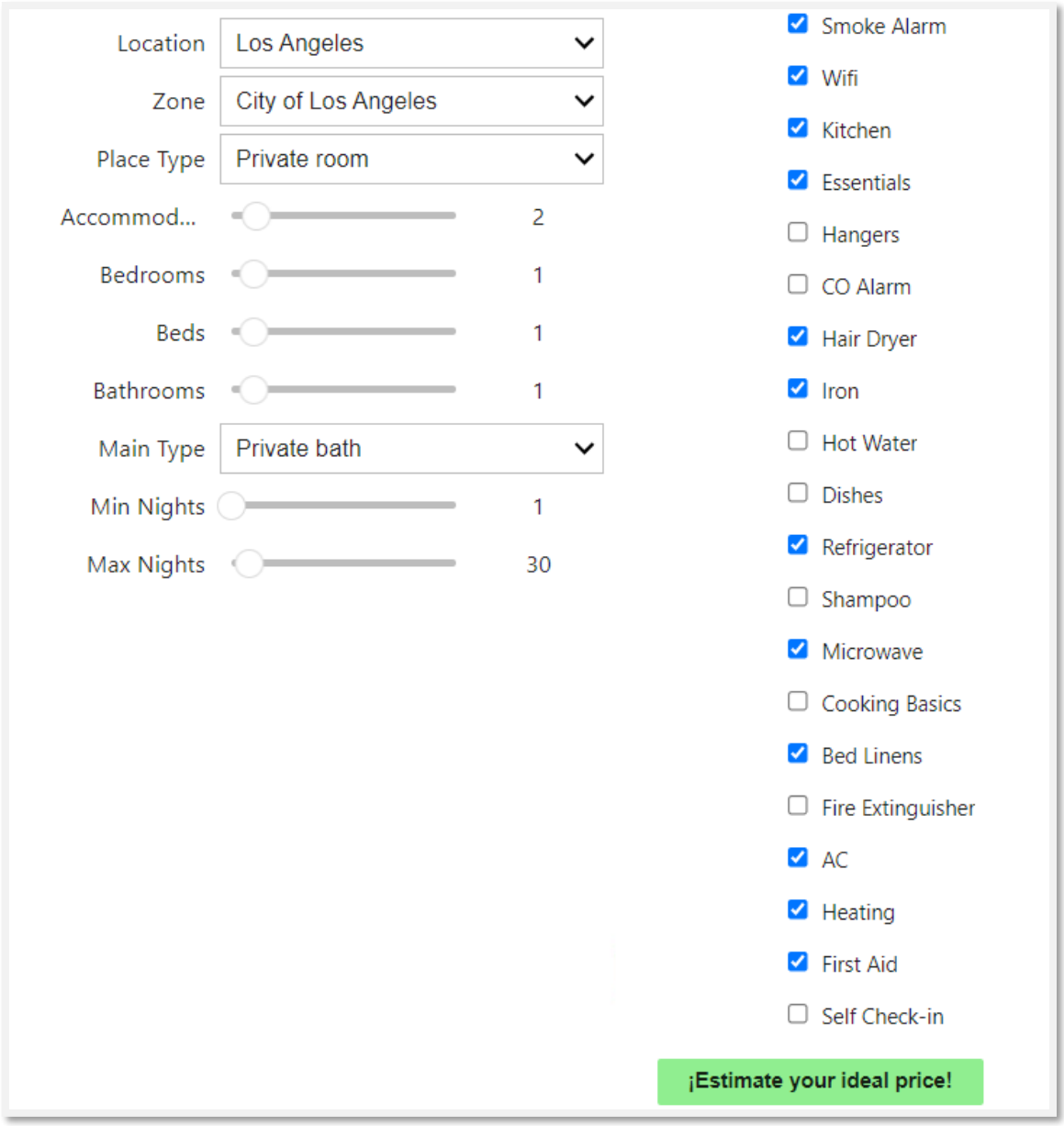


Illustration30- Estimation example – Room in Los Angeles

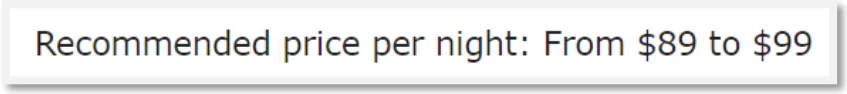


Illustration31- Display of the estimate obtained by using the button to estimate applying the data to the model.

