Airbnb Prices in Madrid

Camila Barbagallo, Ryan Daher, Paula García Serrano, Rocío González Lantero

IE University

Professor: Benjamin Ashley

Introduction

Airbnb has revolutionized the tourism industry. It all started in 2008 when two friends, founders of this site, put an air mattress in their living room and created AirBedandBreakfast.com. In 2009 they received considerable investment, and by 2011 they were already expanding internationally (Wikipedia, 2020). As of today, they already offer "over 7 million accommodations and 50,000 handcrafted activities, all powered by local hosts" (Airbnb, About Us).

Airbnb works the following way. There are two types of users, hosts and visitors. As a guest, one has several filters to use to find the perfect room, apartment or house to stay in. As a host, one has to specify all the features of the listing, for example, the number of beds, whether it has specific amenities or not. Adding to this, one can decide whether to put any limits on the number of nights and the most critical thing, the price. Choosing the price can be one of the most challenging parts when deciding what Airbnb to stay in, because on the one hand, it needs to be worth it for the host, and on the other, appealing enough for the visitor. That is why we decided to create a model that predicts the price, so future hosts can have an idea and estimation of how much it should cost per night.

Data Understanding

The data was retrieved from Inside Airbnb. The raw dataset has 107 variables and 21845 observations. At the beginning we did a general cleaning in which we took out all spaces and replaced them with underscores, this was extremely helpful for column names. We then changed "-" to underscores and put all column names in lowercase.

Dropped Variables

Due to the high number of variables, we first did an overview analysis to reduce this number and discard those that are not useful from the beginning due to format, apparent

relation with the price (our target) or the proportion of missing values. The first eight variables: URLs (host_picture_url, host_thumbnail_url, host_url, xl_picture_url, picture_url, medium_url, thumbnail_url, listing_url), were dropped as our potential models do not support these data types, and they had a high percentage of missing values.

There were also three variables regarding the scraping of the data (scrape_id, last_scraped, calendar_last_scraped). We decided to drop them because our models do not support these variable types either, and we found no valuable transformation to make them useful to us.

We then dropped additional information about the host that had nothing to do with the listings (host_name, host_location, host_about, host_neighbourhood). Hosts also had two corresponding ids (id, host_id), which we also dropped because we were not going to merge any new dataset and we did not need them for the predictions.

We also found some inconsistent variables. Some of them (experiences_offered, country_code, country, is_business_travel_ready) had single, constant values, which would add no value to our model. Others were difficult to understand their meaning (interaction, has_availability, market, property_type, minimum_minimum_nights, maximum_minimum_nights, minimum_maximum_nights, maximum_maximum_nights, minimum_nights_avg_ntm, maximum_nights_avg_ntm, calendar_updated), we contacted Airbnb to get more information about them, but we received no response. We dropped them due to lack of understanding. There were many other variables concerning the location of the listing (neighborhood_overview, access, street, neighbourhood, neighbourhood_cleansed, neighbourhood_group_cleansed, city, state, zipcode, smart_location), that described different types of location and did not look consistent. We decided to drop them and create a variable from latitude and longitude; this topic will be addressed in the data preparation section.

Some variables (host_total_listings_count, weekly_price, monthly_price, calculated_host_listings_count, calculated_host_listings_count_entire_homes, calculated_host_listings_count_private_rooms, notes, reviews_per_month, calculated_host_listings_count_shared_rooms, availability_30, availability_60 and availability_90) included information already included within other variables or were calculated by Airbnb. As we do not know their calculation procedures, we decided to drop all of them.

First_review, last_review and number_of_reviews were dropped because each host started on a different date, making differences between older and newer hosts a bias to our model.

Transit and house_rules were deleted because they both seemed to depend on what the hosts considered to be appropriate. In other words, some hosts consider no babies an essential rule and others do not. Also, as for transit, we thought it would not have such an impact on price, and the information available could have been subjective. Review_scores_rating was also deleted for the same reasons.

Other variables (is_location_exact, requires_license) were dropped because we believed it would have no impact on price. Lastly, some variables (square_feet, license, jurisdiction_names) were deleted because they had a very high percentage of missing values (most of them over 80%).

EDA

ID

There are 21845 different ids, there is the same number of observations, meaning none of them is replicated. This also means the IDs are from the listing and not from the host.

Host_Since

This variable describes the length in which a host has been active on Airbnb. It is in date format, and there are seven missing values. To use this variable in a model, it should be transformed or recalculated into a data type supported by the respective model.

Host_Response_Time

It is a categorical variable with four different categories (within an hour, within a few hours, within a day, a few days or more) describing how long it usually takes for a host to answer. It has 4718 missing values (around 22%), and the most common category is within an hour (Annex 1), meaning hosts usually do not take long to answer.

When looking at this variable with relation to price (our target), we can see the distribution is right-skewed (Annex 2) in all cases, and it has a lot of upper-bound outliers (Annex 3). When looking at the scales of each category, one can see hosts that answer within an hour show higher prices.

Host_Response_Rate

In percentage format indicating the percentage of messages the host answers, it has 4718 missing values (around 22%). To further analyse this variable a bit of cleaning needed to be done, we eliminated the response rate and left it as numeric. After doing this, we detected this distribution has two extremes, either they are very responsive, or they do not respond at all (Annex 4,5).

Host_acceptance_rate

The value is a percentage describing the number of guests accepted by the host, with no missing values. To be able to conduct the EDA on this variable, it needed the same transformations like the ones mentioned above. The distribution also shows that hosts usually

have high acceptance rates, but others rarely accept anyone (Annex 6). The distribution also has several lower-bound outliers (Annex 7).

Host_is_superhost

Binary variable, with seven missing values. The most common category is false, meaning there are fewer superhosts than hosts (Annex 8). The values are f and t; these should be transformed to 0 and 1 for modelling purposes and multivariate EDA.

When looking at this variable with price, we can see the distribution is right-skewed in both classes (Annex 9). We can see that the range of prices is much more prominent when the host is a superhost rather than when it is not. When looking at the boxplots (Annex 10), we can see there are many upper-bound outliers, mainly in the no superhost category.

Host_listings_count

Numeric variable with seven missing values. This variable indicates how many listings each host owns. The distribution is right-skewed (Annex 11) and has a lot of upper-bound outliers (Annex 12). There are not many people that own more than 2 or 3 listings, so the distribution seems to describe reality. The upper-bound outliers also demonstrate the same thing.

Host_verifications

String variable. It has a list of strings mentioning through which platforms the host was verified. This variable needs cleaning so that we can correctly detect through which platforms the host was verified. It is recommended to create a binary variable for each platform.

Host_has_profile_pic

Binary variable indicating if the host has a profile picture should be changed to 0 and 1. The majority of hosts have one (Annex 13).

When looking at this variable in relation to price, we see the distribution is right-skewed (Annex 14) and that having a profile picture category reaches a higher price than not having one. The boxplots show a lot of upper-bound outliers (Annex 15) in both classes.

Host_Identity_Verified

Binary variable indicating if the host was verified or not, should be changed to 0 and 1. The majority of hosts are not verified (Annex 16).

Regarding this variable related to price, both distributions are right-skewed (Annex 17) and have a lot of upper-bound outliers (Annex 18).

Latitude and Longitude

Latitude and longitude are coordinates, and they have no missing values. By their own, as inputs for a model, they bring no valuable information, but it may be interesting to use them to create new variables regarding location.

Room_type

Categorical variable, with four different categories (Entire home/apt, Private room Shared room, Hotel room). This variable has no missing values, and the most common category is entire home/apt (Annex 19).

In relation with price the distributions are all right-skewed (Annex 20). In the case of the hotel room, we can see there is a significant number of cases in which the price is pretty high. The distributions also have, in both cases, many upper-bound outliers (Annex 21).

Accommodates

Describes the number of accommodates permitted in each listing. Numeric, with no missing values. Right-skewed (Annex 22) and with several upper-bound outliers (Annex 23).

The distribution seems to be describing reality as in general, the listings are for a small group of people, and few can accommodate ten or more.

Bathrooms

Numeric variable describing the number of bathrooms per listing. The distribution is right-skewed (Annex 24), and it has a lot of outliers (Annex 25), mainly in the upper-bound. This is due to the same reason as in the accommodates variable, as usually, listings have 1 to 3 bathrooms, but in some cases, in bigger listings, they have more.

Bedrooms

Discrete, numeric variable indicating the number of bedrooms of each listing. The distribution is right-skewed (Annex 26), and it has several outliers in the upper-bound (Annex 27). The distribution seems to show reality, as there are few big listings.

Beds

Numeric variable indicating the number of beds per listing. The distribution is approximately normal (Annex 28), and it has mainly upper-bound outliers (Annex 29).

Bed_type

Categorical variable, with five different categories (Real Bed, Pull-out Sofa, Futon, Couch, Airbed) describing the type of bed they have. The most common category is Real Bed (Annex 30). We know from experience many have more than one type of bed, so we believe this variable describes the best type of bed each listing has.

When looking at the difference in the distribution of price in the classes (Annex 31), we can see real bed and pull-out sofa are the only ones that are clearly right-skewed. In the other ones, we cannot identify a specific type of distribution. When looking at the boxplots (Annex 32), we can see the real bed category has the most upper-bound outliers.

Amenities

String variable, composed of a list of strings indicating the different amenities listings have. Needs cleaning to identify each amenity separately. It is recommended to use binary variables for each amenity.

Security_deposit

This variable needed a bit of cleaning to be able to analyze it. It has a dollar sign before the numbers, so we deleted the dollar sign. We also noticed that those values over 999 had a comma, so we deleted this as well and then transformed the variable into numeric. Then we transformed this into a binary variable by indicating those that had one or more as a security deposit would be 1 and those below would be 0. The distribution is right-skewed (Annex 33), and it has upper-bound outliers (Annex 34). The host decides this.

Cleaning Fee

Numeric, continuous variable. It needed a bit of cleaning to be able to perform the EDA, as it had a dollar sign and the commas just like security_deposit. Cleaning fee was also changed to binary the same way as security deposit. The distribution is right-skewed (Annex 35) and has upper-bound outliers (Annex 36). The host also decides this.

Guests_included

Discrete, numeric variable. This variable describes the number of people accepted before needing to pay an extra fee. The distribution is right-skewed (Annex 37) and has upper-bound outliers (Annex 38). The distribution shows again the size of the listings is right-skewed.

Extra_people

This is a numeric variable indicating the cost of extra people in each listing. This variable also needed a bit of cleaning to be able to perform EDA due to the dollar sign. The distribution is right-skewed (Annex 39) and has upper-bound outliers (Annex 40).

Minimum_nights

This is a numeric variable, describing the minimum nights the listing has been rented. The distribution seems right-skewed (Annex 41), but due to the many upper-bound outliers (Annex 42), we cannot identify it accurately.

Maximum_nights

This is a numeric variable, describing the maximum nights the listing has been rented. The distribution seems right-skewed (Annex 43), and it has a couple of upper-bound outliers (Annex 44).

Availability_365

This is a discrete, numeric variable, which indicates the availability of the listing in the last year, meaning how many days it was out for rent. This variable does not have a normal distribution (Annex 45), and it has no outliers (Annex 46).

Number_of_reviews_ltm

This is a numeric variable, describing the number of reviews that listing received in the last year (12 months). The distribution is right-skewed (Annex 47) and has a lot of upper-bound outliers (Annex 48).

Review_scores_accuracy

Numerical variable, ranging from 0 to 10. The distribution is left-skewed (Annex 49), and it has many lower-bound outliers (Annex 50). Meaning users usually have a high review score on accuracy, we assume by accuracy they mean how well hosts describe their listing.

Review_scores_cleanliness

Numerical variable, ranging from 0 to 10. The distribution is left-skewed (Annex 51), and it has many lower-bound outliers (Annex 52). This also means that cleanliness scores are usually high.

Review_scores_checkin

Numerical variable, ranging from 0 to 10. The distribution is left-skewed (Annex 53), and it has many lower-bound outliers (Annex 54). As in the rest of the review scores, scores for check-in are usually high.

Review_scores_communication

Numerical variable, ranging from 0 to 10. The distribution is left-skewed (Annex 55), and it has many lower-bound outliers (Annex 56). This demonstrated that communication scores are usually quite high.

Review_scores_location

Numerical variable, ranging from 0 to 10. The distribution is left-skewed (Annex 57), and it has many lower-bound outliers (Annex 58). This also shows that location review scores are usually high.

Review_scores_value

Numerical variable, ranging from 0 to 10. The distribution is left-skewed (Annex 59), and it has many lower-bound outliers (Annex 60). As in the rest of the review scores, the value tends to be high.

Instant_bookable

Binary variable indicating whether the listing can be booked instantly or not. If it is not instant bookable, it means the guests need the host to approve before booking the Airbnb.

Before doing EDA, we need to transform this variable to 0 and 1. Most listings are instantly bookable (Annex 61).

If we compare the distributions of price in both categories (Annex 62), we can see they are both right-skewed, and they only differ a little in the range of values. If it is instant bookable, the range is higher than if it is not. When looking at the boxplots (Annex 63) we can see both have many upper-bound outliers.

Require_guest_profile_picture

Binary variable indicating whether a guest is required to have a profile picture or not, again, this variable needs to be changed to 0 and 1. Most do not require a profile picture (Annex 64).

Concerning price, the distributions in both classes are right-skewed (Annex 65), and the listings which do not require a profile picture have more upper-bound outliers (Annex 66).

Require_guest_phone_verification

Binary variable indicating if guests need to verify their phone number or not needs to be changed to 0 and 1 before doing EDA. The majority do not require phone verification (Annex 67).

Concerning price, both distributions are right-skewed (Annex 68) and have upper-bound outliers (Annex 69), mainly in the class in which guests require phone verifications.

Price

This is our target variable; it is numeric and continuous. The distribution is right-skewed (Annex 70), and it has upper-bound outliers (Annex 71).

Numeric Multivariate EDA

The correlation matrix (Annex 72) shows that there is no correlation with our target variable. However, there are two different groups of variables that are correlated. All review scores have multicollinearity; if we look at it from the business side, this makes sense as usually when people strive to get good reviews, they take into account all areas and vice versa. The other group of variables with multicollinearity are guests included, bed, bedrooms, bathrooms, accommodates. This also makes sense as everything has to do with the size of the house. If the model used is sensitive to multicollinearity, this should be taken into account.

Data Preparation

Host_since

This variable indicates the starting day for which this host has begun actively posting on AirBNB. We imputed it by going to the website link for each row missing and manually inserting the missing information. If we found that some of these links did no longer exist or the information was missing or not available, we decided to drop these rows. Our following transformation was for host_since, with this variable we took April 8 as a benchmark and subtracted each host_since date from it. We ended up with an actual number indicating the length of which they have been a host, a feature which could be used for our models. The distribution is right-skewed (Annex 73) and has upper-bound outliers (Annex 74).

Host_reponse_time

We imputed host_response_time using a random choice from the data. We used this method because there is no way to replicate behaviour, so we decided to do it randomly so the distribution would not change and not include bias in the model. We lastly created dummies to be able to perform analysis on it.

Host_response_rate

We imputed host_response_rate using a random choice from the data, the same way as host_response_time. We then classified the percentages by 1,2,3,4(1:<= 25%, 2:>25% and <= 50%, 3:>50% and <=75%, 4: >75%). The most common category is 4, meaning most hosts respond more than 75% of the times (Annex 75). When looking at the price in each category, we can see all distributions are right-skewed (Annex 76) and have upper-bound outliers (Annex 77), mainly categories 1 and 4.

Host_acceptance_rate

We imputed host_response_time using a random choice from the data. The same procedure as host_reponse_rate was applied to bin the percentages in classes 1,2,3,4. When looking at the barplot (Annex 78), we can see the most common category is 4, meaning the majority of hosts accept more than 75% of their possible guests. If we look at the distribution of price in each category(Annex 79), we can see all distributions are right-skewed and quite similar, except for the one in class 4, which ranges higher than the rest. If we look at the boxplot (Annex 80), we can see all of them have upper-bound outliers, once again, class 4 has the highest range of values.

Host_is_superhost

We imputed host_is_superhost by going to the website link available and manually inserting the missing information. If we found that some of these links did no longer exist or the information was missing or not available, we decided to drop these rows.

Host_listings_count

Then we repeated the same steps as in host_is_superhost. We then compared the missing values with host_total_listings_count, and we saw that they both had the same

information and the same rows with missing values. Since the information was missing in both of them, we dropped these rows.

Host_verifications

We then transformed host_verification, which had a strange format. The value for this was a list of strings. To clean it we created a list with all possible options, and we replaced all { and “” with blank spaces and all } with commas since this meant a separation of a new row. Then we split the options available per comma, meaning every time there was a comma, there would be a split. We then looked at all the new options that were available and created groups. We were left with fewer options, and we created a list of the host_verifications that contained fewer than 10% of listings. These were then deleted because they were not as frequent in our listings.

As it can be seen in Annex 81 half of the available options for host verifications are used by the hosts for our listings. Our hosts do not as commonly use the other half. By looking at the histograms for all of these options for hosts to verify and comparing them with price, we can see that they are all right-skewed (Annex 82-91). When looking at the boxplots, we can see that all of the options have upper bound outliers (Annex 92-101).

Host_has_profile_pic

To impute the missing values of whether a host has a profile picture or not, we clicked on the links of each one and manually imputed them. If we saw that the link was no longer functioning or the information was not available, we deleted this row from the dataset.

Host_identity_verified

The same procedure as the one mentioned above was done for variable.

Latitude and Longitude

To handle the geographical information, we used a reverse geocoder. We first had to format latitude and longitude into a coordinated list. We then learned how the reverse geocoder worked and inputted our coordinated list into it. To optimize the computing time, we only imputed the unique coordinate pairs. We selected out of all the outputs that it gave the area name, and after we added it to our dataset we deleted latitude, longitude and the coordinated list. This new variable was area_name, and to be able to analyze it we created dummies. From the geographical graph(Annex 102), we can see that all areas seem to have the same average cost.

Room_type

To be able to include this variable in our model we created dummies.

Bathrooms

To impute the missing values of whether a host has a profile picture or not, we clicked on the links of each one and manually imputed them. If we saw that the link was no longer functioning or the information was not available, we deleted this row from the dataset.

Bedrooms

The same procedure as for bathrooms was used to impute the missing values.

Beds

We imputed beds with the median because they had too many missing values to do manual imputation. We chose the median because of the right-skewness observed in the EDA phase.

Bed_type

In order to be able to include it to our model we created dummies out of this variable.

Amenities

We then transformed amenities, which had the same format as host_verifications. To split the list of strings, we followed the same procedure followed in host_verifications. We then looked at all the new options that were available and created groups. An example of one of these groups is air_conditioning which includes air conditioning and central air conditioning. We were now left with fewer options, so we created a list of the amenities that contained fewer than 10% of listings. These were then deleted because they were not as frequent in our listings.

When observing Annex 103, we see that the most common amenities these Airbnb have are air conditioning, TV, white goods, elevator and internet. From the histograms, we can see all of the amenities offered have a right-skewed distribution to price (Annex 104-131). We can also see from the boxplots that all amenities have upper bound outliers (Annex 132-159).

Cancellation_policy

In order to be able to include it to our model we created dummies out of this variable.

General Cleaning

In the end, the rows deleted through the index were 489, 21090, 21264, 19588, 20732, 20737, 21208, 21537, 21590, 21638, 489, 12648, 21842. Before beginning to model we standardized and normalized all variables except for price, so that the scales were overall the same in the numeric variables and it did not bias our predictions.

Modelling

Benchmark Model

Before performing any feature selection, we decided to perform a Multiple Linear Regression, to make sure all steps we were taking that were complicating the model were also

improving our predictions. For this model, the variables were not standardized nor normalized, and all of them were included.

Feature Selection

We used different methods for feature selection, so we could identify as effectively as possible the best features we could use to predict the price.

KBest: Chi-Square

One of the techniques we used was KBest; this used the statistical test of Chi-Square. This measure tests the independence of two events, so we aimed to select those variables, which are highly dependent on our target. We performed this test to check which dummies were most significant from some of the categorical variables that had a lot of dummies (Annex 160-162) and then checked the overall ranking (Annex 163).

ExtraTreesClassifier

We also used the ExtraTreesClassifier to see which variables had more importance. This model is an ensemble technique that creates a model of uncorrelated decision trees, and then you can see the importance that each feature had (Annex 164-165).

LightGBM Recursive Feature Selection

We also performed a recursive feature selection based on a LightGBM model. From this we looked at the feature importance of each variable and compared their ranking with the methods mentioned above.

PCA and LDA

To get an idea about how many features we needed to use, we ran a PCA and an LDA. From the PCA we saw that when it increased from 24 components, the increase of variance explained between one component and the next one was not significant anymore. From LDA we saw it used ten components, so we did a little testing to see what happened if

we reduced the number of features for our model from 24. When doing this we noticed that the errors increased significantly, so we decided to use 24 features.

Final Features

After performing the analysis described above, we got our top 24 features (Annex 166) and used them for all the following models.

Different Models Tested

To make sure we were going to get the best possible model, we competitively compared several models through the MAPE; we will further explain this metric in the evaluation section. In the beginning, we ran some models without fine-tuning to see which one was performing better, and we then worked in the fine-tuning of the top-performing ones.

Regression Tree

Regression trees are like decision trees but for regression. They are constructed through binary recursive partitioning, meaning each node is split into two leaves. The split is then optimized to reduce the error as much as possible. The algorithm keeps splitting until they reach the maximum number of splits. The drawback of this model is that it usually overfits the data.

For this model, we left all parameters in default, and the model was extremely overfitted. On the train, we got a MAPE of 0.03 and on the test a MAPE of 83.23.

Multiple Regression

Multiple regression is like linear regression, but with more than one predictor variable. The algorithm tries to fit a line to the target variable created by weighting the influence of the independent variables. On the train, we got a MAPE of 211.73 and on the test 218.50.

Stochastic Gradient Descent

Gradient Descent is used to optimize the parameters of the algorithm. It works with the cost function, used to monitor the error in machine learning models. This way, it also minimizes the error.

For this model, we also left all parameters on the default, and we got lower errors than in our benchmark, but they were still very high. On the train, we got a MAPE of 200.18 and on the test 202.13.

Elastic Net

Elastic Net is a regularization algorithm for multiple regression. It is the combination of two other regularization algorithms (Lasso and Ridge regression). What it essentially does is it generates a sparse model, by zeroing out the least important features, and removes the limitations on the selected features.

This was another quick tryout, which gave similar results from the rest. On the train, we got a MAPE of 204.21 and on the test 199.59.

Light GBM

LightGBM is a Gradient Boosting Machine, which uses tree-based learning algorithms, such as gradient boosting. The main difference to other tree-based learning algorithms is that it grows leaf-wise instead of level-wise.

After performing the quick tryouts on this model, we saw it was performing better than the ones mentioned above, so we proceeded to fine-tune the parameters. To get an idea of the parameters we should use to optimize the MAPE, we used the optuna.integration.lightgbm library. This algorithm tries out different values, and you only need to specify the objective, which was a regression, the metric you are optimizing for, in this case, MAPE and the boosting_type, for which we chose gradient boosting. After this, we

got preliminary parameters (Annex 167), and we trained and validated the model again and iterated to fine-tune our model further. We then selected our best model iteration and got a MAPE of 138.11 on the train and 138.72 on the test, which is better than all of the ones mentioned above.

MLP

Another of our top models in the quick tryouts was MLPRegressor, a neural network. This model works by the activation function that each node in each layer has. The nodes in each layer combine the input from the data with a set of coefficients that either amplify or dampen that input. These input-weight products are summed, and then the sum is passed through a node's so-called activation function, to determine whether and to what extent that signal should progress further through the network to affect the ultimate outcome, in this case, predict the price for each Airbnb house. Each layer's output is simultaneously the subsequent layer's input, starting from an initial input layer receiving your data.

With this understanding, we found out that the fine-tuning that was helping us was using 'adam' as our solver, 'tanh' as our activation function and establishing two hidden layers with 16 each. The results turned out much better than the models mentioned above, on the train we got a MAPE of 80.38 and on the test 77.65. The model is a bit overfitted but is still better than the models mentioned above.

Random Forest

Random Forest is an ensemble method, which combines different decision trees and uses the mean of all predictions to get the final prediction. To create the different decision trees, it uses bootstrapping; this means it takes different samples from the training set and creates a decision tree for each subsample.

Random Forest was also one of our top models. We used RandomizedSearchCV to fine-tune this model. We optimized the parameters: max_depth, n_estimators, min_samples_leaf, min_samples_split and we got the optimal estimator (Annex 168). With these parameters, we got a MAPE of 70.48 on the train and 76.71 on the test. Although the MAPE for the test was smaller than in MLP, the model was more overfitted.

XGBoost

XGBoost is also an ensemble decision-tree based modelling algorithm. One of its key characteristics is that it works very well with unstructured data. What it does is that it fits one decision tree and the next one is fitted to correct the errors of the previous one, this way it minimizes errors.

XGBoost was also one of our top-performing models. We also used RandomizedSearchCV to tune the parameters and optimize model performance. We tuned the following parameters: max_depth, min_child_weight, n_estimators. After iterating, we reached the best estimator parameters (Annex 169). After this, we also ran a GridSearchCV, and we got another best estimator parameters (Annex 170). After comparing both of these models, we compared their MAPEs, and the parameters we got from the GridSearchCV turned out to be better than those from the RandomizedSearchCV. We got a MAPE of 75.18 on the train and 89.67 on the test.

SVR

SVR algorithm is similar to linear regression, but it also works with margins. The main advantage of SVR is that it lets us control how much error we want to accept. Therefore, we need to get to the perfect balance between being flexible so that it does not overfit the training data but being strict so it does not introduce bias to the training data, and it still can be generalizable. We can control the error with the epsilon parameter, which

defines the width of the margins we are creating. To control the points that are outside of our prediction, we have the parameter C, as this increases the tolerance for points outside of the epsilon (the margin) also increases.

We also ran an SVR, which performed the best out of the ones mentioned above. We fine-tuned C and chose the kernel type with GridSearchCV, we then compared the results from the GridSearchCV with the results of the default parameters, and we saw the default model was performing better (Annex 171). With these parameters, we got a MAPE of 42.67 in the train and 42.50 in the test.

Average Ensemble

Average Ensemble is a technique to average the predictions done by different models. Each model makes its prediction, and then all of these results are averaged, this averaging can also be weighted.

We also tried combining our top performers to see if we could improve the model further. We did an average ensemble of SVR, XGBoost and MLP. We then tested different weights for each model when averaging, and the best model we got was with the following weights: 0.9 for SVR and 0.05 for XGBoost and MLP, with this we got a MAPE of 42.87 on the train and 44.30 on the test. This model was not performing better than SVR alone, and the weights indicated that SVR had the most power in the prediction. We concluded we were overcomplicating the model, and the results were not getting better, so it was discarded.

Voting Ensemble

Voting Ensemble is another way of combining different models. This one also averages the predictions of two models, which are trained with the same training data.

We also tried two different Voting Ensembles. The first one was of SVR and XGBoost (Annex 172). With this one, we got a MAPE of 51.79 on the train and 57.22 on the

test. The second one was with SVR and MLP (Annex 173), with a MAPE of 67.62 on the train and 64.31 on the test. Once again, none of these performed better than SVR, so they were also discarded.

Bagging Regressor

The Bagging Regressor works similarly to random forest, but instead of doing it with decision trees, you can choose the algorithm you want. It takes random subsamples from the training dataset and fits the model you choose to each of these. After this, it predicts with each model and takes the average of the prediction.

First, we ran the Bagging Regressor with its default model, decision trees. This model was very overfitted; we got a MAPE of 30.72 on the train and 79.50 on the test.

As SVR was our top-performing model, we thought trying it with bagging could improve it. We then optimized it with GridSearchCV and got the best performing estimator (Annex 174). The results we got were a MAPE of 42.54 for the train and 42.40. Although these results are slightly better than SVR without bagging, they are not significantly better, and the model complexity is higher, so we also decided to discard this one.

Stacking Regressor

Stacking Regressors first predict with the variables you input; after this, predictions are made by them and are used as input for the meta regressor, which predicts based on the predictions from the other models.

For the stacking regressor, we used XGBoost and MLP as models and SVR as a meta regressor (Annex 175). The results we got were again worse than SVR alone, on the train we got a MAPE of 50.69 and 61.43 on the test.

Final Model

After competitively comparing all models through MAPE and also considering model complexity, we decided SVR was our best model.

Evaluation

Validation Method

To validate our models and make sure they generalized correctly, we used train and test split. We trained all our models with a random 70% of our dataset and tested with the remaining 30%. We also tried out nested validation, which consists of dividing the dataset in train and validation, so we can then divide the train set in test and train. This way, we would train on the train, fine-tune hyperparameters on the test and validate on the validation. However, as our data was small, we needed as much data as possible to train, so we stuck to basic training and testing.

Evaluation Metric

Before starting modelling, we agreed to use MAPE as a measure for evaluation. This is the average absolute percentage error. We used this because our data has many outliers, and we thought it would be an adequate metric to use to evaluate. This metric was also used to compare model performance and select the best model, based on the predictive power.

Implications and Deployment

Although our model seems to be performing quite well, we believe several things could be done to improve the results, not only statistical wise but also from the business standpoint.

First of all, we think adding new, more predictive, features to the dataset could improve the model predictions. Maybe introducing the average prices of hotels in the area or the income per capita per area could be strong predictors for price. Other options could also

be the distance to the city centre and airports or train stations and whether there are conference buildings nearby for business purposes.

Another possible way of improving our model could be predicting the logarithm value of the price. This way, the price would be a bit more standardized, and it is also easy to back transform and explain in the business context.

Related to the industry, we think predicting the rank of the price instead of the price itself could also be useful. Prices change seasonally and also throughout the years, depending on the general economic situation at that point. Therefore, if we predict the rank of price and then check the prices at that time, our model will be able to last throughout the years, and it would not need to be retrained as much.

Conclusion

In conclusion, predicting the price of an AirBNB listing is influenced heavily by external factors, most of which are not present in this dataset. Although we cleaned and selected our data, tested out a variety of models, and optimized their parameters to the best we could, our model was not able to predict prices too accurately due to several reasons. First of all, none of the independent variables are strongly correlated with the price, and they do not greatly influence housing prices. In addition, there are many more factors that can control price that is outside of the scope of this project. In the end, the price of an AirBNB listing comes down to a subjective decision from the host. Although they may be impacted or driven based on several factors such as the location of the listing, the ratings, or the size, prices fluctuate greatly, and outside factors relating to the host's personal life, as well as general regional economic status, among other influences, will have a greater influence on the final price.

Resources

3.2. Tuning the hyper-parameters of an estimator. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/grid_search.html#multimetric-grid-search

3.3. Metrics and scoring: quantifying the quality of predictions¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

6.3. Preprocessing data. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/preprocessing.html

About Us. (n.d.). Retrieved May 7, 2020, from https://news.airbnb.com/about-us/

Airbnb. (2020, May 6). Retrieved May 7, 2020, from https://en.wikipedia.org/wiki/Airbnb#Product_overview

Allibhai, E. (2018, November 19). Building an Ensemble Learning Model Using Scikit-learn. Retrieved May 10, 2020, from https://towardsdatascience.com/ensemble-learning-using-scikit-learn-85c4531ff86a

Alto, V. (2019, December 31). Ensemble Methods for Decision Trees. Retrieved May 10, 2020, from

https://medium.com/analytics-vidhya/ensemble-methods-for-decision-trees-f4a658af754d

Amrmahmoud123. (2019, April 25). Guide to Ensembling methods. Retrieved May 10, 2020,from https://www.kaggle.com/amrmahmoud123/1-guide-to-ensembling-methods

Andrew. (2015, August 11). What does the "verbosity" parameter of a random forest mean? (sklearn). Retrieved May 10, 2020, from https://stackoverflow.com/questions/31952991/what-does-the-verbosity-parameter-of-a-random-forest-mean-sklearn/51876268

Bhattacharyya, I. (2018, July 12). Support Vector Regression Or SVR. Retrieved from https://medium.com/coinmonks/support-vector-regression-or-svr-8eb3acf6d0ff

Bronshtein, A. (2017, May 09). Simple and Multiple Linear Regression in Python. Retrieved May 10, 2020, from https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9

Bergman, A. (2019, September 02). Predicting Left Ventricular End Diastolic Volume. Retrieved May 10, 2020, from https://medium.com/analytics-vidhya/predicting-left-ventricular-end-diastolic-volume-d7d3059d6be0

Brownlee, J. (2019, December 13). Ensemble Machine Learning Algorithms in Python

with scikit-learn. Retrieved May 10, 2020, from https://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/

Brownlee, J. (2020, April 8). Feature Importance and Feature Selection With XGBoost in Python. Retrieved May 10, 2020, from https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/

Brownlee, J. (2020, January 10). How to Develop a Weighted Average Ensemble for Deep Learning Neural Networks. Retrieved May 10, 2020, from https://machinelearningmastery.com/weighted-average-ensemble-for-deep-learning-neural-networks/

Brownlee, J. (2019, May 14). How to Improve Machine Learning Results. Retrieved May 10, 2020, from https://machinelearningmastery.com/how-to-improve-machine-learning-results/

Calderini, M. (2019, October 27). Plot Organization in matplotlib - Your One-stop Guide. Retrieved May 10, 2020, from https://towardsdatascience.com/plot-organization-in-matplotlib-your-one-stop-guide-if-you-are-reading-this-it-is-probably-f79c2dcbc801

Chutani, S. (2015, September 16). How can I assign weights to regressor models in ensemble framework?. Retrieved May 10, 2020, from https://www.researchgate.net/post/How_can_I_assign_weights_to_regressor_models_in_ensemble_framework

Development Guide. (n.d.). Retrieved from https://lightgbm.readthedocs.io/en/latest/Development-Guide.html

Dubey, A. (2018, December 15). Feature Selection Using Random forest. Retrieved from https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f

Ensemble methods. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/ensemble.html

Galarnyk, M. (2020, May 1). PCA using Python (scikit-learn). Retrieved May 10, 2020, from https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

Girgin, S. (2019, May 22). Support Vector Regression in 6 Steps with Python. Retrieved May 10, 2020, from https://medium.com/pursuitnotes/support-vector-regression-in-6-steps-with-python-c4569acd062d

Ghouzam, Y. (2017, August 19). EDA, Introduction to Ensemble Regression. Retrieved May 10, 2020, from https://www.kaggle.com/yassineghouzam/eda-introduction-to-ensemble-regression

Hale, J. (2020, February 21). Scale, Standardize, or Normalize with Scikit-Learn. Retrieved May 10, 2020, from https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02

Howal, S. (2017, December 06). Ensemble Learning in Machine Learning: Getting Started. Retrieved May 10, 2020, from

https://towardsdatascience.com/ensemble-learning-in-machine-learning-getting-started-4ed85eb38e00

How To Make Grouped Boxplots in Python with Seaborn? (2019, March 1). Retrieved May 10, 2020, from https://cmdlinetips.com/2019/03/how-to-make-grouped-boxplots-in-python-with-seaborn/

Huneycutt, J. (2018, May 29). An Introduction to Clustering Algorithms in Python. Retrieved May 10, 2020, from https://towardsdatascience.com/an-introduction-to-clustering-algorithms-in-python-123438574097

Implementing a Principal Component Analysis (PCA). (2014, April 13). Retrieved May 10, 2020, from https://sebastianraschka.com/Articles/2014_pca_step_by_step.html

Juhi. (2019, February 26). Simple guide for ensemble learning methods. Retrieved May 10, 2020, from https://towardsdatascience.com/simple-guide-for-ensemble-learning-methods-d87cc68705a2

Kapadia, S. (2019, September 5). Topic Modeling in Python: Latent Dirichlet Allocation (LDA). Retrieved May 10, 2020, from https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0

Kaushik, S., & Saurav. (2017, February 15). Ensemble Models in machine learning? (with code in R). Retrieved May 10, 2020, from

https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/

Limam, M. (2010, February). SVR with different kernels. Retrieved May 10, 2020, from https://www.researchgate.net/figure/SVR-with-different-kernels-a-Linear-b-Polynomial-c-Gaussian-RBF-and-d_fig1_46526966

Liu, C. (2020, February 13). SVM Hyperparameter Tuning using GridSearchCV. Retrieved May 10, 2020, from https://towardsdatascience.com/svm-hyper-parameter-tuning-using-gridsearchcv-49c0bc55ce29

Madrid. Adding data to the debate. (n.d.). Retrieved March 25, 2020, from http://insideairbnb.com/madrid/?neighbourhood=&filterEntireHomes=false&filterHighlyAvailable=false&filterRecentReviews=false&filterMultiListings=false

Mandot, P. (2017, August 17). What is LightGBM, How to implement it? How to fine tune the parameters? Retrieved May 10, 2020, from https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

Matplotlib Bar Chart: Create bar plot from a DataFrame. (n.d.). Retrieved from https://www.w3resource.com/graphics/matplotlib/barchart/matplotlib-barchart-exercise-11.php

Matplotlib.pyplot.grid. (n.d.). Retrieved May 10, 2020, from https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.grid.html

Matplotlib.pyplot.hist. (n.d.). Retrieved May 10, 2020, from https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.hist.html

Matplotlib.pyplot.subplots. (n.d.). Retrieved May 10, 2020, from https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplots.html

Mikulskibartosz. (2019, June 3). PCA - how to choose the number of components? Retrieved May 10, 2020, from https://www.mikulskibartosz.name/pca-how-to-choose-the-number-of-components/

Morde, V. (2019, April 08). XGBoost Algorithm: Long May She Reign! Retrieved May 10, 2020, from https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Neelindresh. (n.d.). neelindresh/NeelBlog. Retrieved from https://github.com/neelindresh/NeelBlog

Nicholson, C. (n.d.). A Beginner's Guide to Neural Networks and Deep Learning. Retrieved May 10, 2020, from https://pathmind.com/wiki/neural-network

Optuna. (n.d.). optuna/optuna. Retrieved from https://github.com/optuna/optuna/blob/master/examples/lightgbm_tuner_simple.py

Ozaki, K. (2020, March 3). LightGBM Tuner: New Optuna Integration for Hyperparameter Optimization. Retrieved from

https://medium.com/optuna/lightgbm-tuner-new-optuna-integration-for-hyperparameter-optimization-8b7095e99258

Pandas.DataFrame.boxplot. (n.d.). Retrieved May 10, 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.boxplot.html

Pandas.to_numeric. (n.d.). Retrieved May 10, 2020, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_numeric.html

Partial Dependence Plots. (n.d.). Retrieved from https://scikit-learn.org/stable/auto_examples/inspection/plot_partial_dependence.html#sphx-glr-auto-examples-inspection-plot-partial-dependence-py

Poussel, G. P. G. (1964, April 1). Clustering of items based on their category belonging. Retrieved May 10, 2020, from https://stats.stackexchange.com/questions/108586/clustering-of-items-based-on-their-category-belonging

PowerBi. (n.d.). Retrieved from https://powerbi.microsoft.com/es-es/

Puxama. (2018, July 09). Support Vector Regression Boston Data. Retrieved May 10, 2020, from https://www.kaggle.com/puxama/support-vector-regression-boston-data

Random forest. (2020, May 02). Retrieved May 10, 2020, from https://en.wikipedia.org/wiki/Random_forest

Raschka, S. (n.d.). Mlxtend. Retrieved May 10, 2020, from http://rasbt.github.io/mlxtend/

Raschka, S. (n.d.). StackingRegressor. Retrieved May 10, 2020, from http://rasbt.github.io/mlxtend/user_guide/regressor/StackingRegressor/

Regression Trees. (2015, December 30). Retrieved May 10, 2020, from https://www.solver.com/regression-trees

Sethi, A. (2020, March 27). Support Vector Regression In Machine Learning. Retrieved May 10, 2020, from https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/

Showmik, D. (2020, January 19). Unorthodox but efficient way of selecting features in solving machine learning problems in Python: Retrieved from https://medium.com/@das_showmik/unorthodox-but-efficient-way-of-selecting-features-in-solving-machine-learning-problems-in-python-ae44d6c1e291

Singh, A. (2018, June 18). A Comprehensive Guide to Ensemble Learning (with Python codes). Retrieved May 10, 2020, from https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

Sklearn.cluster.MiniBatchKMeans. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html#sklearn.cluster.MiniBatchKMeans

Sklearn.decomposition.PCA. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Sklearn.ensemble.BaggingRegressor. (n.d.). Retrieved May 10, 2020, from

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html

Sklearn.ensemble.VotingRegressor. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingRegressor.html

Sklearn.model_selection.GridSearchCV. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Sklearn.lda.LDA. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/0.16/modules/generated/sklearn.lda.LDA.html

Sklearn.linear_model.LinearRegression. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Sklearn.neural_network.MLPRegressor. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

Sklearn.preprocessing.MinMaxScaler. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Sklearn.svm.SVR. (n.d.). Retrieved May 10, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

Support Vector Regression (SVR) using linear and non-linear kernels. (n.d.). Retrieved

May 10, 2020, from https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html

Teknomo, K. (n.d.). NonLinear Transformation. Retrieved May 10, 2020, from https://people.revoledu.com/kardi/tutorial/Regression/nonlinear/NonLinearTransformation.htm

Tripathi, S. (2015, May 14). How do I select hyper parameters in support vector regression? Retrieved May 10, 2020, from https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/

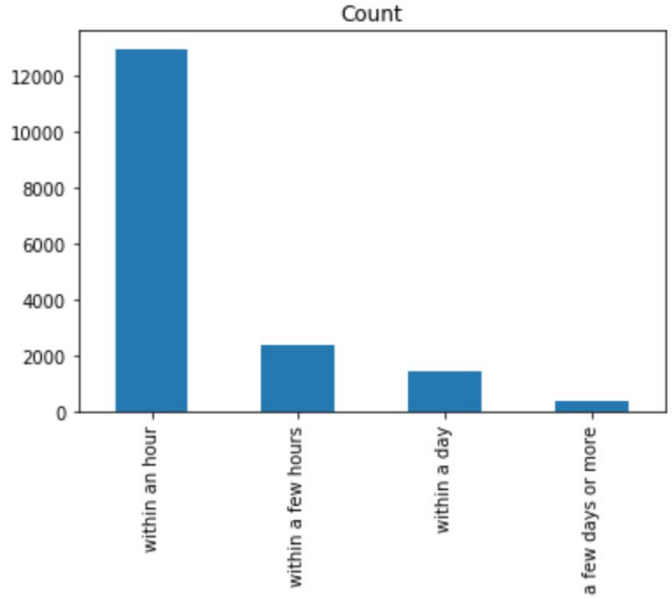
VanderPlas, J. (n.d.). In Depth: Principal Component Analysis. Retrieved May 10, 2020, from https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

VanderPlas, J. (n.d.). Multiple Subplots. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/04.08-multiple-subplots.html

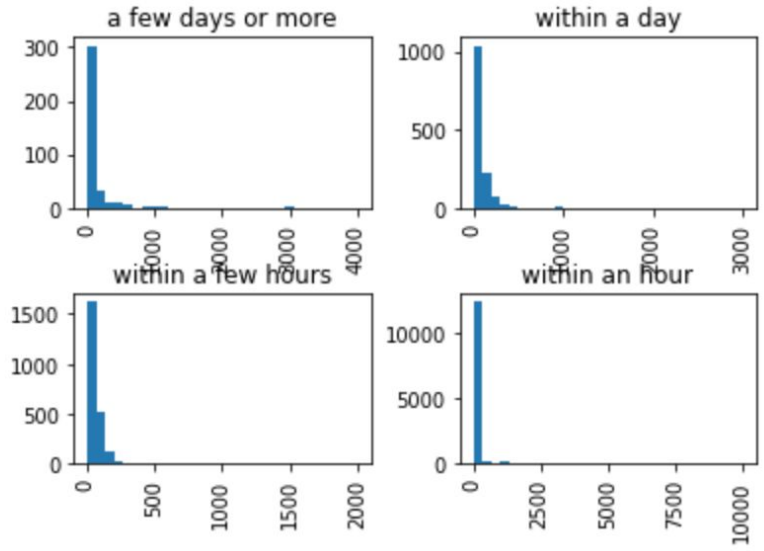
Welcome to LightGBM's documentation! (n.d.). Retrieved May 10, 2020, from https://lightgbm.readthedocs.io/en/latest/

Annex

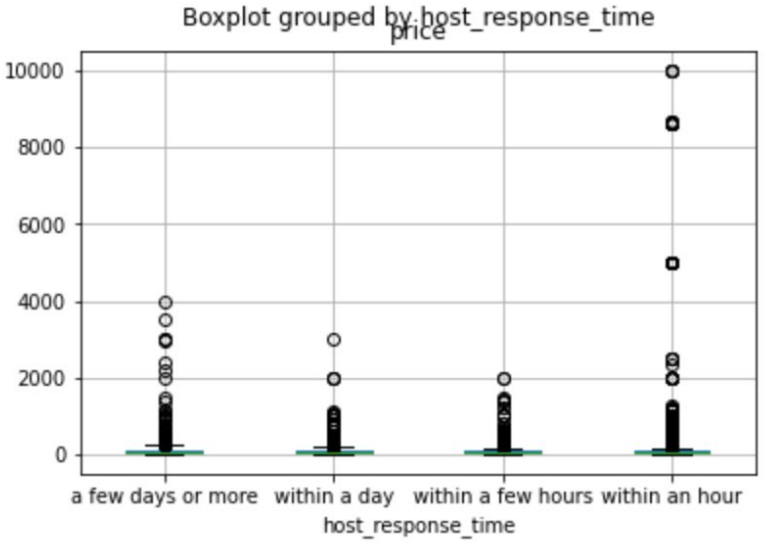
Annex 1: Barplot of Host_Response_Time



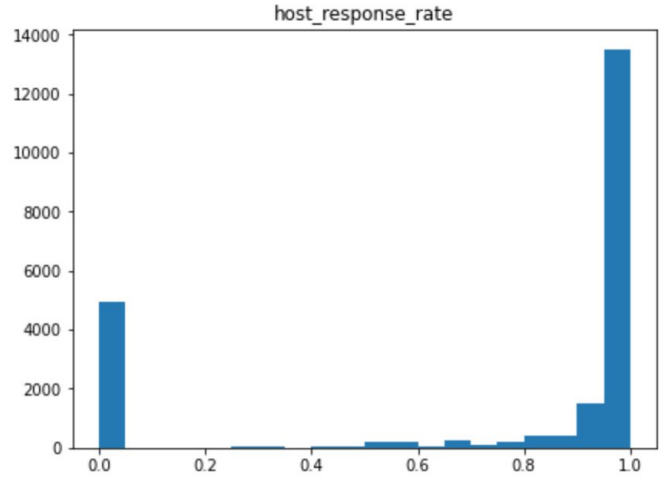
Annex 2: Histogram of Price and Host_Response_Time



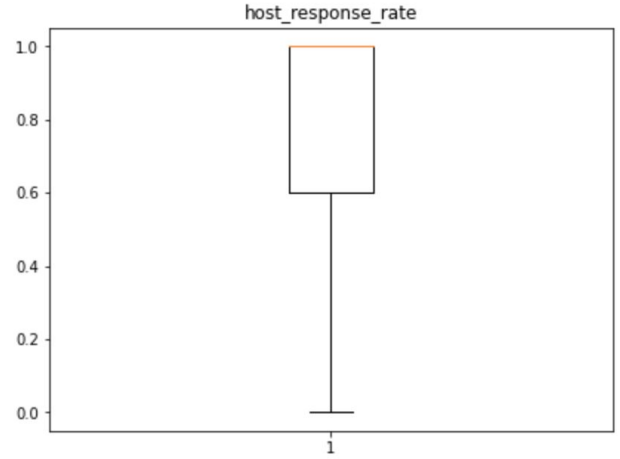
Annex 3: Boxplot of Price and Host_Response_Time



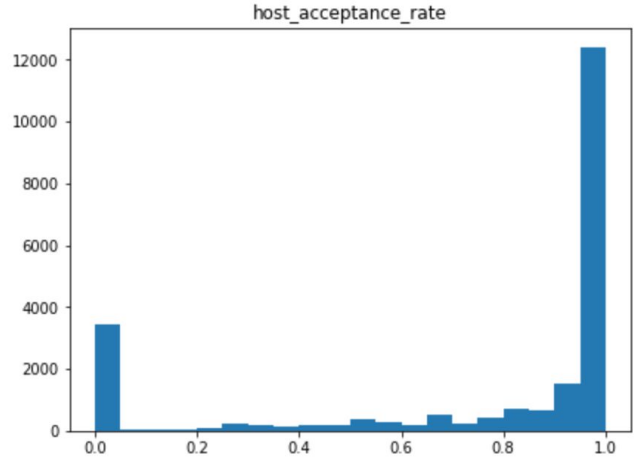
Annex 4: Histogram of Host_response_rate