We have obtained an estimated price per night of $89 to $99. Again a fact that makes sense, it is a good sign. We are going to change some variables to make sure that the widgets transmit information correctly to the model and the estimates make sense with respect to the applied variables.

We enter data with exactly the same characteristics, but located in Rhode Island – Kent. With this we will verify that the model correctly interprets the change in location.

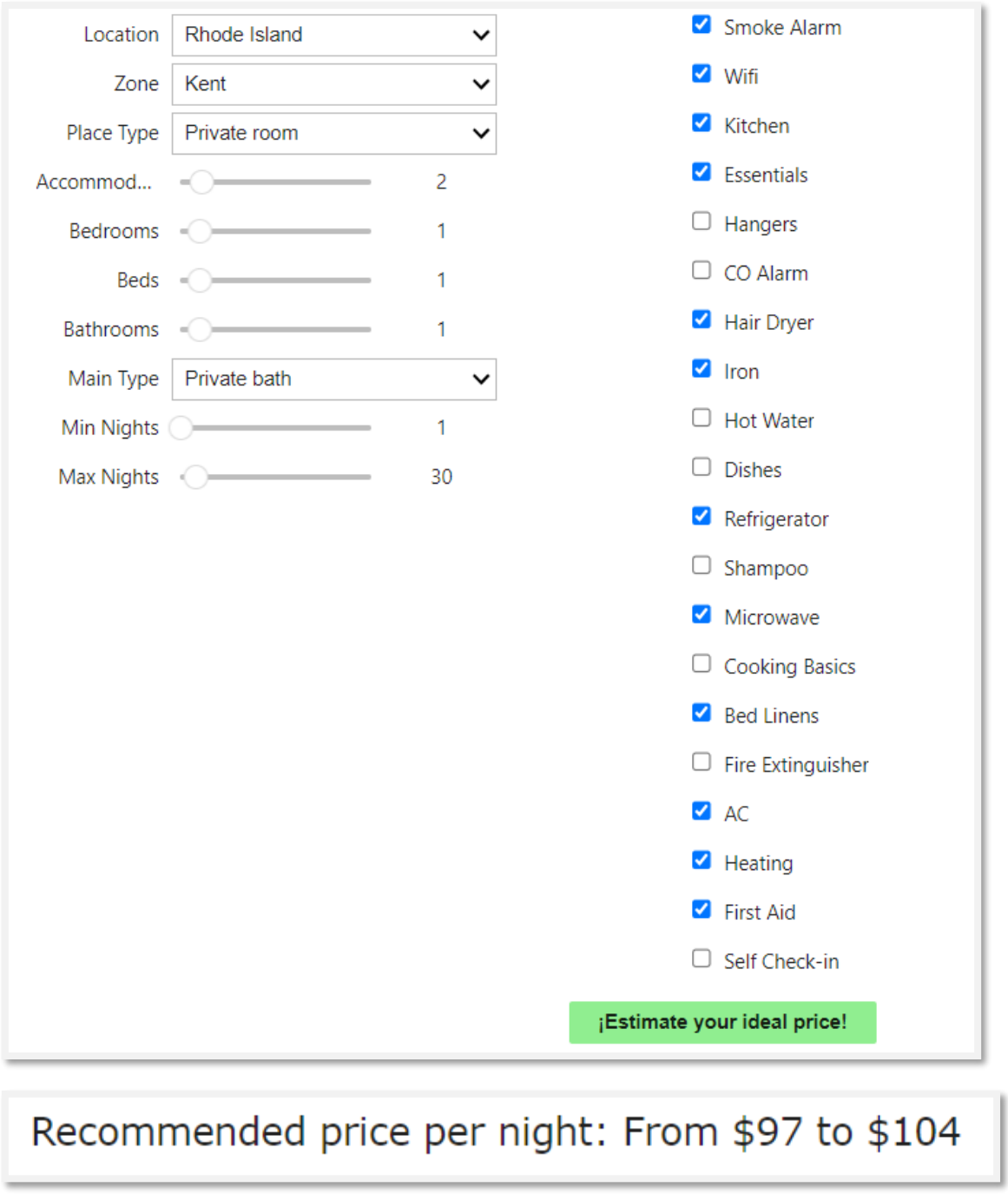


Illustration32- Estimate for room in Kent (Rhode Island)

We can see how the price has been successfully altered. For a room with these characteristics, the prices are a little more expensive for this city. This estimate is again satisfactory since the price variation exists, but it is within expected variation margins so that, despite having changed location, it continues to be a room with exactly the same characteristics for the same number of people. .