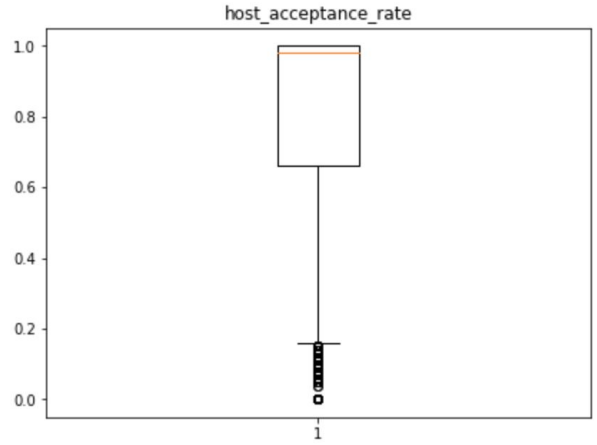
Annex 5: Boxplot of Host_response_rate



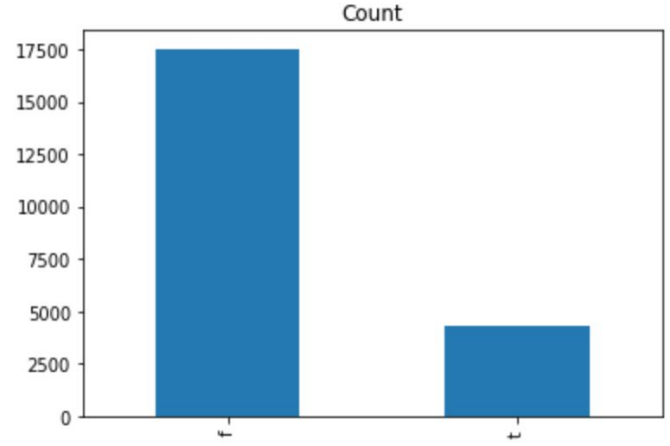
Annex 6: Histogram of Host_acceptance_rate



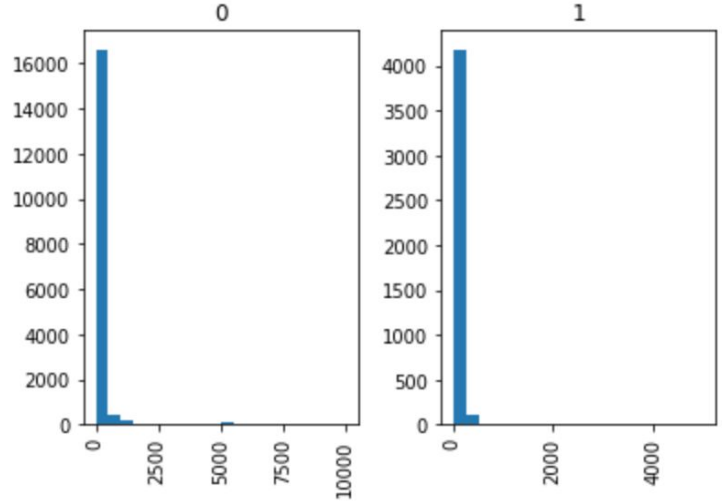
Annex 7: Boxplot of Host_acceptance_rate



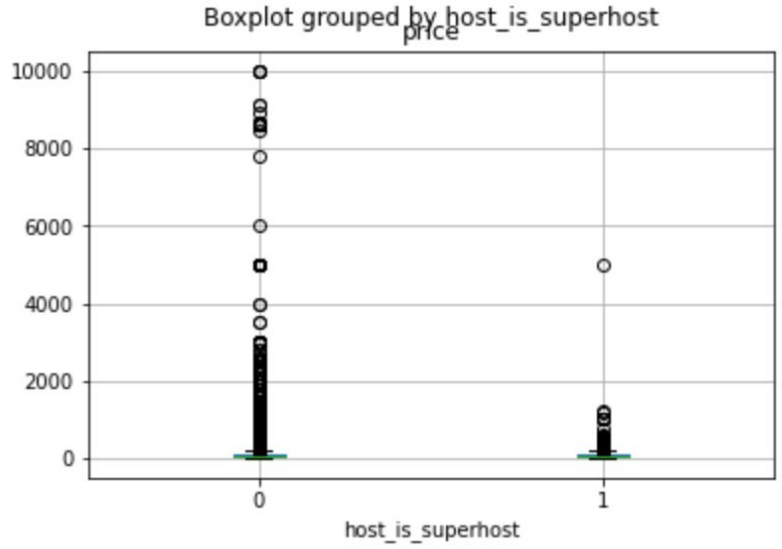
Annex 8: Barplot of Host_is_superhost



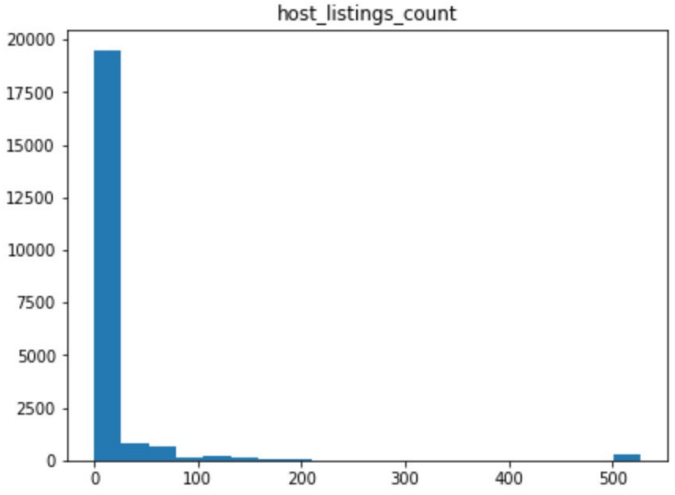
Annex 9: Histogram of Price and Host_is_superhost



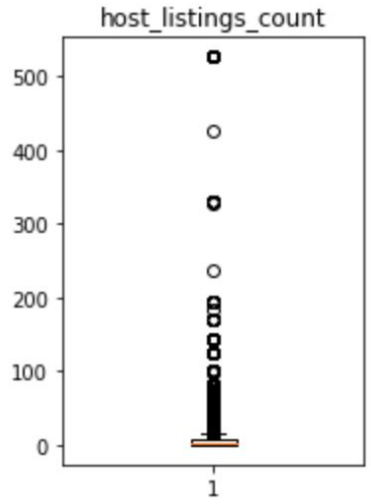
Annex 10: Boxplot of Price and Host_is_superhost



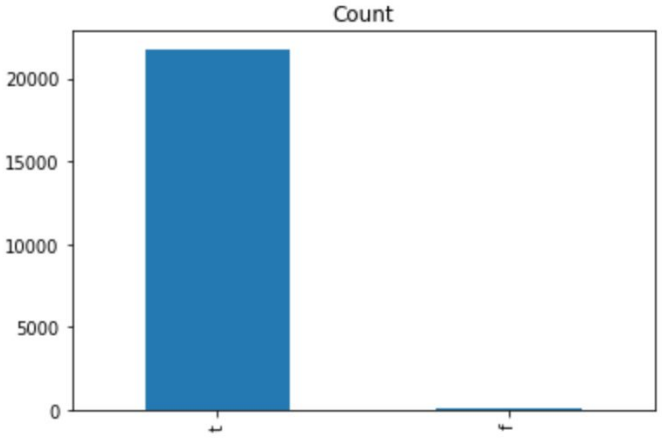
Annex 11: Histogram of Host_listings_count



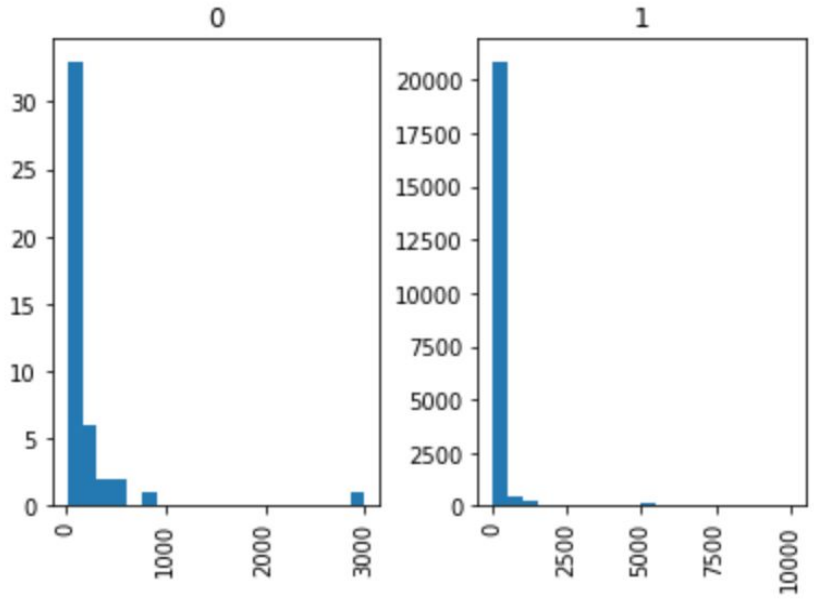
Annex 12: Boxplot of Host_listings_count



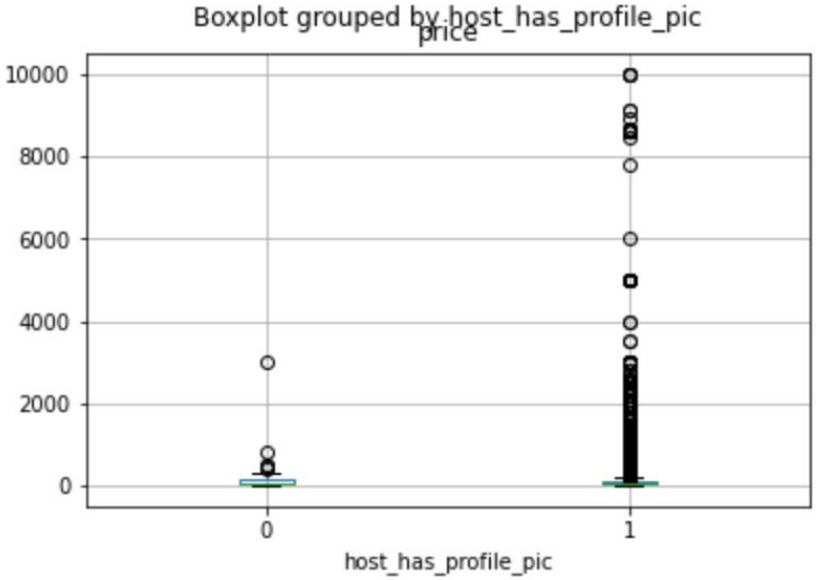
Annex 13: Barplot of Host_has_profile_pic



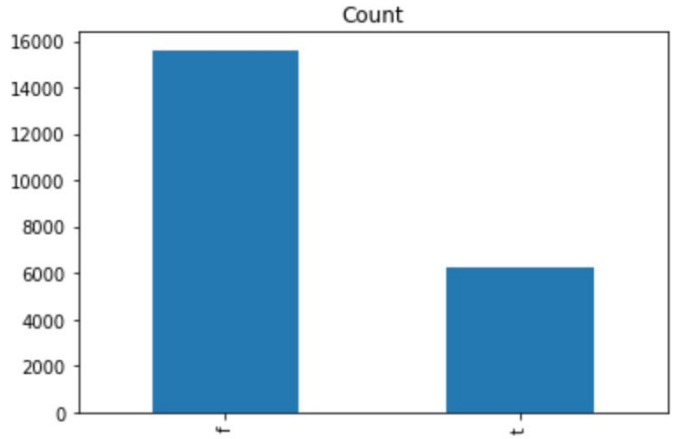
Annex 14: Histogram of Price and Host_has_profile_pic



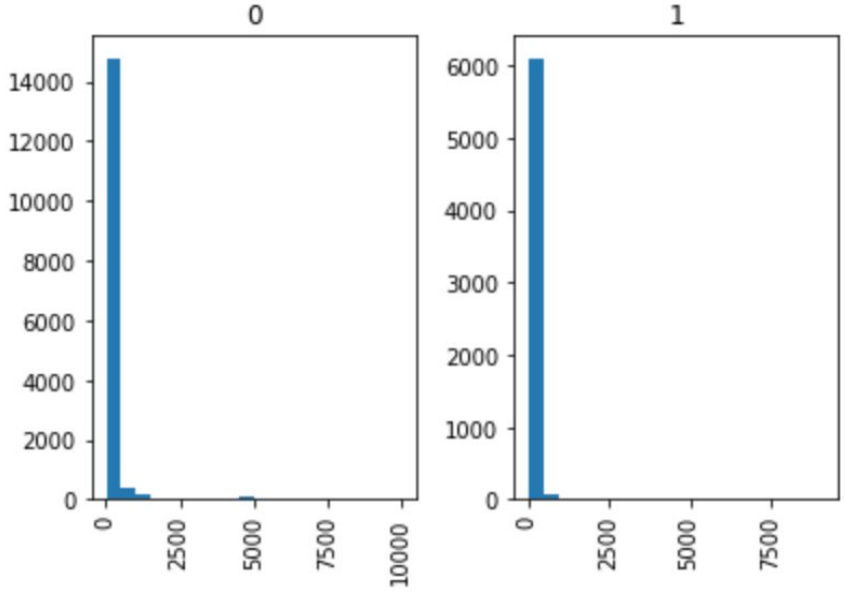
Annex 15: Boxplot of Price and Host_has_profile_pic



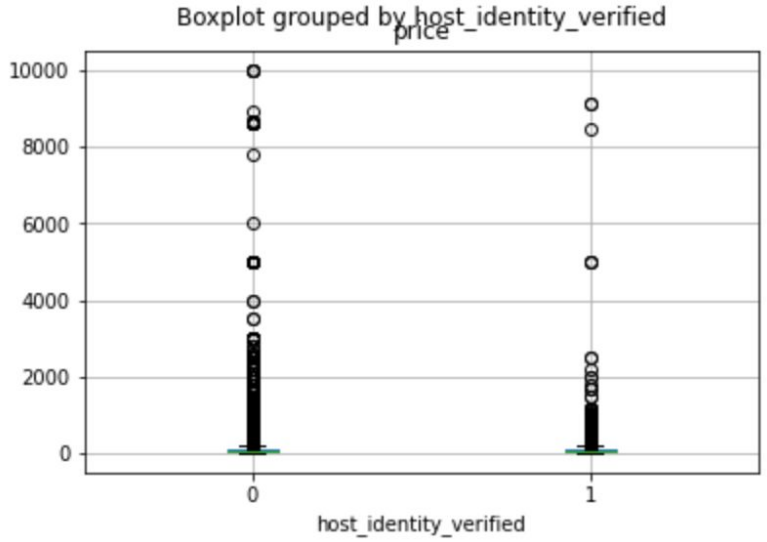
Annex 16: Barplot of Host_Identity_Verified



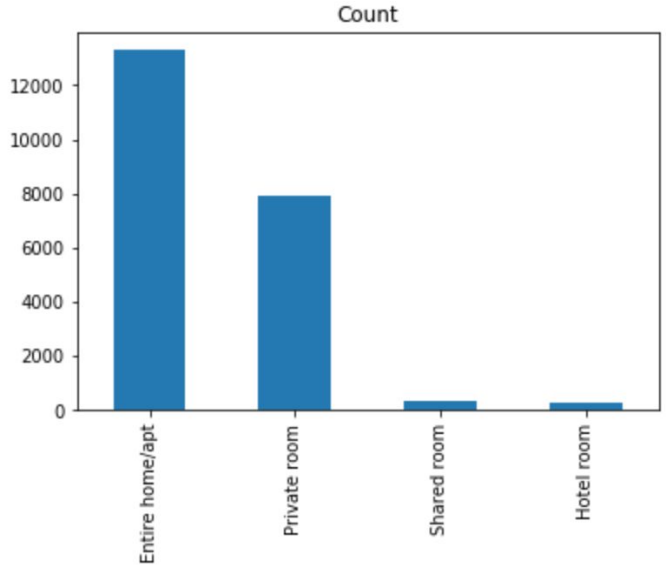
Annex 17: Histogram of Price and Host_Identity_Verified



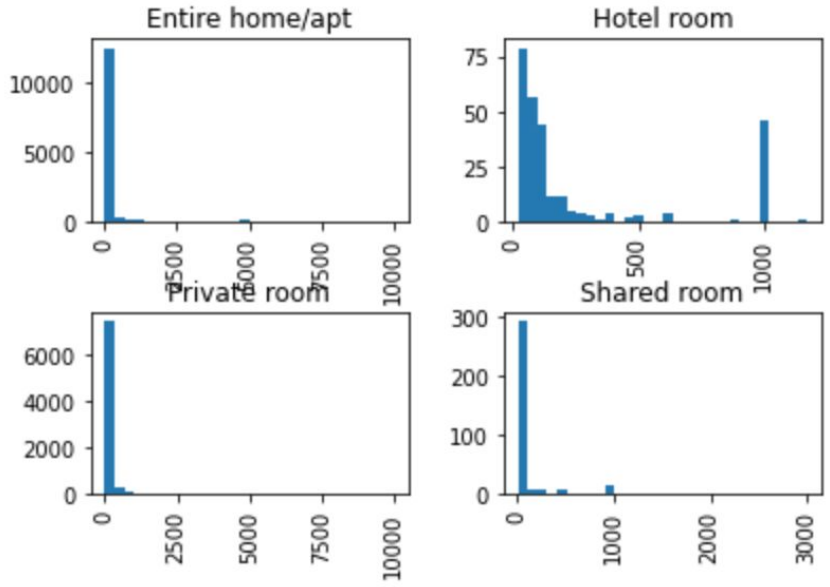
Annex 18: Boxplot of Price and Host_Identity_Verified



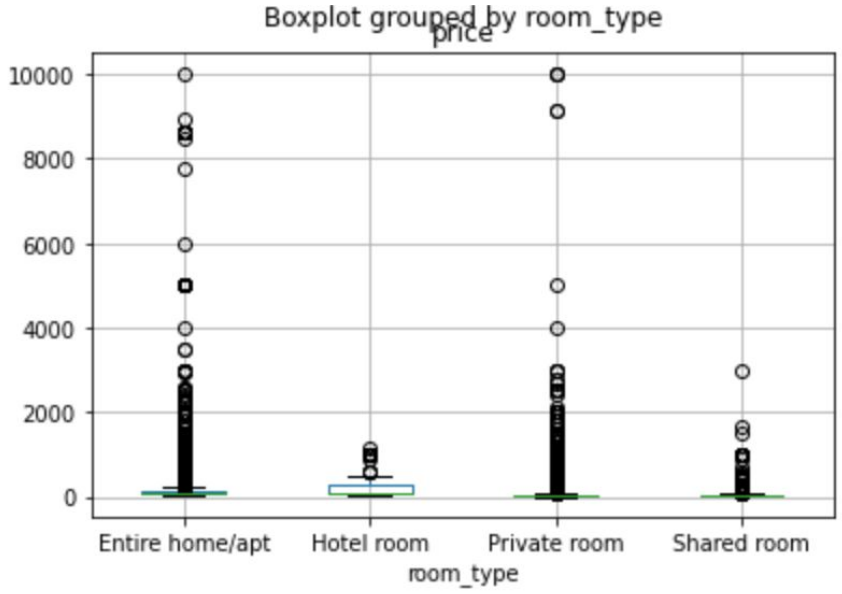
Annex 19: Barplot of Room_type



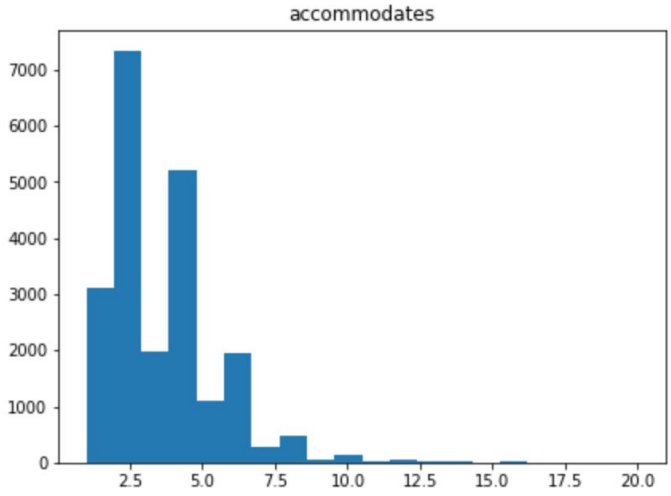
Annex 20: Histogram of Price and Room_type



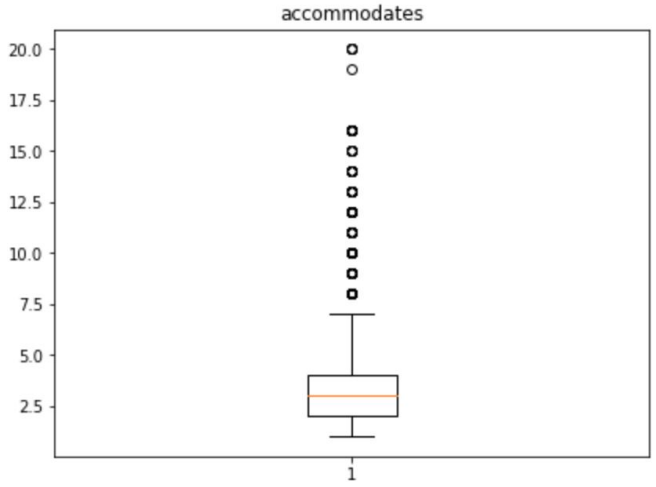
Annex 21: Boxplot of Price and Room_type



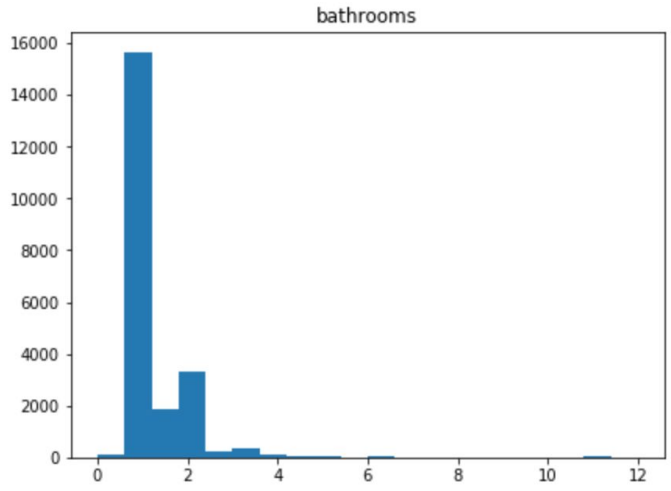
Annex 22: Histogram of Accommodates



Annex 23: Boxplot of Accommodates

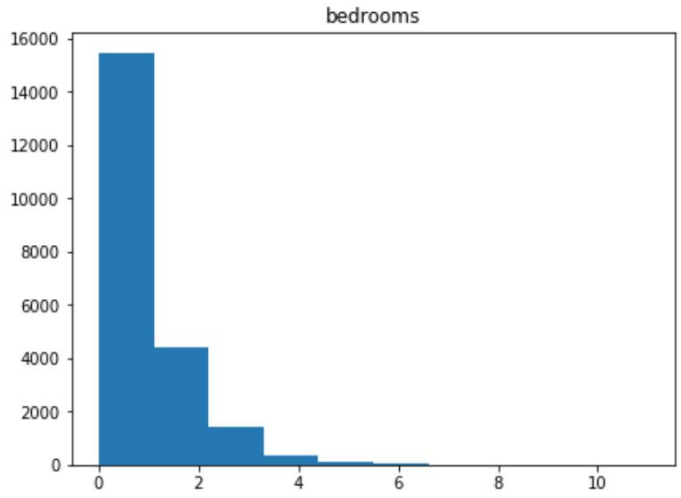


Annex 24: Histogram of Bathrooms



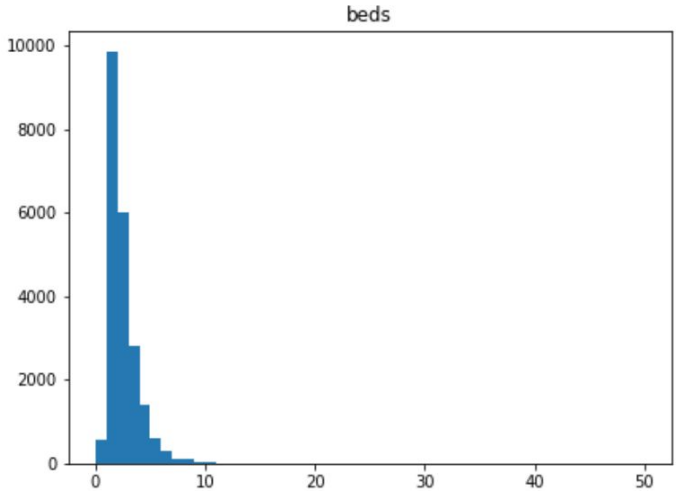
Annex 25: Boxplot of Bathrooms

Annex 26: Histogram of Bedrooms

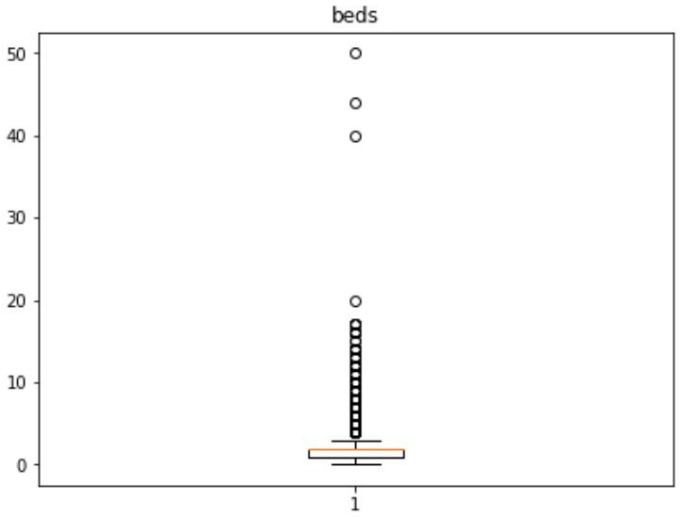


Annex 27: Boxplot of Bedrooms

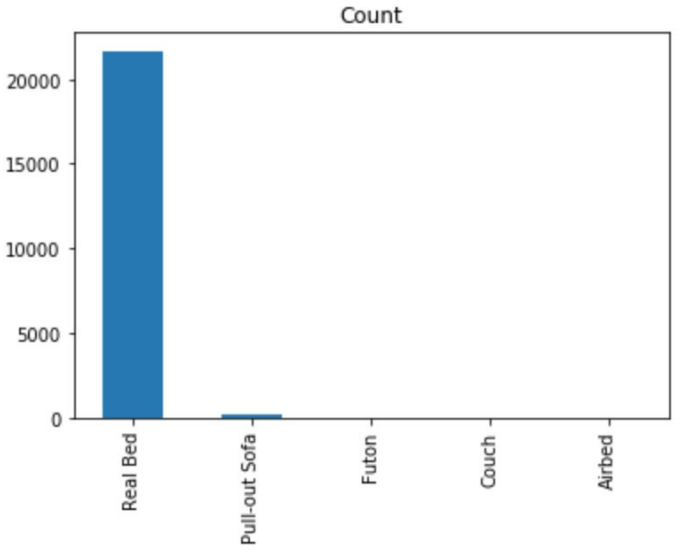
Annex 28: Histogram of Beds



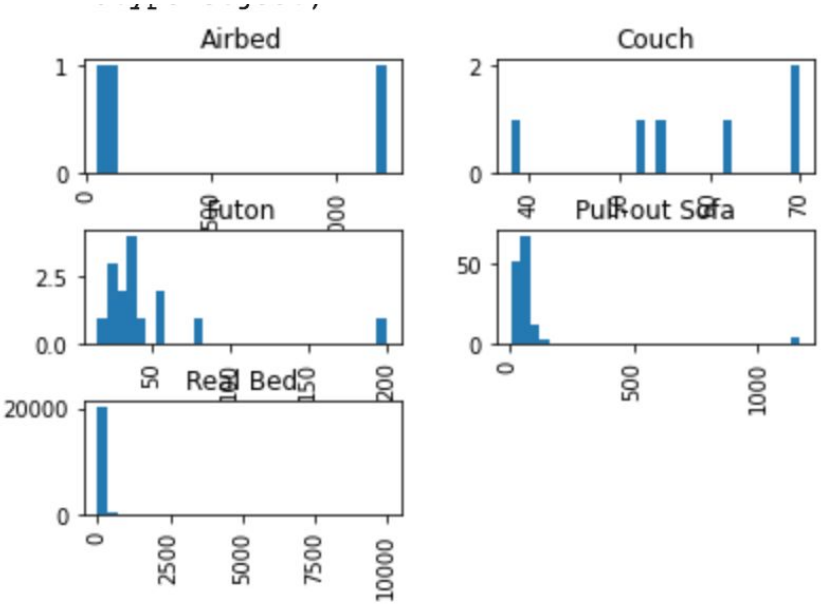
Annex 29: Boxplot of Beds



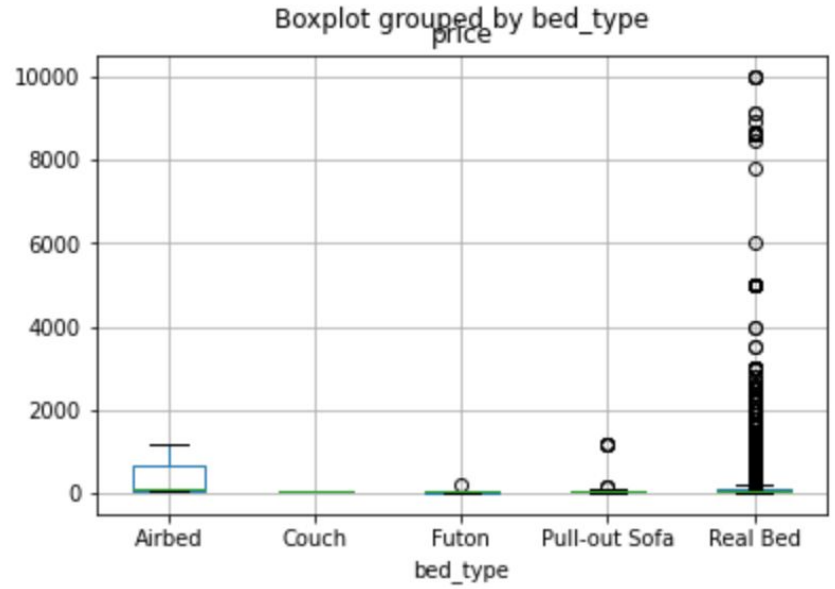
Annex 30: Barplot of Bed_type



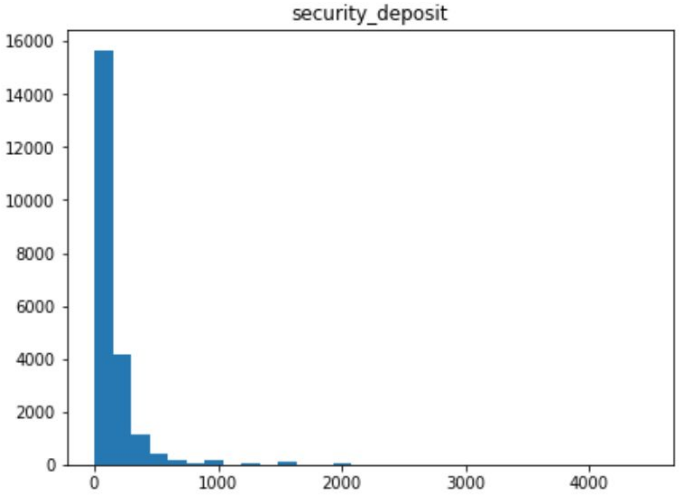
Annex 31: Histogram of Price and Bed_type



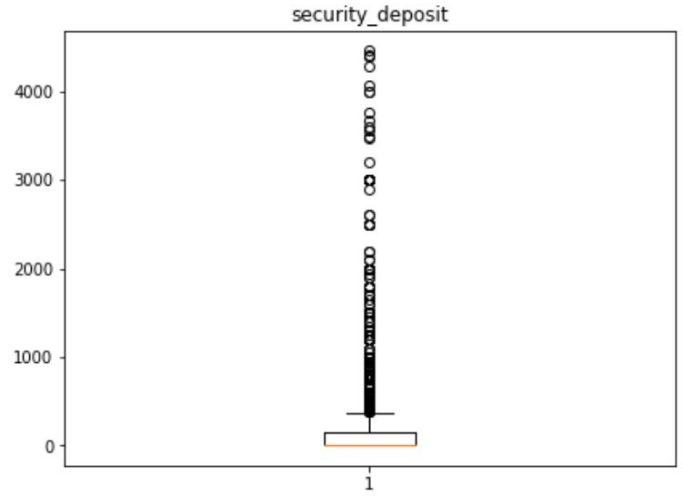
Annex 32: Boxplot of Price and Bed_type



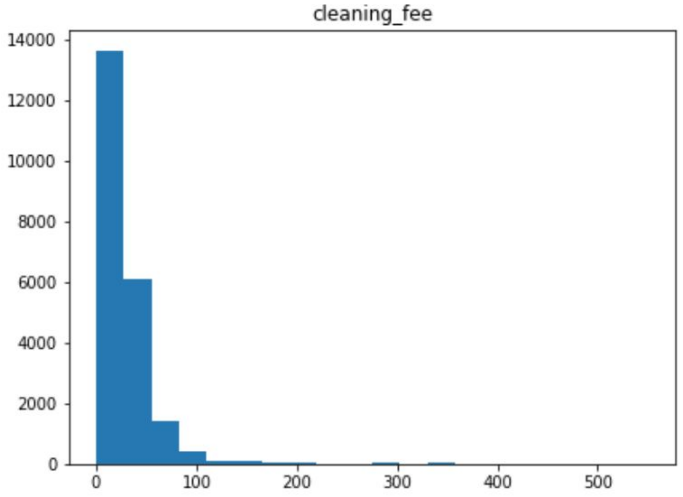
Annex 33: Histogram of Security_deposit



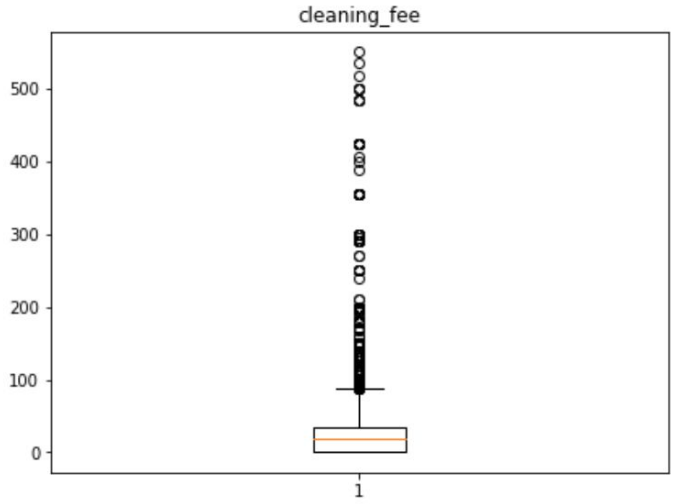
Annex 34: Boxplot of Security_deposit



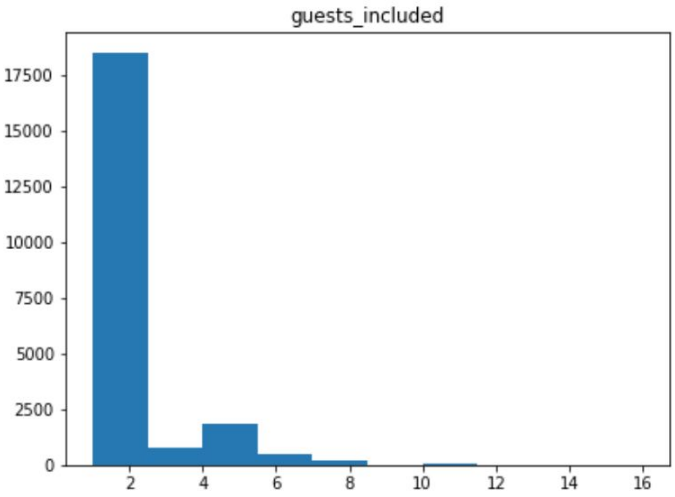
Annex 35: Histogram of Cleaning_fee



Annex 36: Boxplot of Cleaning_fee

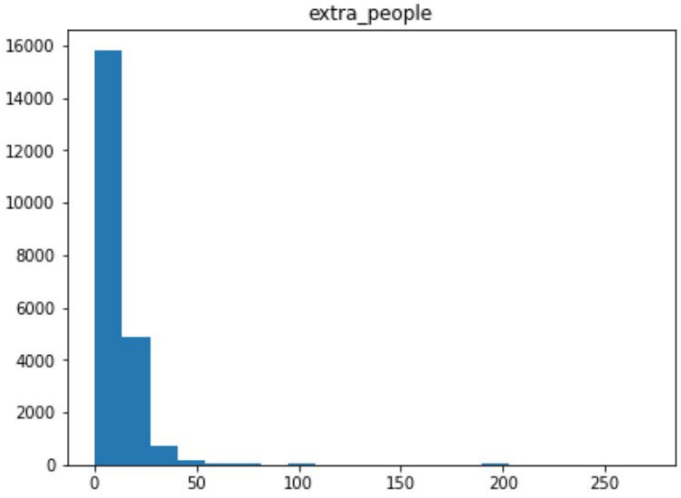


Annex 37: Histogram of Guests_included

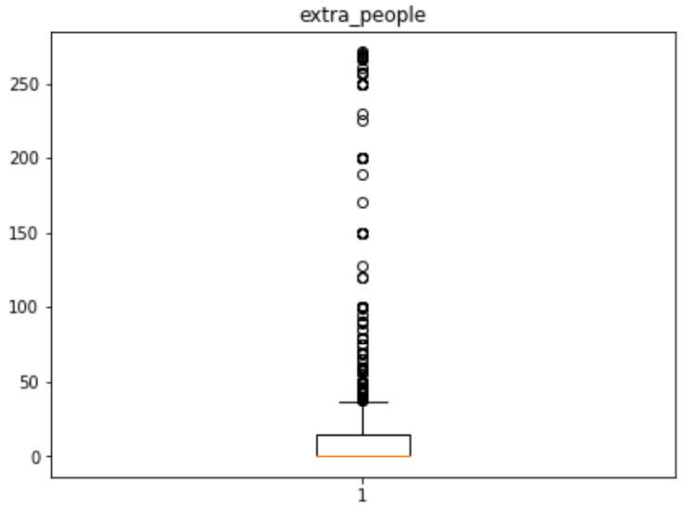


Annex 38: Boxplot of Guests_included

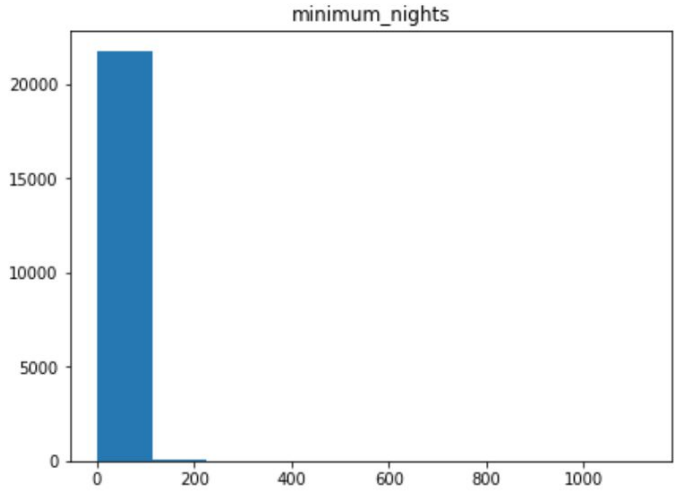
Annex 39: Histogram of Extra_people



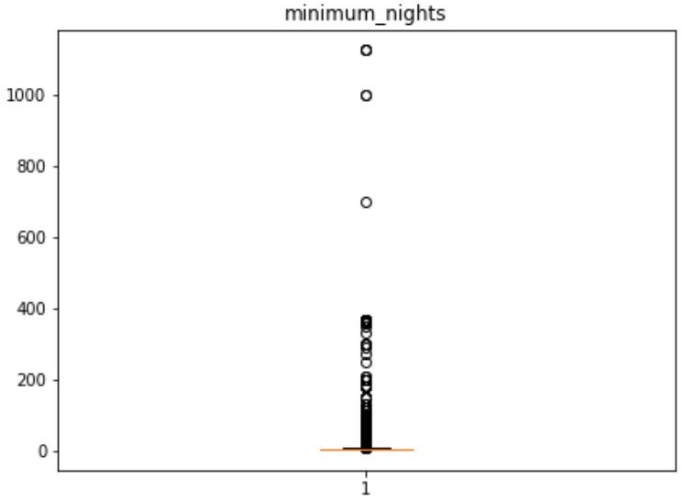
Annex 40: Boxplot of Extra_people



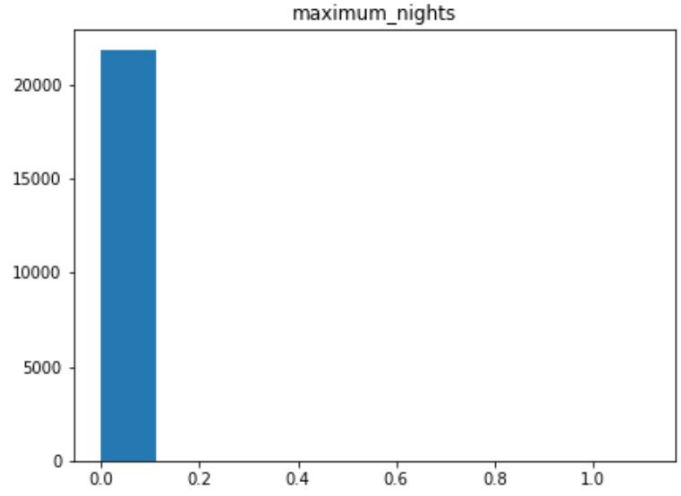
Annex 41: Histogram of Minimum_nights



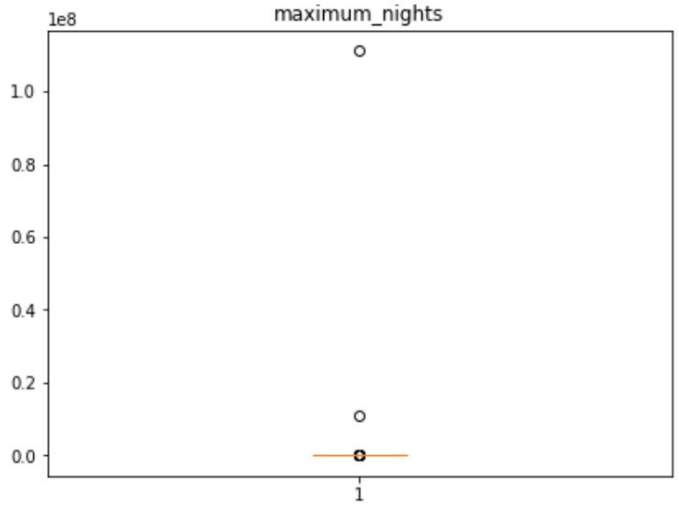
Annex 42: Boxplot of Minimum_nights



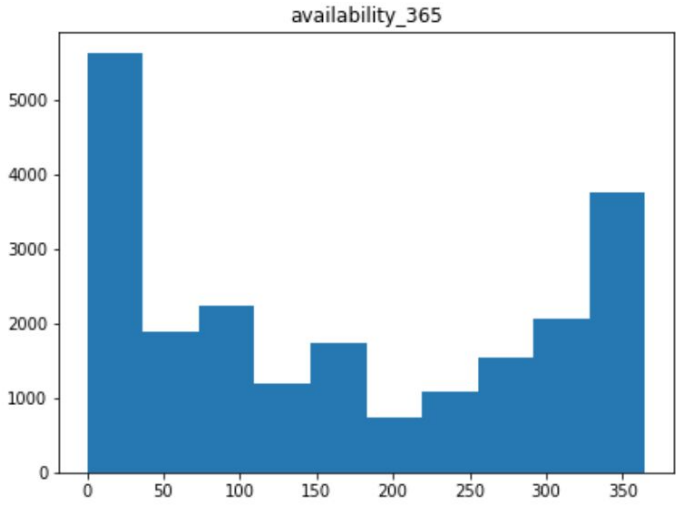
Annex 43: Histogram of Maximum_nights



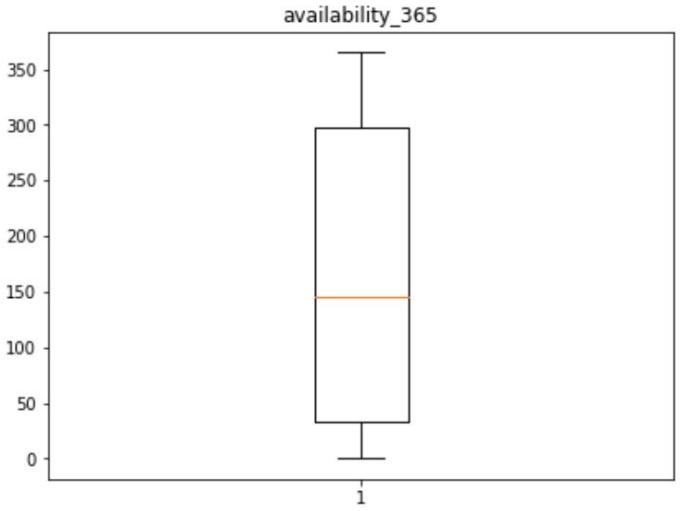
Annex 44: Boxplot of Maximum_nights



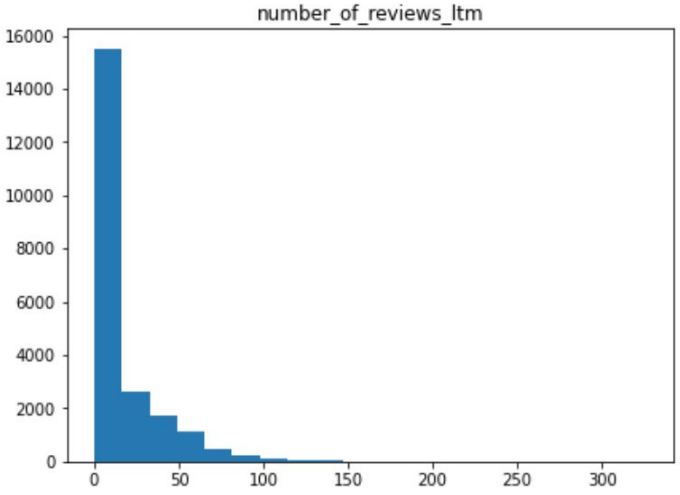
Annex 45: Histogram of Availability_365



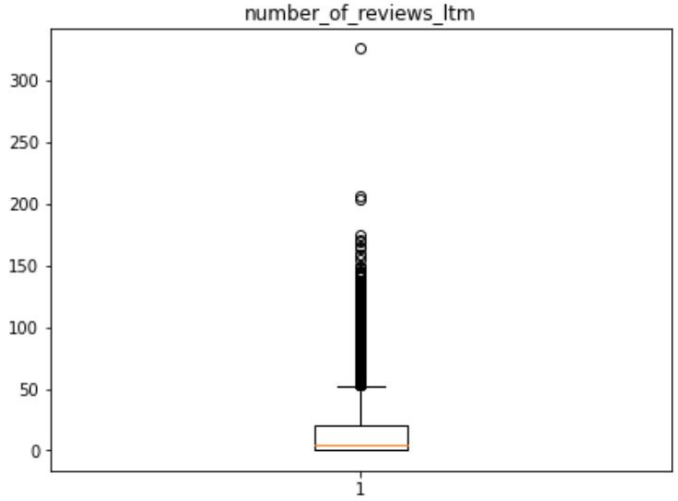
Annex 46: Boxplot of Availability_365



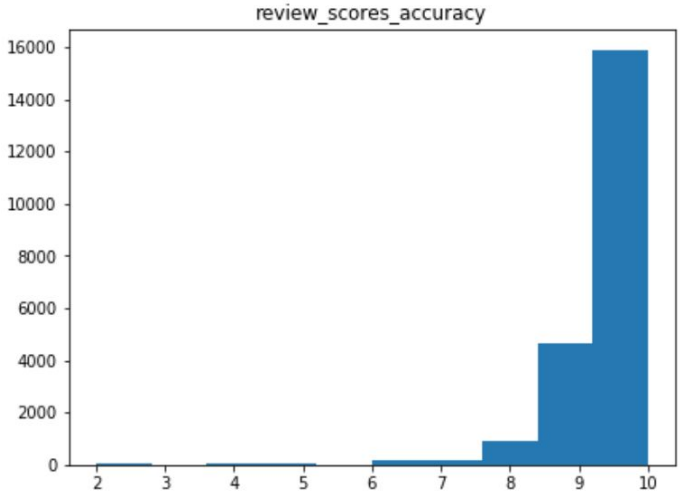
Annex 47: Histogram of Number_of_reviews_ltm