Checking the impact of numerical variables. We repeat the information from Los Angeles, but this time instead of introducing a room for two people, we will introduce a complete house for four people, two bedrooms, four beds and two bathrooms.

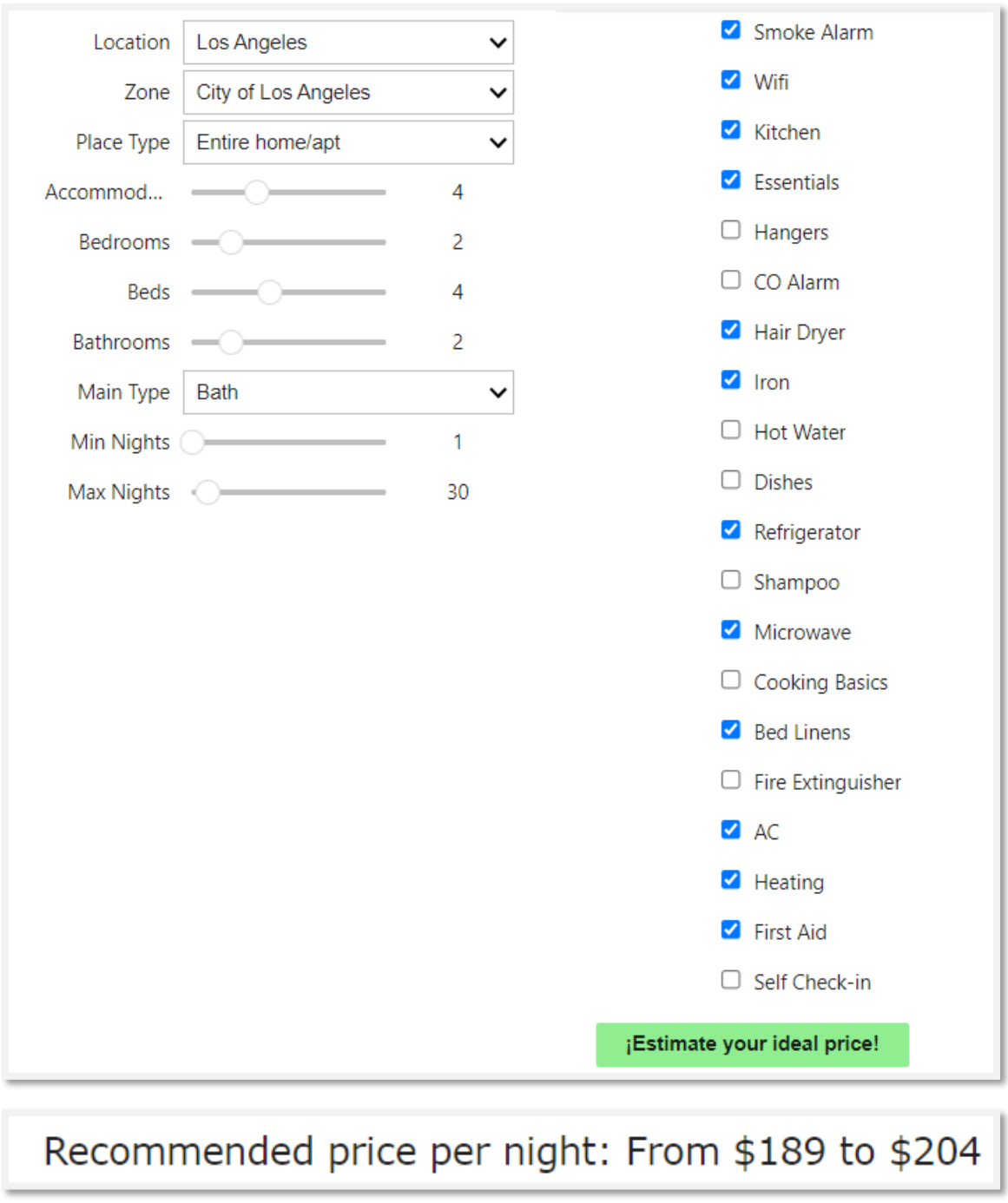


Illustration33- Estimate in Los Angeles, but with complete housing.

The price has gone from $89 to $99 per night to $189 to $204 per night which, again, is the behavior we expected given the data entered.

The estimator appears to be functional, however, until now we have kept the value of the amenities constant, establishing their existence or absence at random.