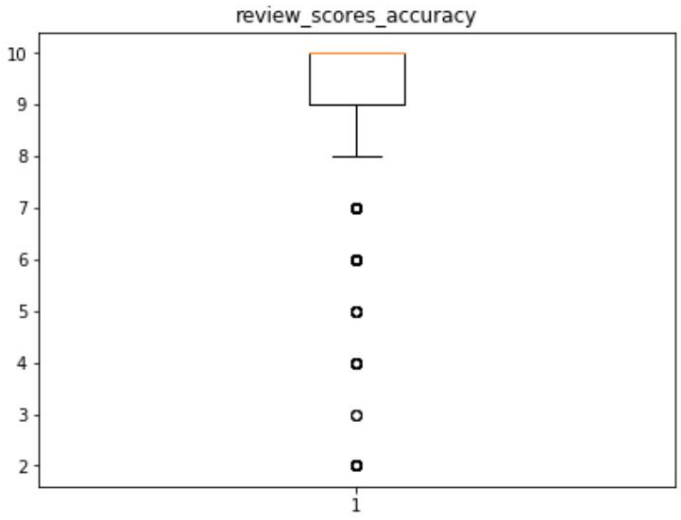
Annex 48: Boxplot of Number_of_reviews_ltm



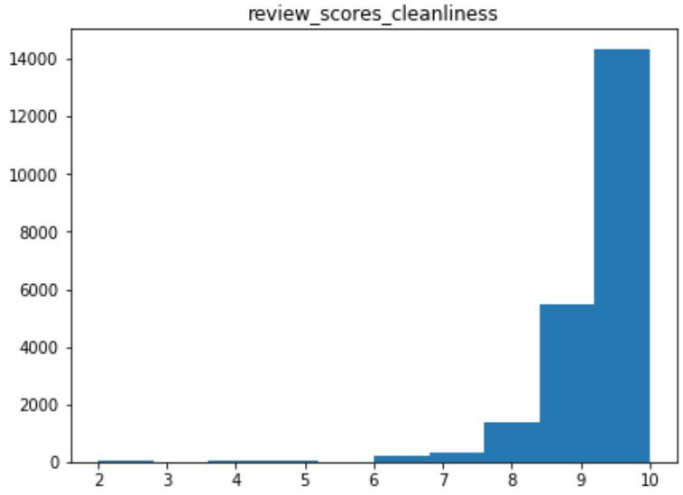
Annex 49: Histogram of Review_scores_accuracy



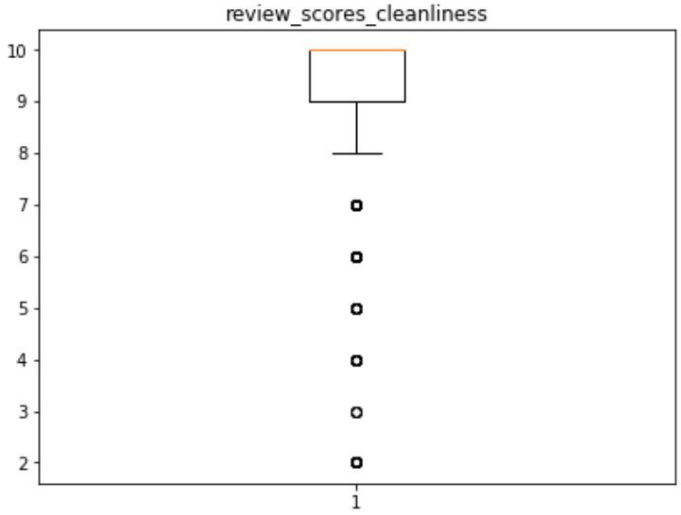
Annex 50: Boxplot of Review_scores_accuracy



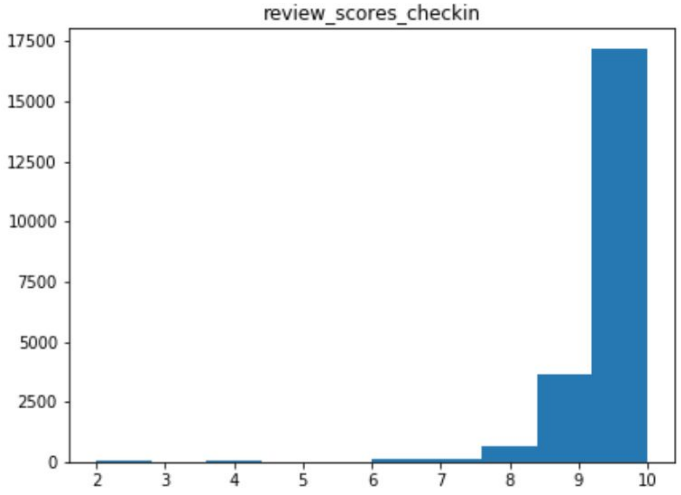
Annex 51: Histogram of Review_scores_cleanliness



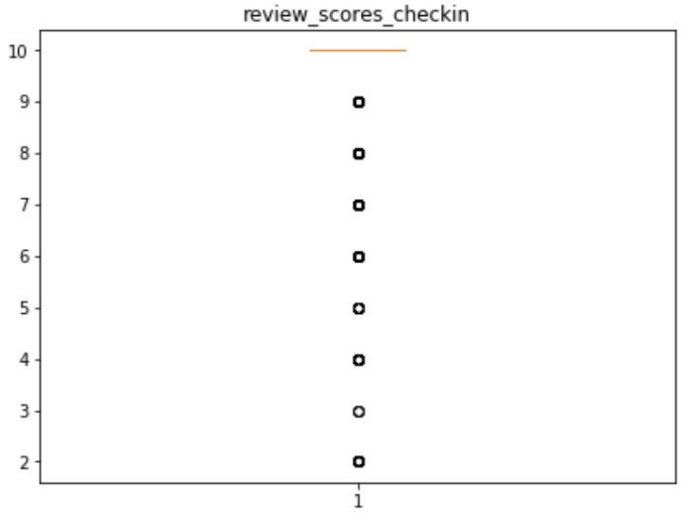
Annex 52: Boxplot of Review_scores_cleanliness



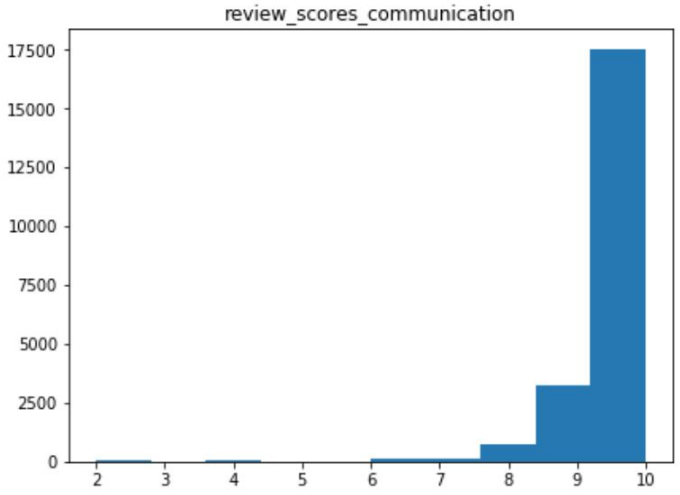
Annex 53: Histogram of Review_scores_checkin



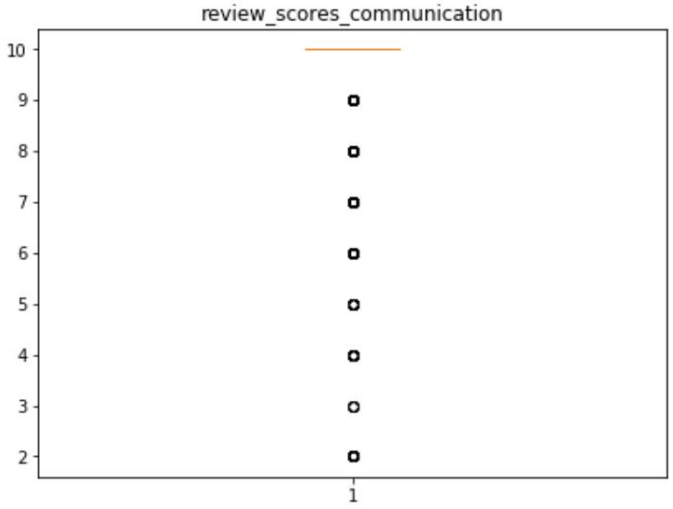
Annex 54: Boxplot of Review_scores_checkin



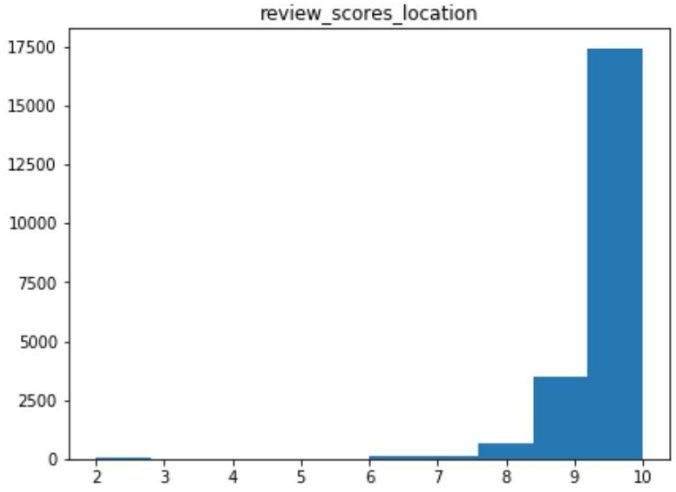
Annex 55: Histogram of Review_scores_communication



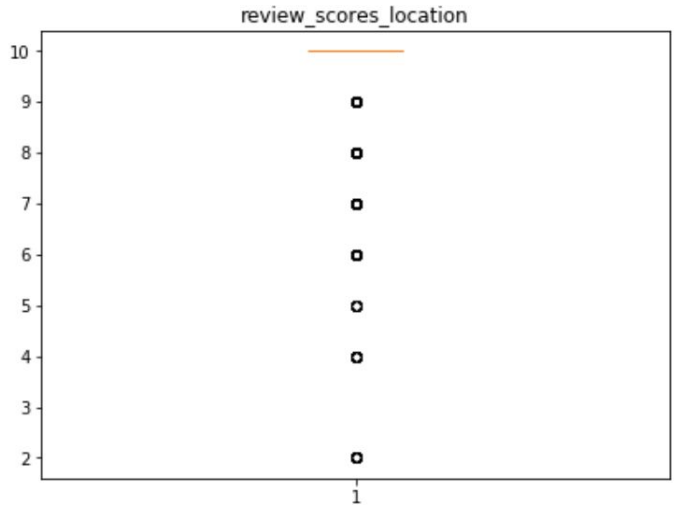
Annex 56: Boxplot of Review_scores_communication



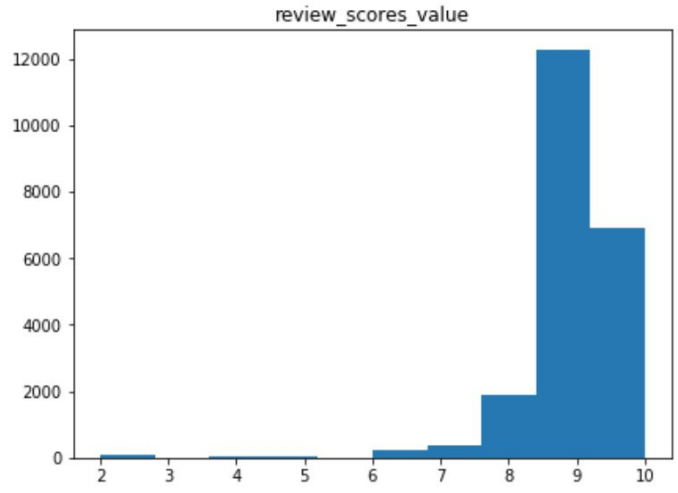
Annex 57: Histogram of Review_scores_location



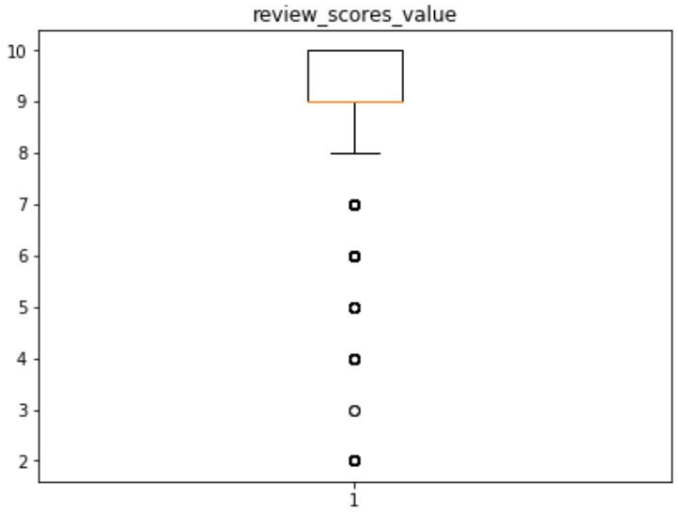
Annex 58: Boxplot of Review_scores_location



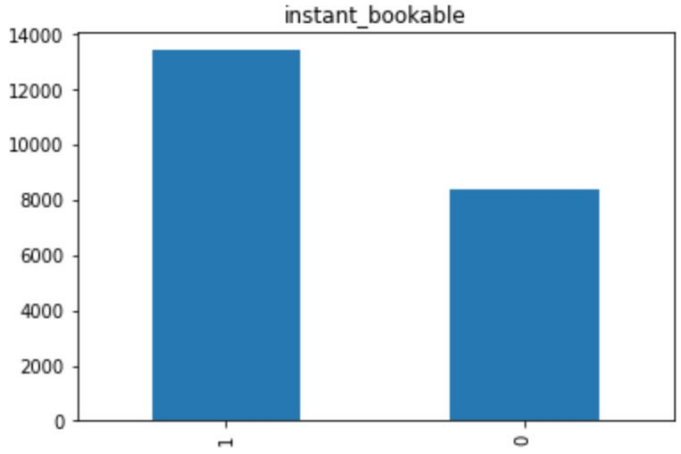
Annex 59: Histogram of Review_scores_value



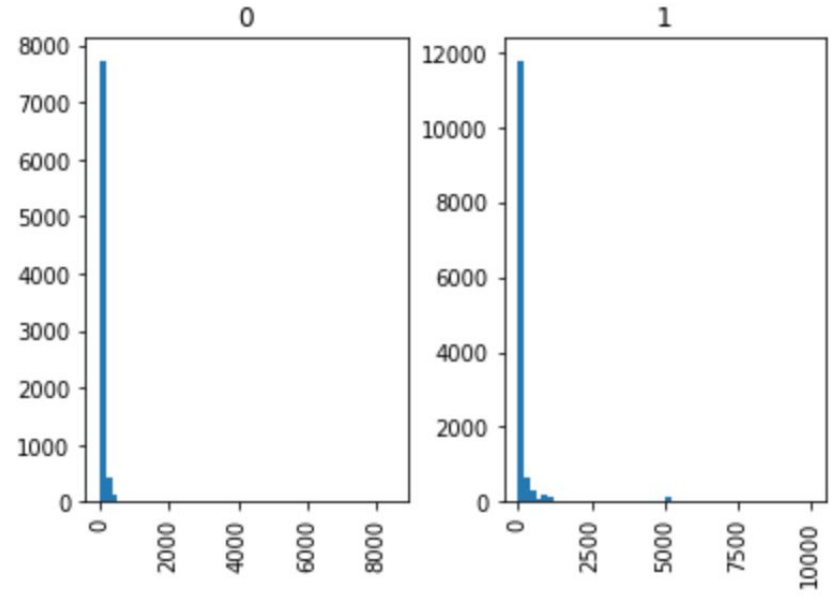
Annex 60: Boxplot of Review_scores_value



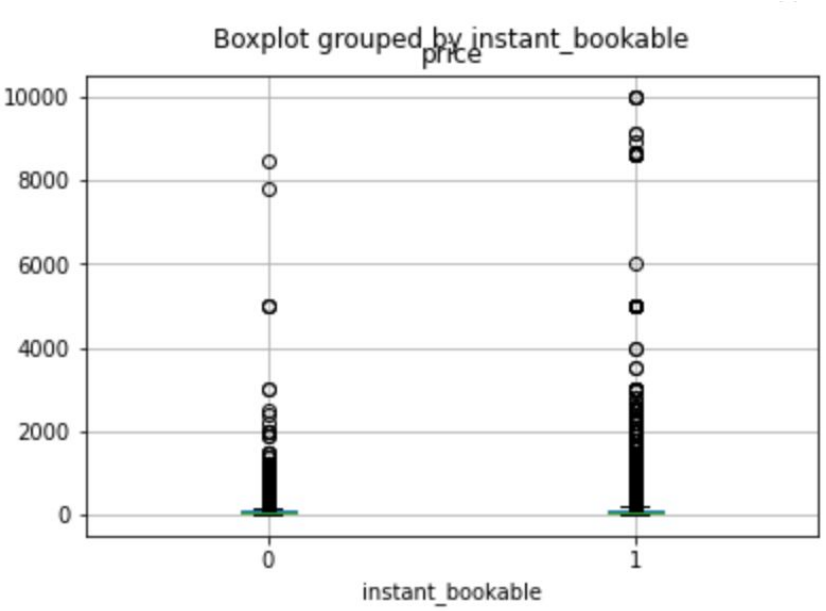
Annex 61: Barplot of Instant_bookable



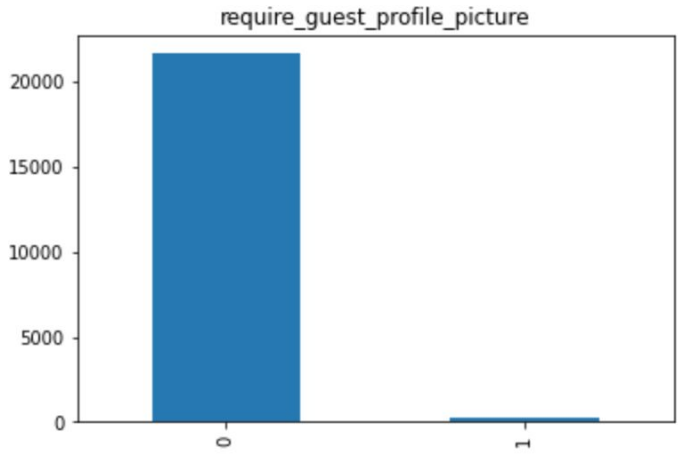
Annex 62: Histogram of Price and Instant_bookable



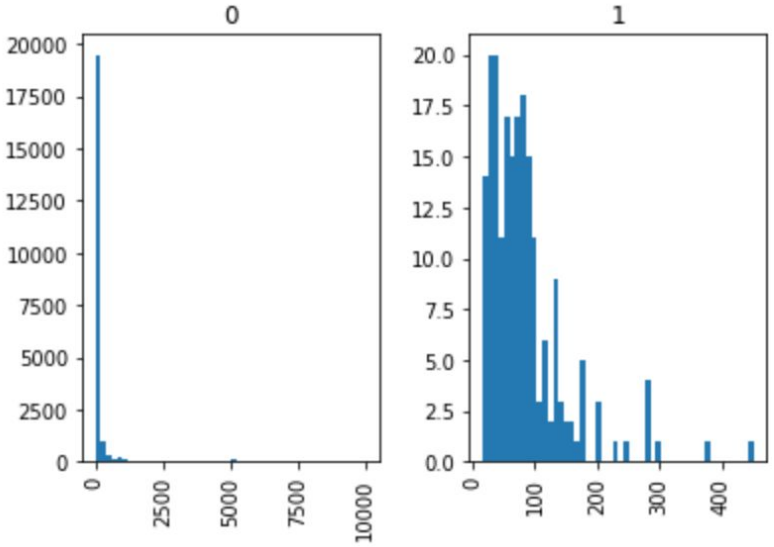
Annex 63: Boxplot of Price and Instant_bookable



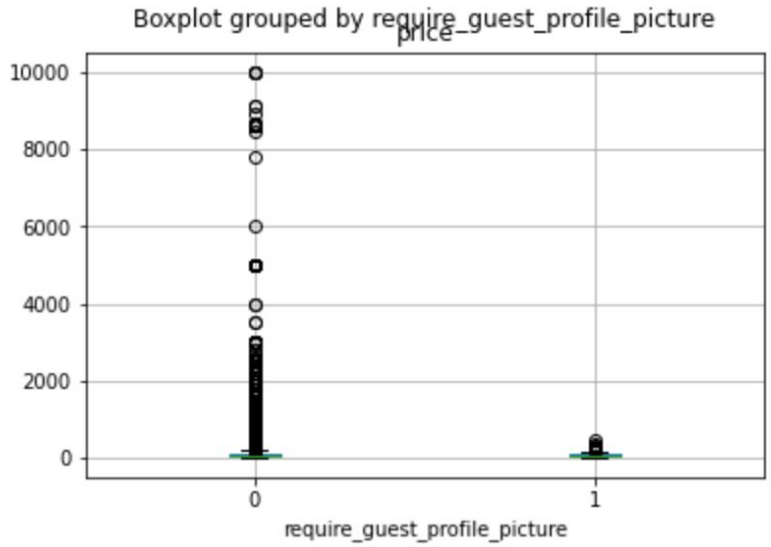
Annex 64: Barplot of Require_guest_profile_picture



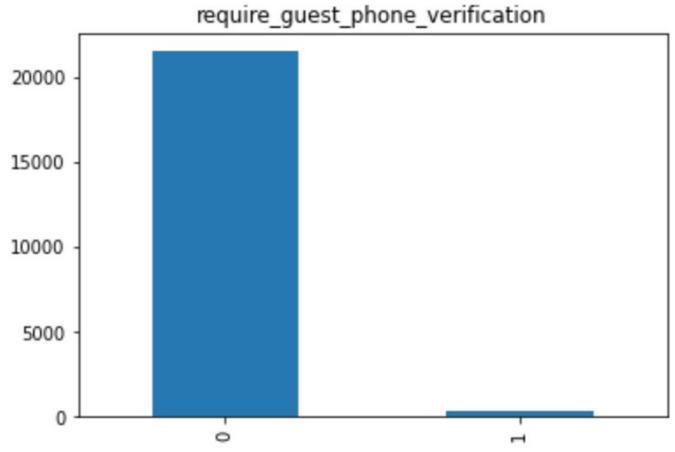
Annex 65: Histogram of Price and Require_guest_profile_picture



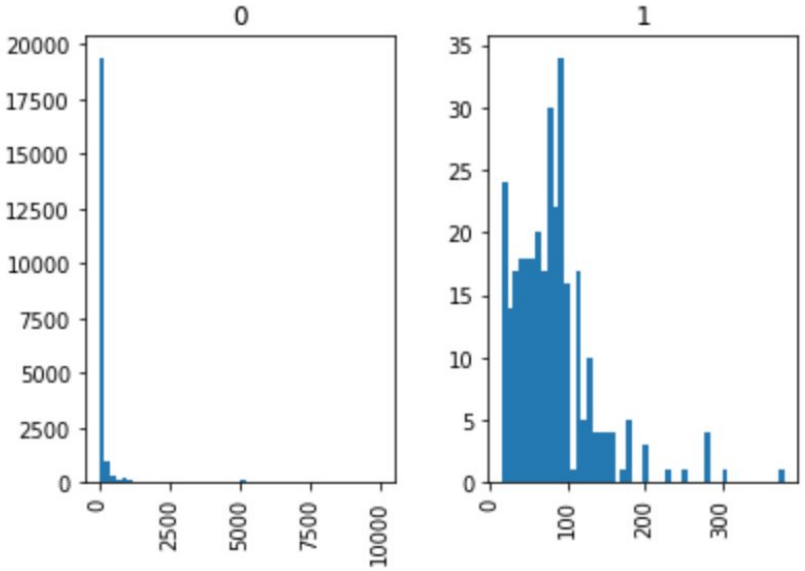
Annex 66: Boxplot of Price and Require_guest_profile_picture



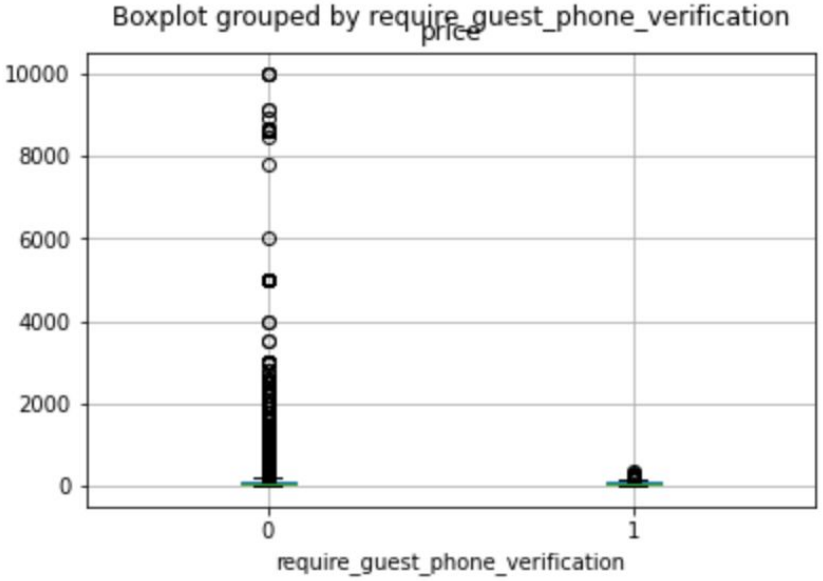
Annex 67: Barplot of Require_guest_phone_verification



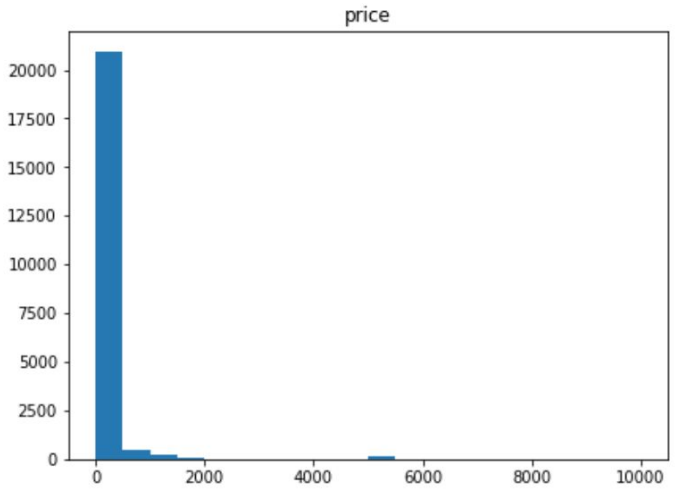
Annex 68: Histogram of Price and Require_guest_phone_verification



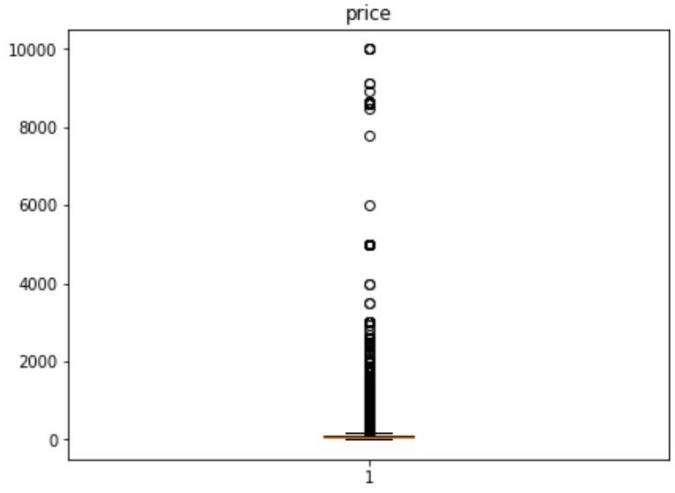
Annex 69: Boxplot of Price and Require_guest_phone_verification



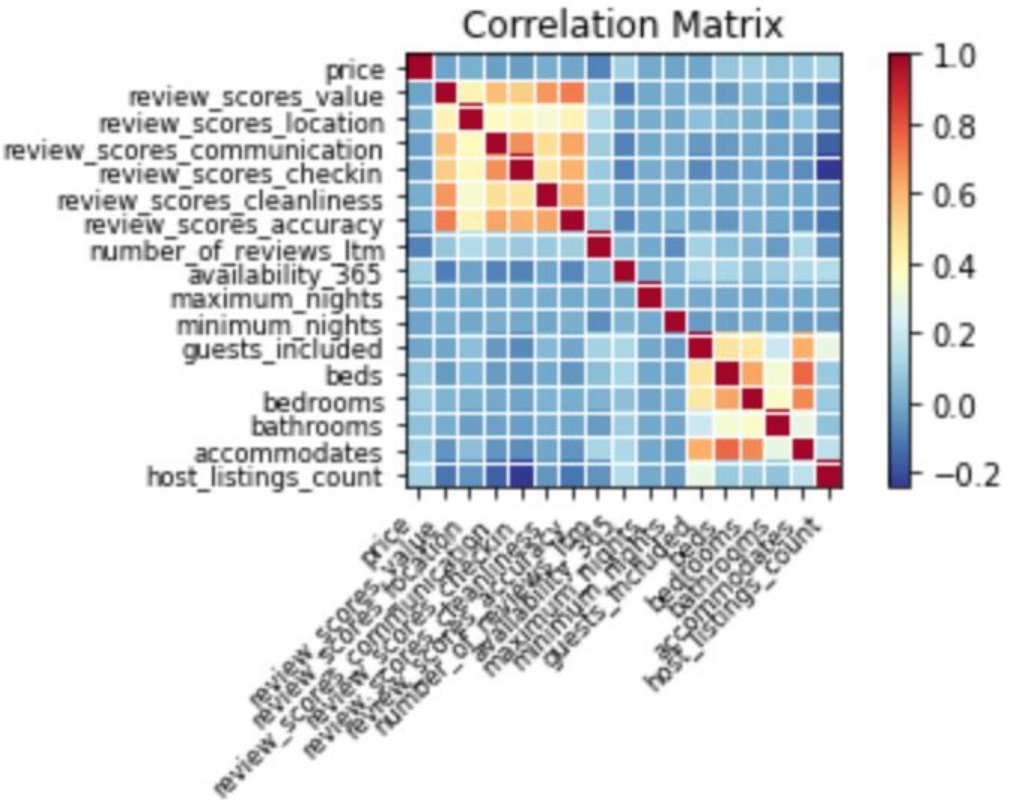
Annex 70: Histogram of Price



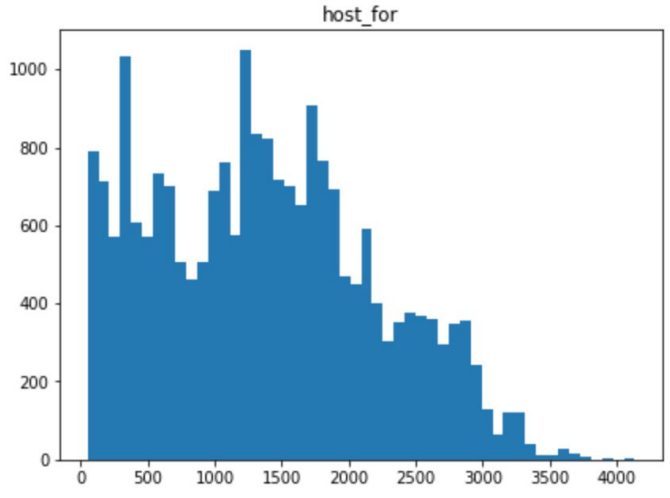
Annex 71: Boxplot of Price



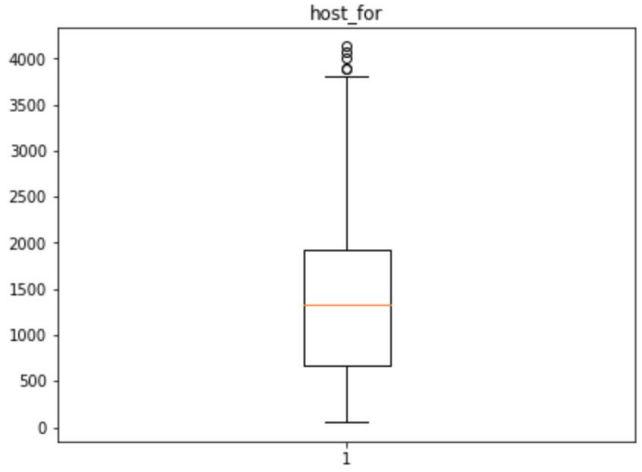
Annex 72: Correlation Matrix



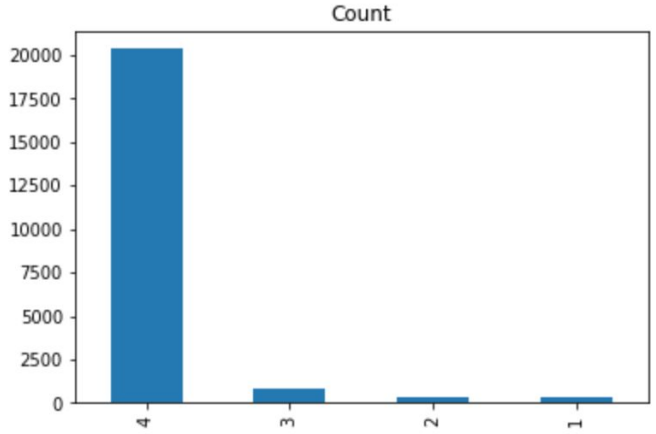
Annex 73: Histogram of Host_since



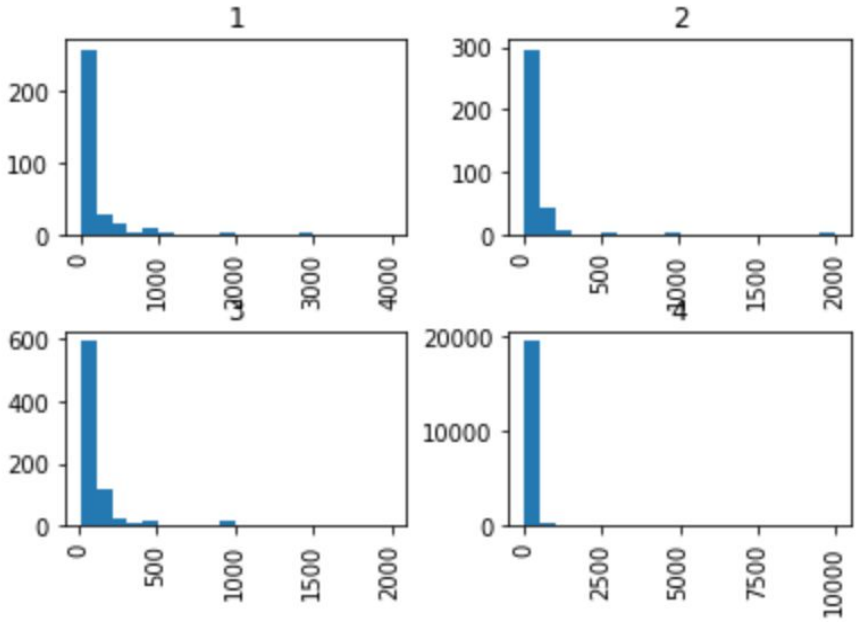
Annex 74: Boxplot of Host_since



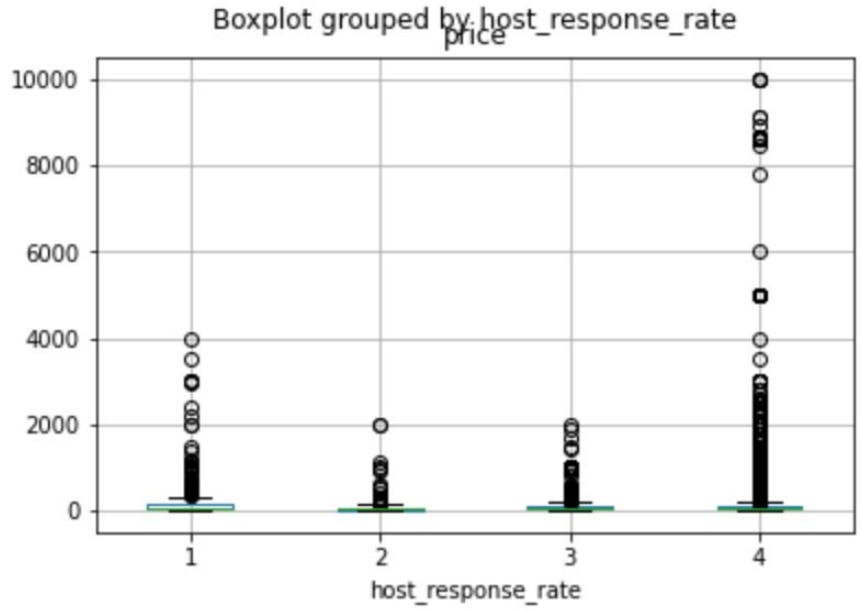
Annex 75: Barplot of Host_response_rate



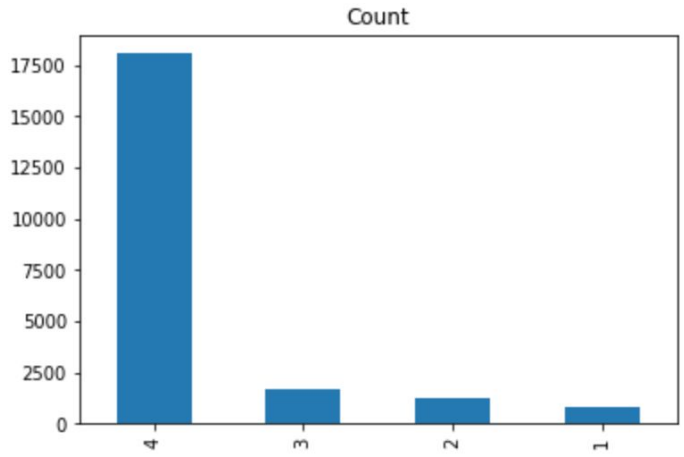
Annex 76: Histogram of Price and Host_response_rate



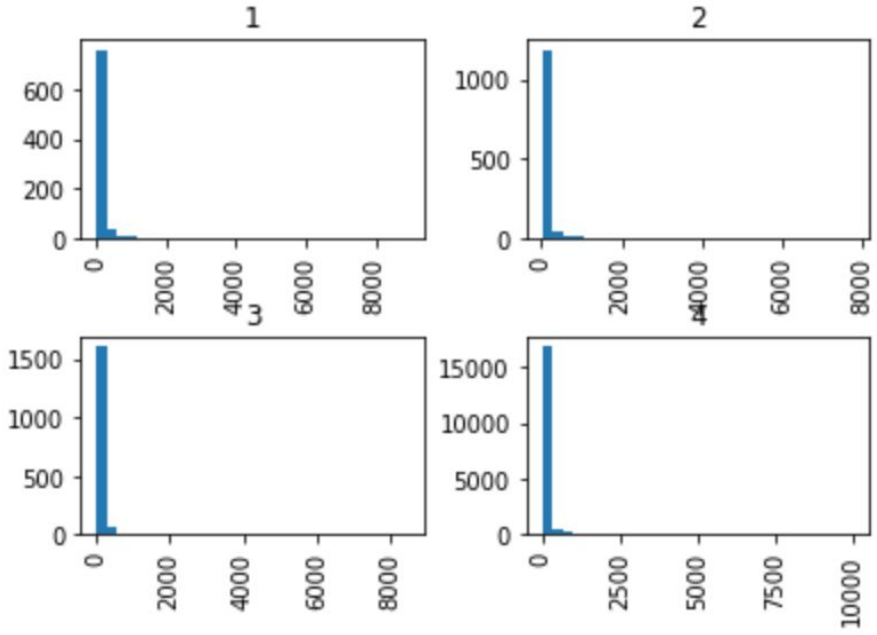
Annex 77: Boxplot of Price and Host_response_rate



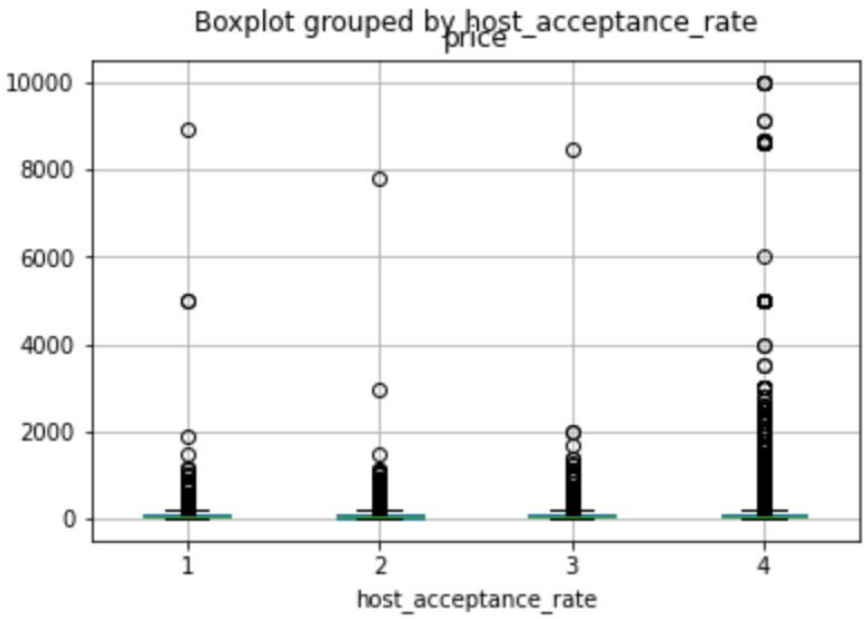
Annex 78: Barplot of Host_acceptance_rate



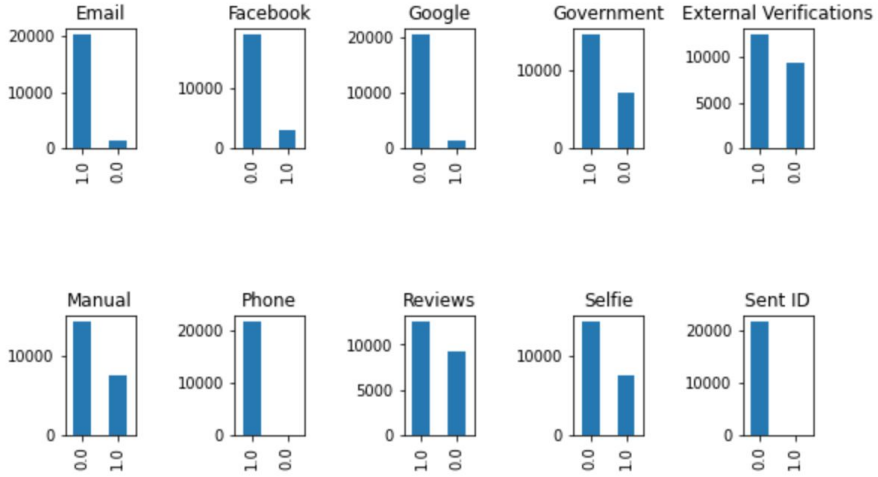
Annex 79: Histogram of Price and Host_acceptance_rate



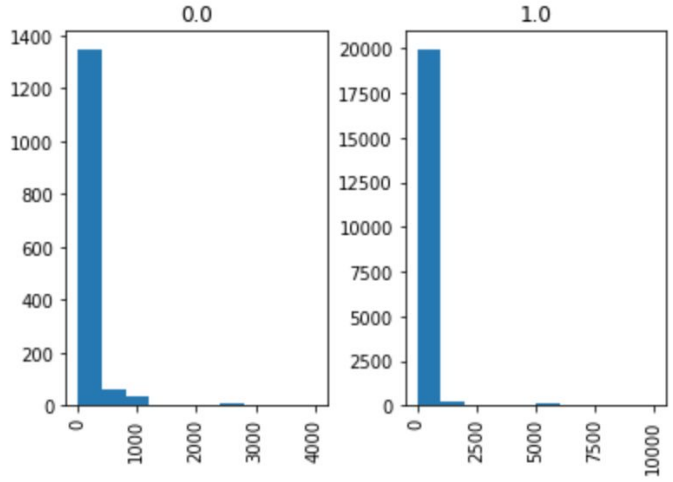
Annex 80: Boxplot of Price and Host_acceptance_rate



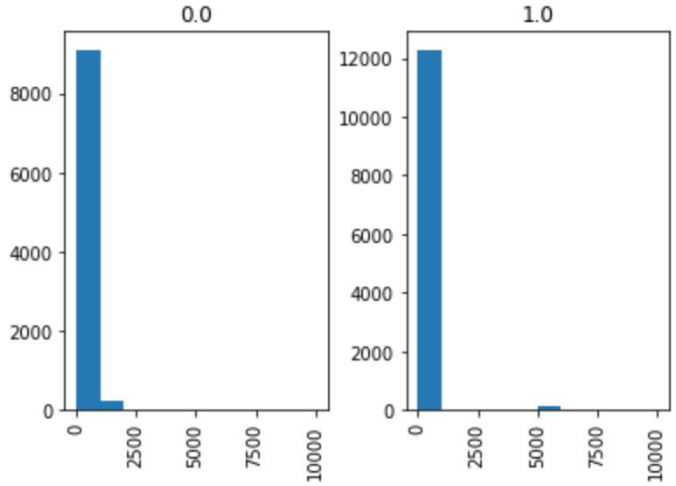
Annex 81: Value Counts Host_verification



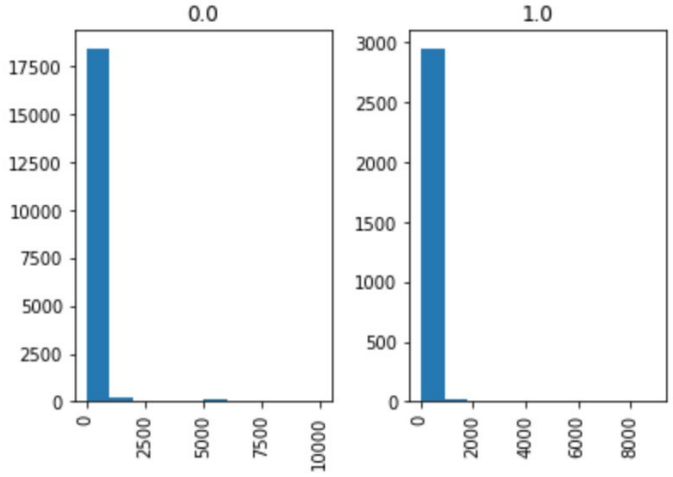
Annex 82: Histogram for Price and Email



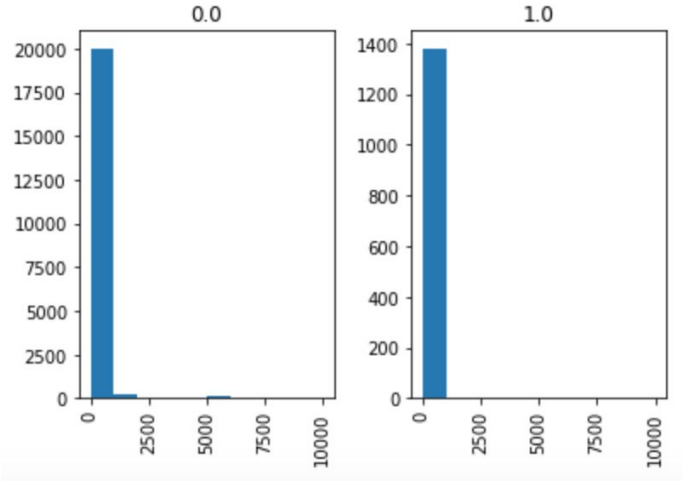
Annex 83: Histogram for Price and External Verifications



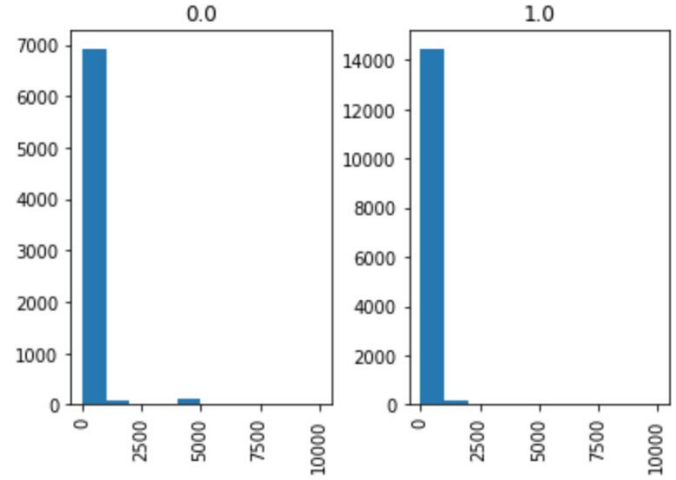
Annex 84: Histogram for Price and Facebook