When establishing the models we have analyzed how these can have a strong impact on the predictive capacity of the model, so we introduce exactly the same data as the previous one, but changing an amenitie.

We are going to remove the refrigerator keeping the rest of the variables with the same value. This implies that for the model it is exactly a property with the same characteristics and in the same area, but now the property does not have a refrigerator for personal use.

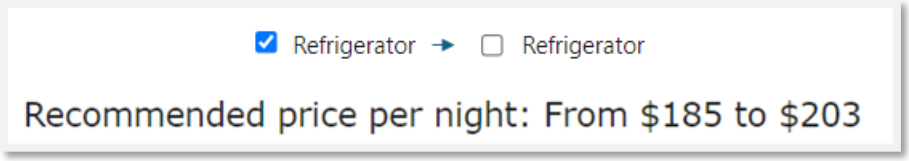


Illustration34– Repeat estimate by changing a checkbox

The minimum recommended price per night has dropped by $4 and the maximum by $1. So removing the fridge reduces the price, which makes sense.

We have therefore established estimates conditional on:

- Different locations.
- Housing of various types.
- Different number of tenants and rooms.
- Different amenities available.

The variation in the price when altering each of these data is as expected, so the model works correctly and the estimator executes it with the fidelity we were looking for.

We can confirm that the result is a success, so now that we have fulfilled the last fundamental objective of the project.
Next, we are going to compile the results obtained in each of the phases of the project to establish the definitive conclusion to it.

10. Conclusions

This project had four major objectives that we have met during its development.

- **Successful data collection and preparation of an Airbnb database.**

The dataframe to be used has been correctly imported, the data has been treated and cleaned, and irrelevant variables have been eliminated, as well as null values and outliers. New variables have also been created from variables whose data were not suitable for data processing via Machine Leanring.

- **Obtain the keys to understanding the determining factors of the rental price.**

We have carried out a statistical analysis of the variables, understanding their nature and studying their correlation. Both between the same variables and especially with respect to the dependent variable price.

- **Develop the successful Machine Learning model.**

6 possible models have been established (Linear, Ridge, Lasso, Decision Tree, Random Forest and Gradient Boosting), explaining their potentials and obtaining conclusions from their predictions. Finally taking the Random Forest model as it was the one with the best quality.

- **Implement an accessible estimator that provides the Machine Learning model with the data necessary for its prediction.**

We have developed an estimator with a friendly interface via HTML in which the average user can enter the characteristics of their home, which are transformed into data that the Random Forest model can process to obtain a price recommendation.

Of course, the model has room for improvement. As we have been able to verify when reviewing the health of the model and with respect to the determination coefficient. Because of this, there are certain combinations of amenities that have a very high impact on the price that might not be considered realistic, there are also some that for certain cities the fact of their presence slightly reduces the price instead of increasing it.

To make the model more robust and also more universal, it is recommended to incorporate more data in future iterations. Although date-specific data scraping has provided satisfactory results, they are not perfect. Obtaining additional data and from different moments in time could substantially improve the accuracy of the model.

Estimating the price of a private property can be a complex task since there are many details that are not easy to reflect in the basic data of an advertisement. That is, there are variables that are not reflected in the statistical model, either because they are not required for the creation of the advertisement or because they are abstract.

For example, there may be two apartments in the same block with four beds and while one is a renovated apartment with four beds in good condition, the other may be an unrenovated apartment and poorly treated due to the years that it indicates having four beds but two of them. Those seats are from an unfolded sofa bed.

In this example, not only would the quality and location of the beds be included, but also the condition of the property. When renting a property, the price may be conditioned by the photos shown, which may show a gray and old house or a luxurious and colorful house.

It is crucial to have the largest possible number of samples of the available variables. This helps minimize the impact of unforeseen factors. A large volume of data allows the model to better capture variations and trends, thus providing more accurate estimates.

Therefore, constantly expanding the dataset by applying transformations to the model should improve its predictive capacity in the long term.

Even so, there are variables with a very clear impact that provide affordable estimates, such as the number of tenants, and allow the creation of models like this one, with satisfactory results.

This model satisfies a real need in the market and can be really useful both for the use of individuals and for specialists who are dedicated to management or advice.

Massive data processing provides much more satisfactory, precise and faster results than manually comparing different homes and, therefore, this estimator is an asset with very valuable characteristics. Therefore, this project is indicative of the potential of data science and its real application to satisfy existing needs in our society.