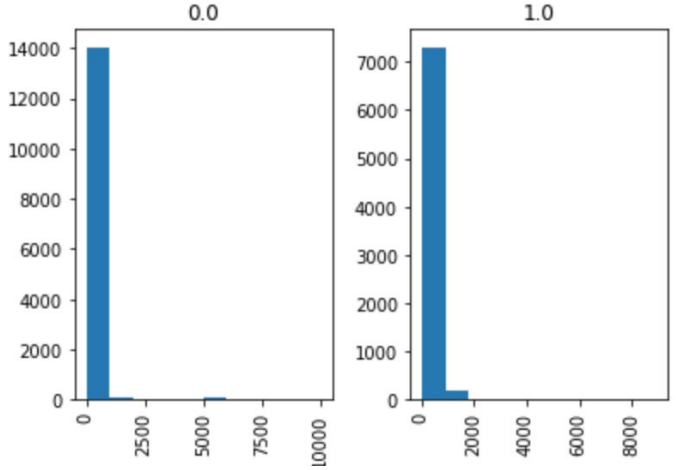
Annex 85: Histogram for Price and Google



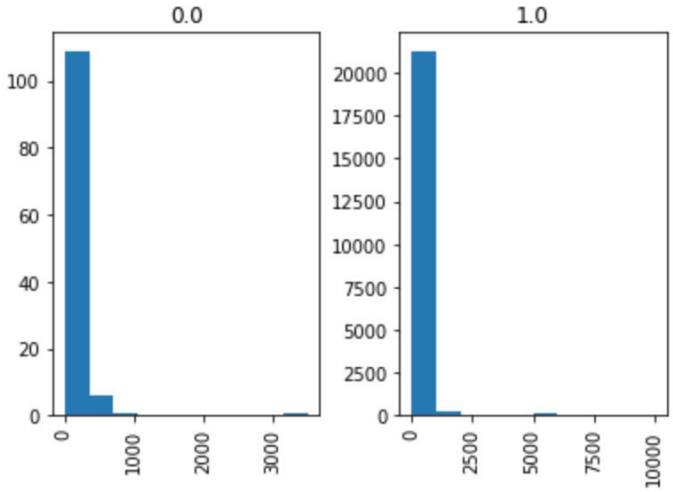
Annex 86: Histogram for Price and Government



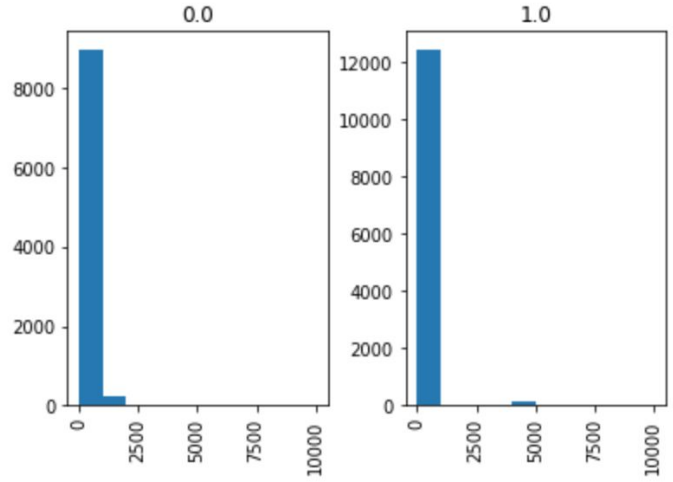
Annex 87: Histogram for Price and Manual



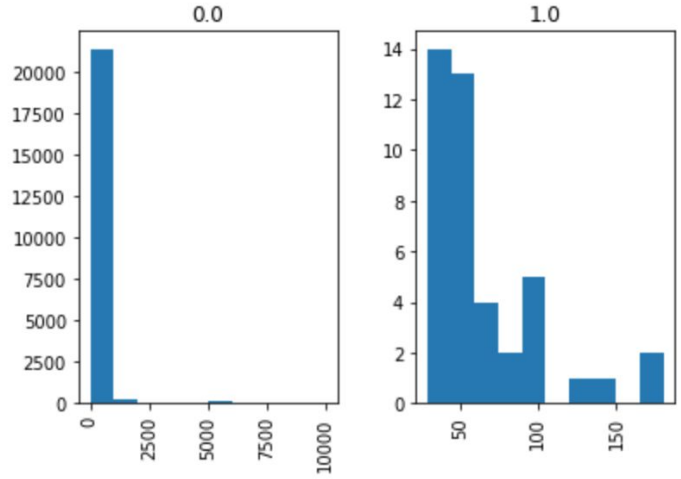
Annex 88: Histogram for Price and Phone



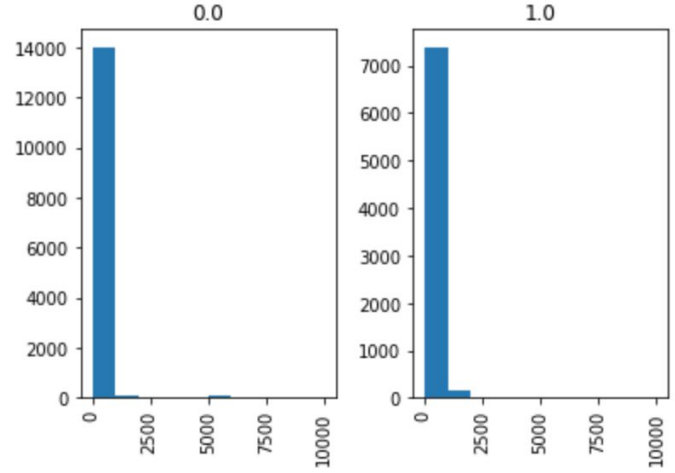
Annex 89: Histogram for Price and Reviews



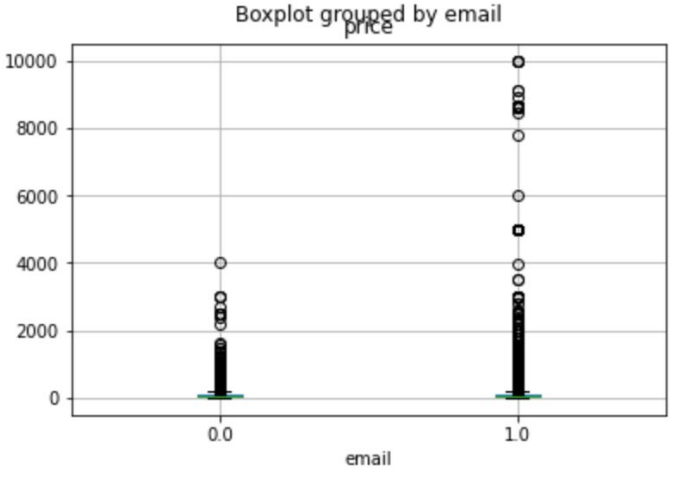
Annex 90: Histogram for Price and Sent-id



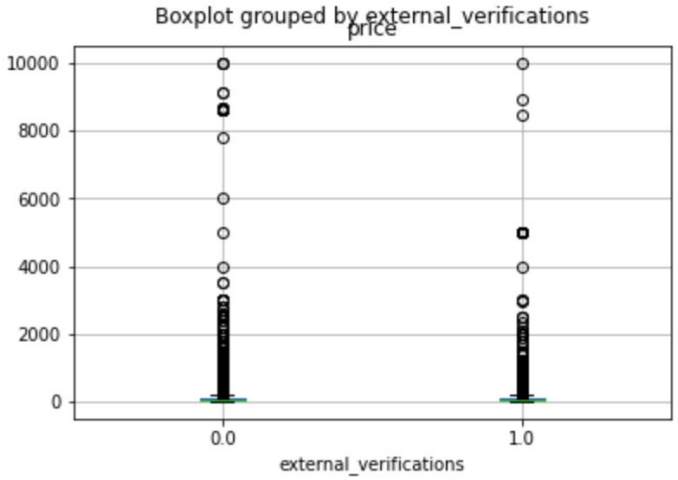
Annex 91: Histogram for Price and Selfie



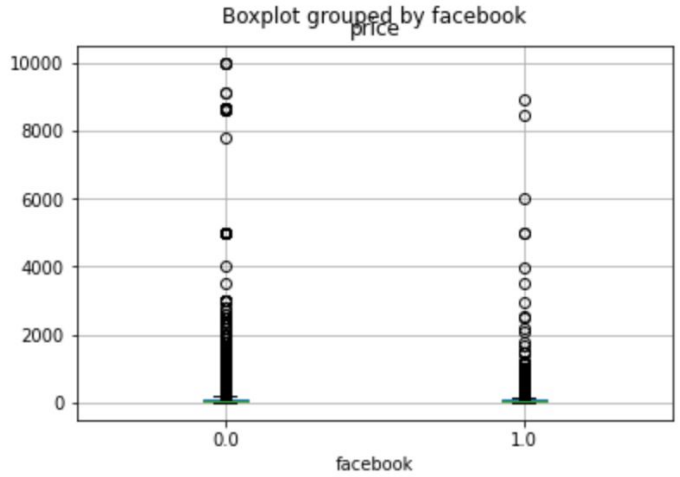
Annex 92: Boxplot for Price and Email



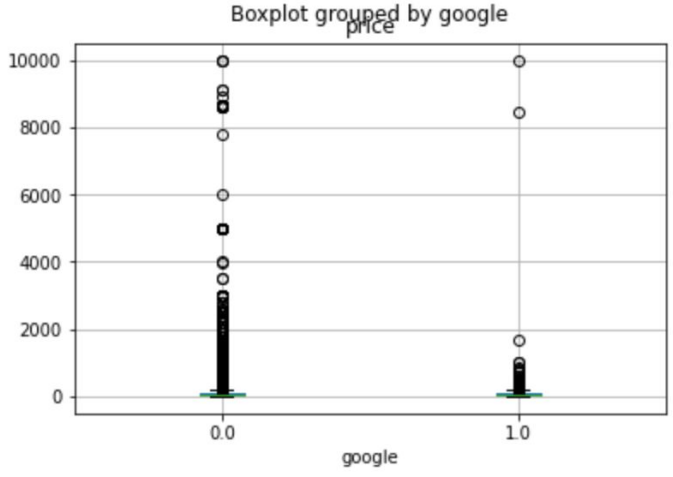
Annex 93: Boxplot for Price and External Verifications



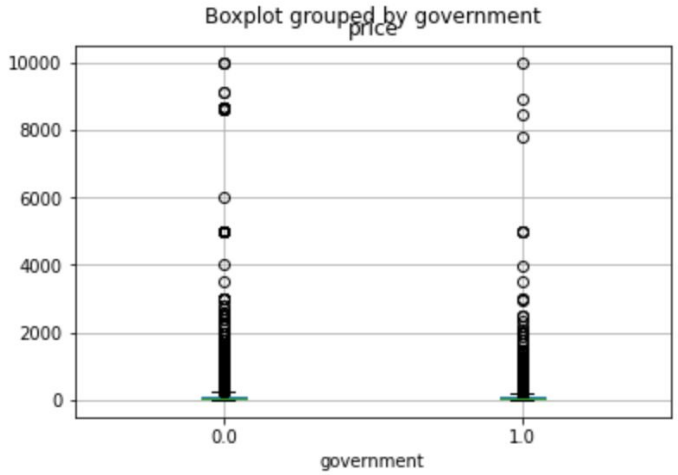
Annex 94: Boxplot for Price and Facebook



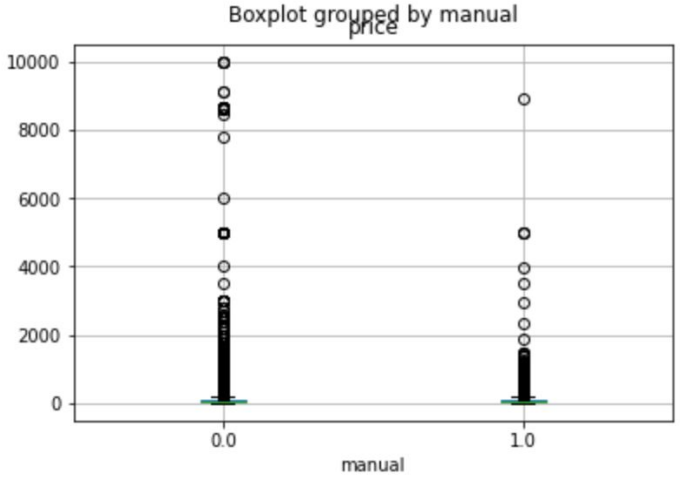
Annex 95: Boxplot for Price and Google



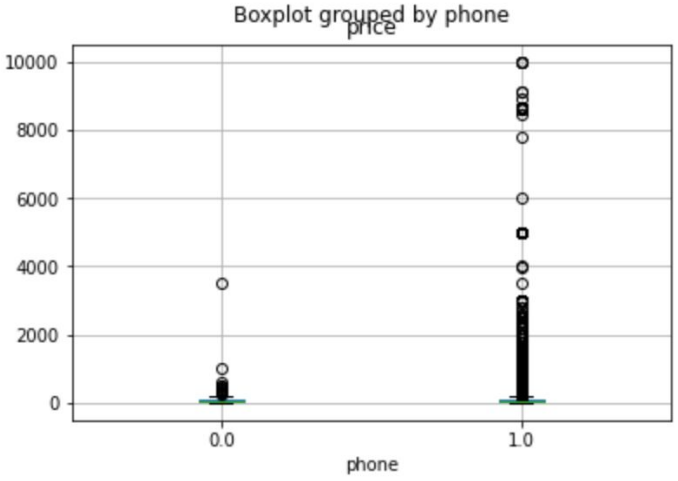
Annex 96: Boxplot for Price and Government



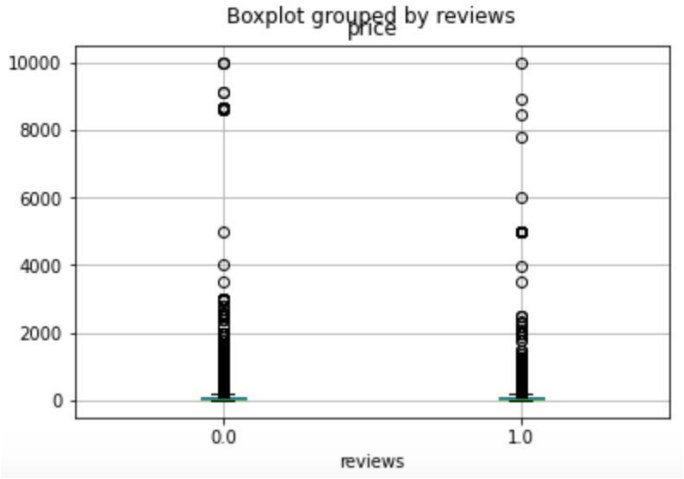
Annex 97: Boxplot for Price and Manual



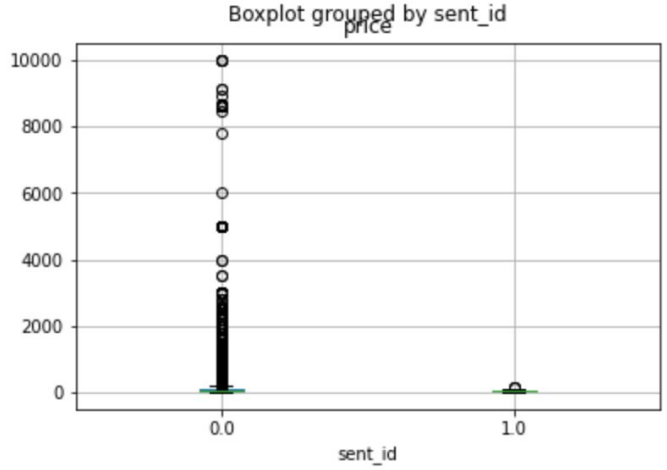
Annex 98: Boxplot for Price and Phone



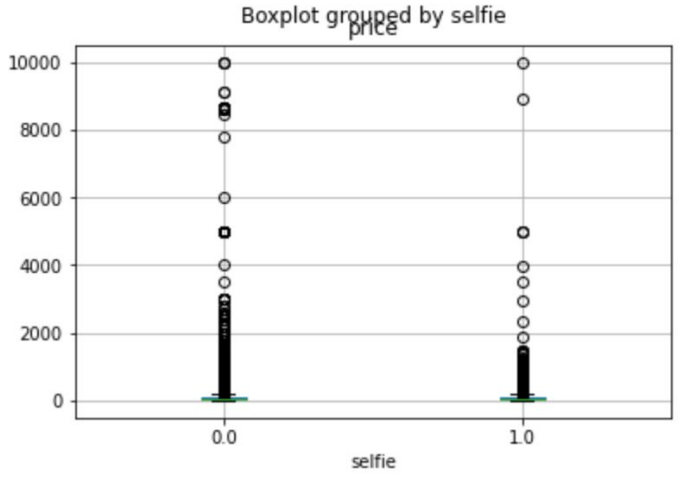
Annex 99: Boxplot for Price and Reviews



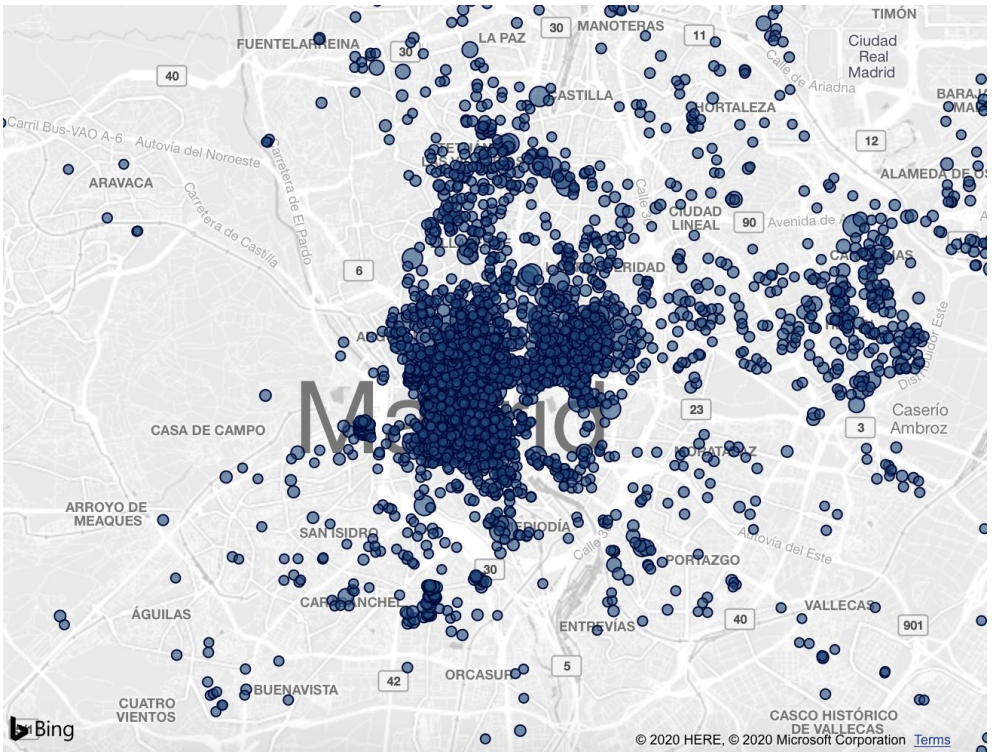
Annex 100: Boxplot for Price and Sent-id



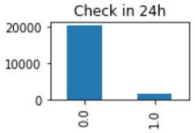
Annex 101: Boxplot for Price and Selfie

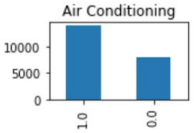


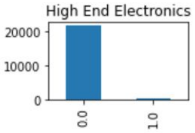
Annex 102: Latitude and Longitude Map

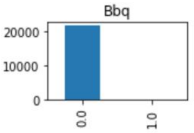


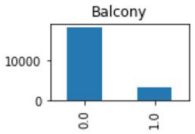
Annex 103: Value Counts for Amenities

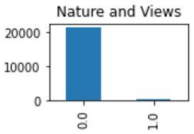


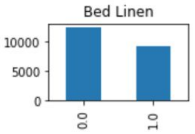


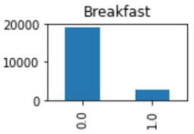


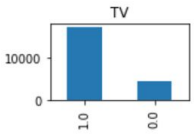


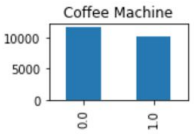


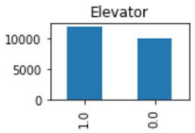


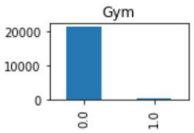


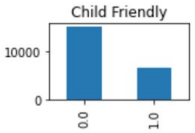


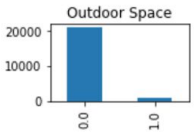


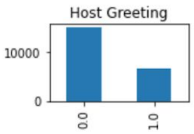


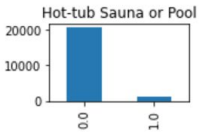


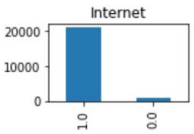


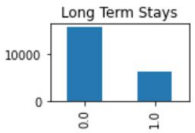


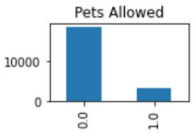


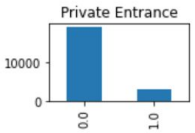


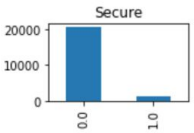


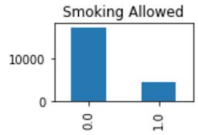


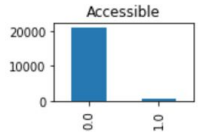


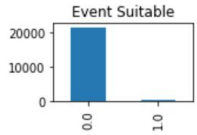




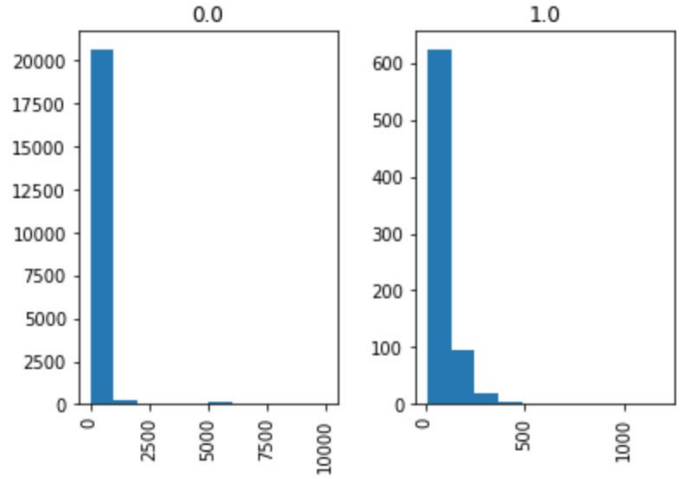




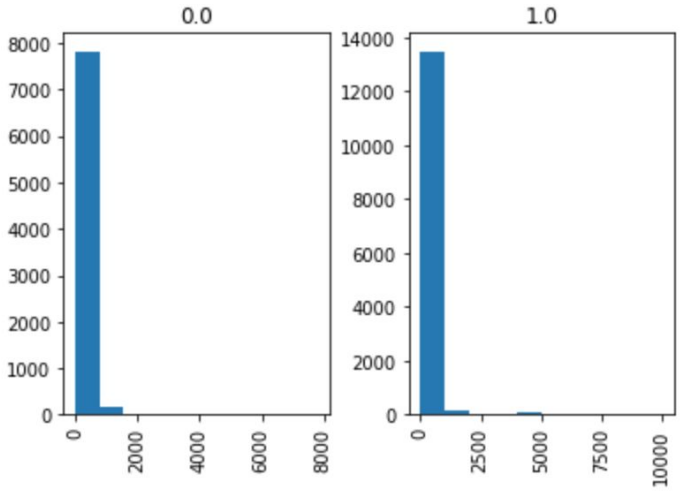




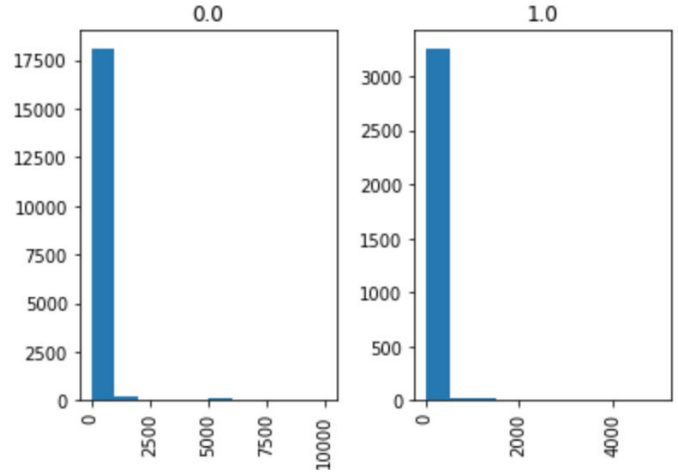
Annex 104: Histogram for Price and Accessible



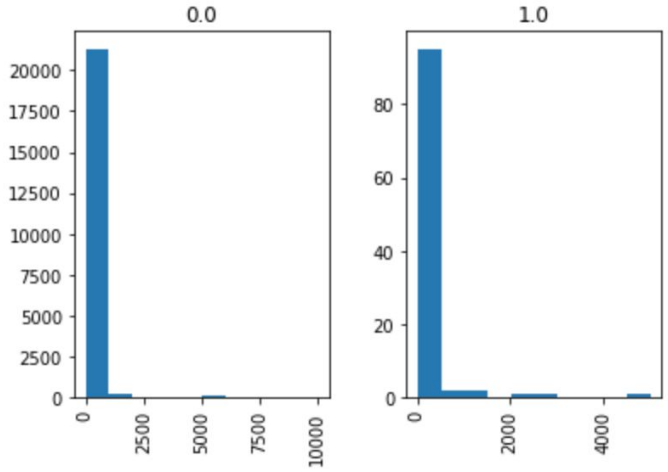
Annex 105: Histogram for Price and Air Conditioning



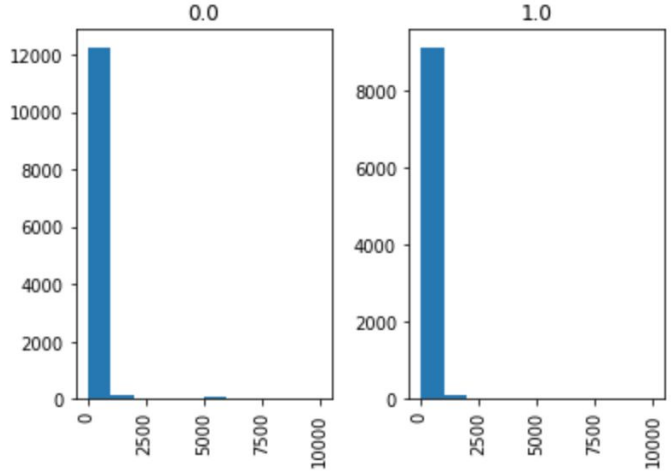
Annex 106: Histogram for Price and Balcony



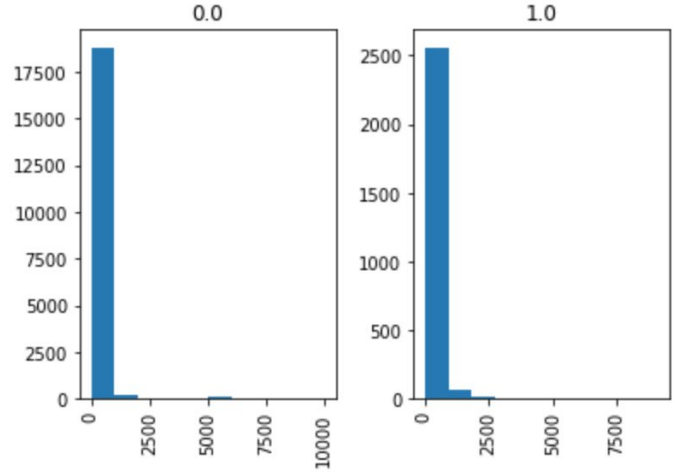
Annex 107: Histogram for Price and BBQ



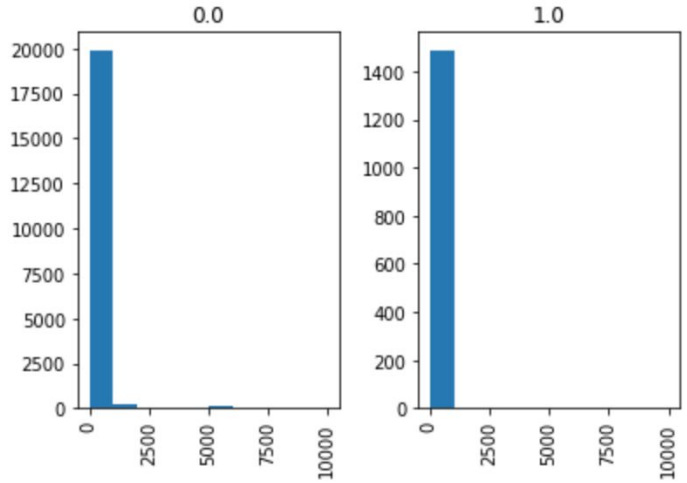
Annex 108: Histogram for Price and Bed_linen



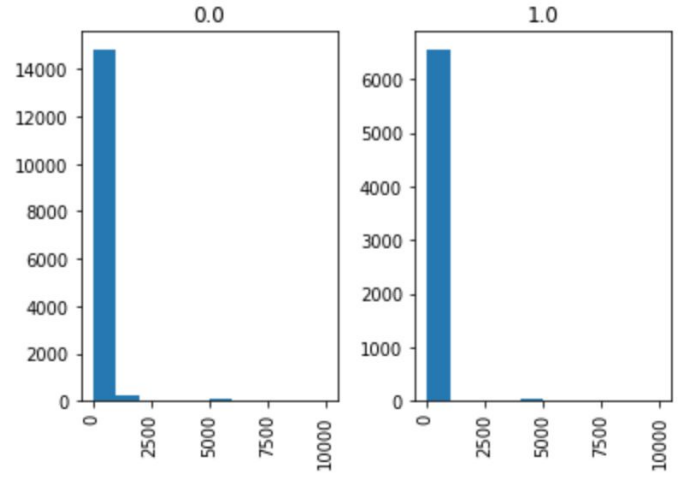
Annex 109: Histogram for Price and Breakfast



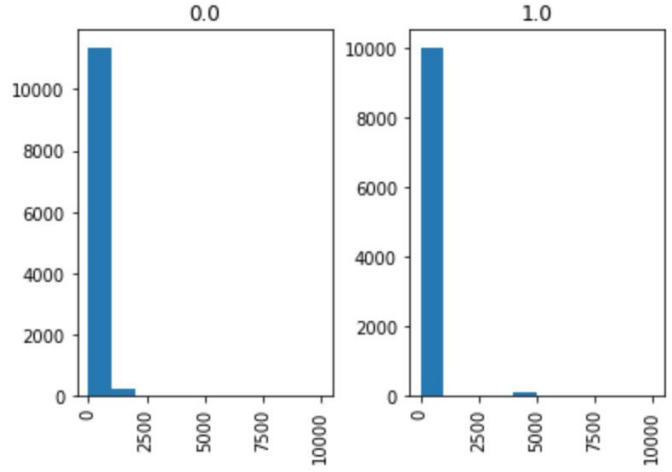
Annex 110: Histogram for Price and Check-in 24hr



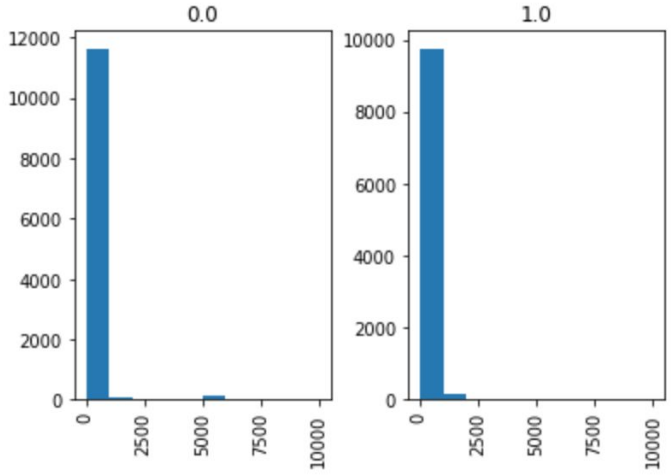
Annex 111: Histogram for Price and Child_friendly



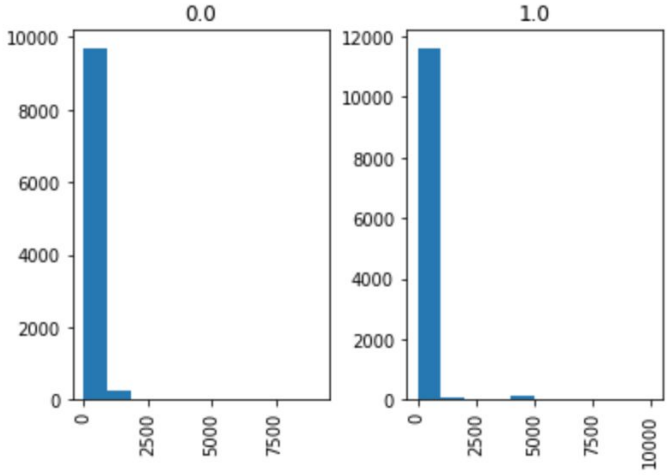
Annex 112: Histogram for Price and Child_friendly



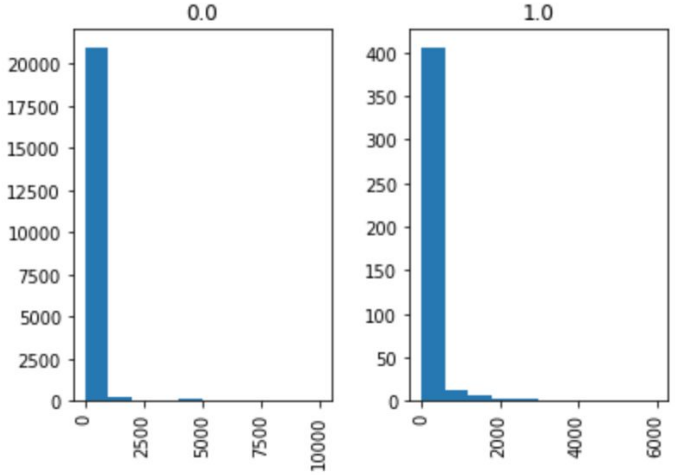
Annex 113: Histogram for Price and Cooking_basics



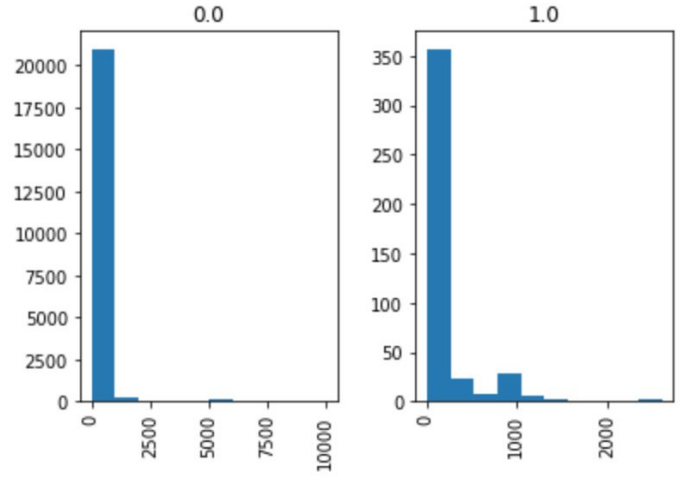
Annex 114: Histogram for Price and Elevator



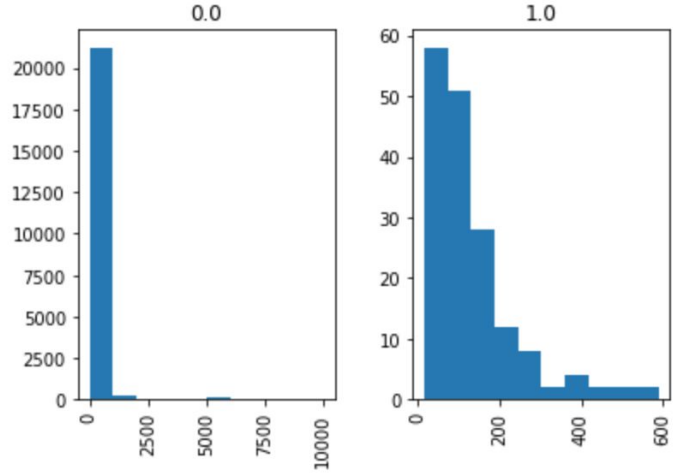
Annex 115: Histogram for Price and Event_suitable



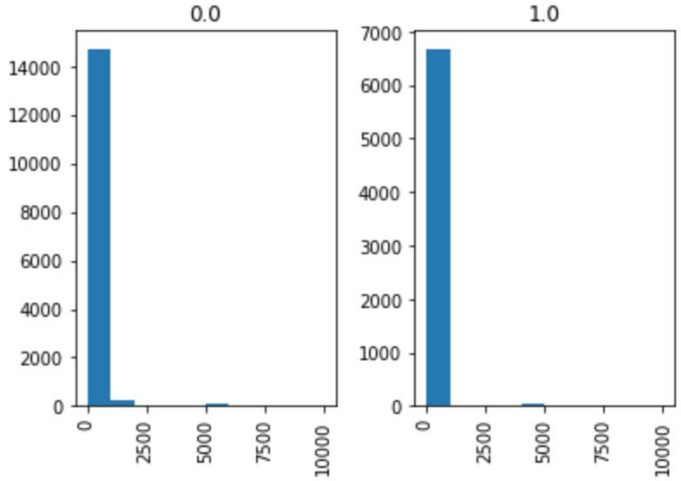
Annex 116: Histogram for Price and Gym



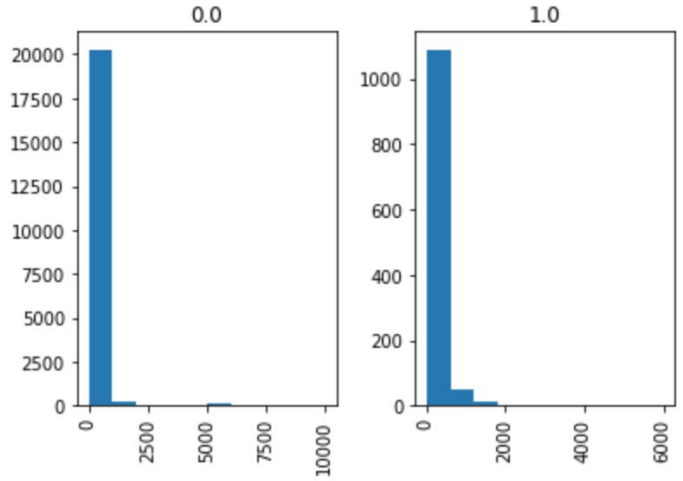
Annex 117: Histogram for Price and High_end_electronics



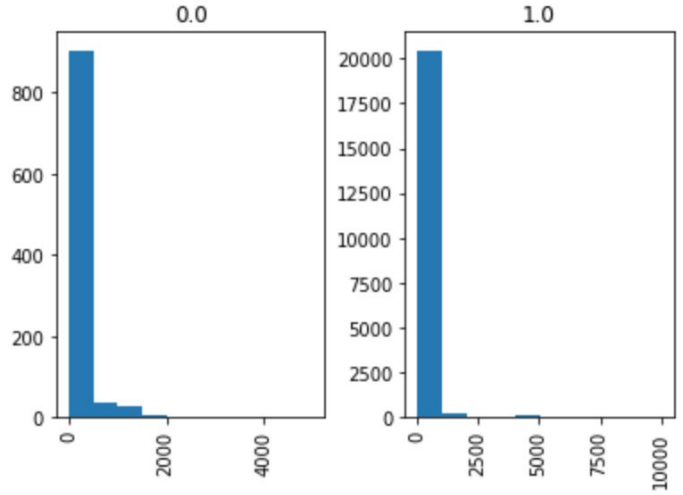
Annex 118: Histogram for Price and Host_greeting



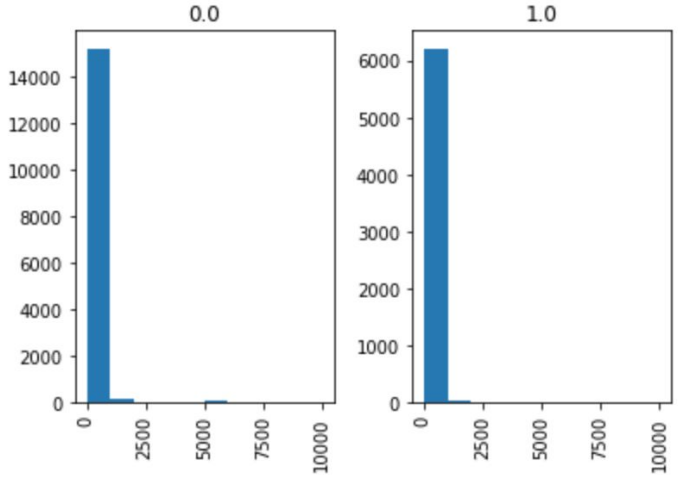
Annex 119: Histogram for Price and Hot_tub_sauna_or_pool



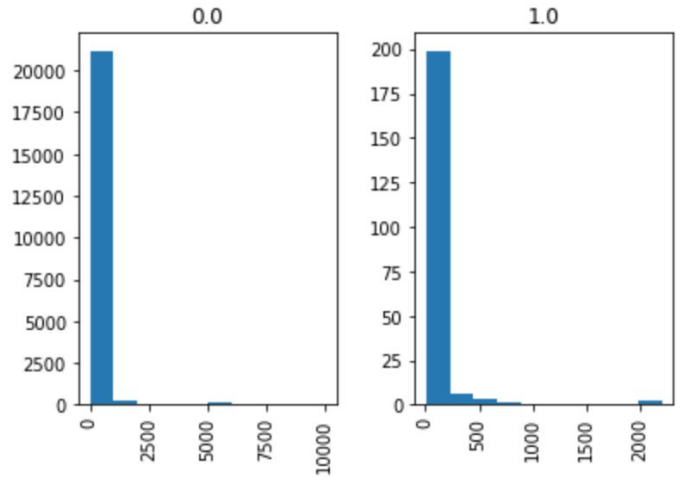
Annex 120: Histogram for Price and Internet



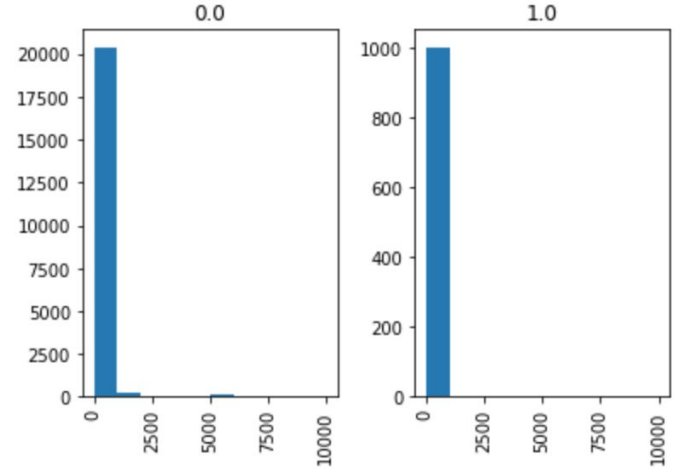
Annex 121: Histogram for Price and Long_term_stays



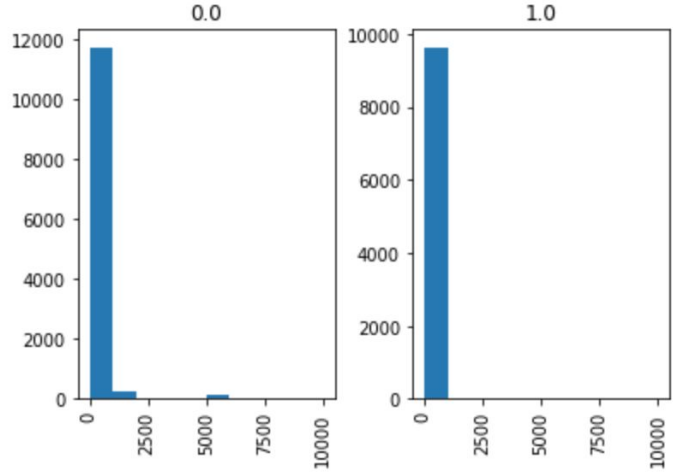
Annex 122: Histogram for Price and Nature_and_views



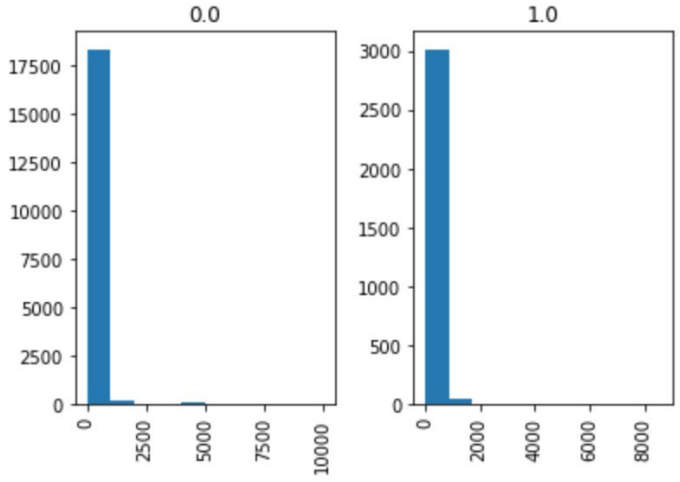
Annex 123: Histogram for Price and Outdoor_space



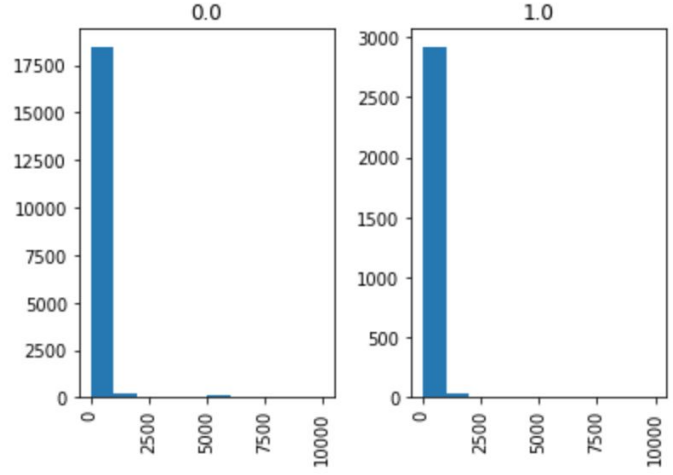
Annex 124: Histogram for Price and Parking



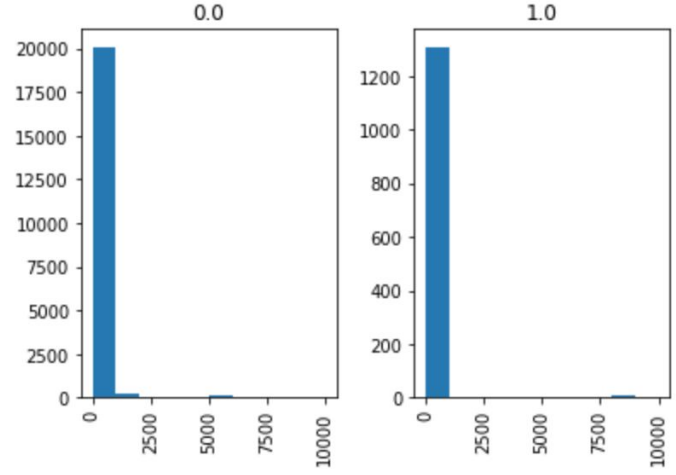
Annex 125: Histogram for Price and Pets_allowed



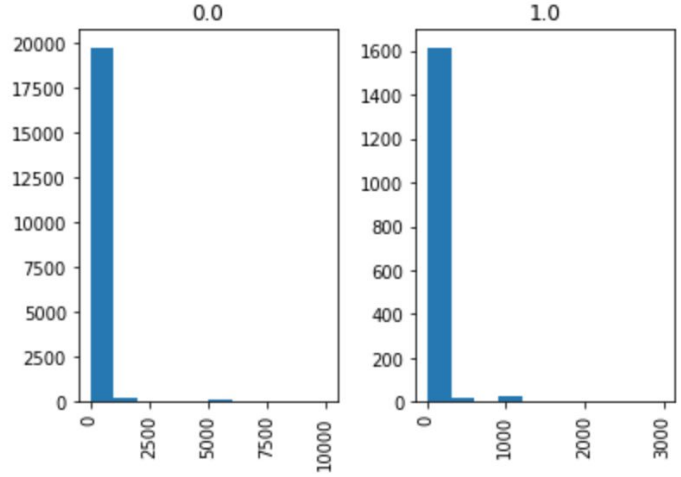
Annex 126: Histogram for Price and Private_entrance



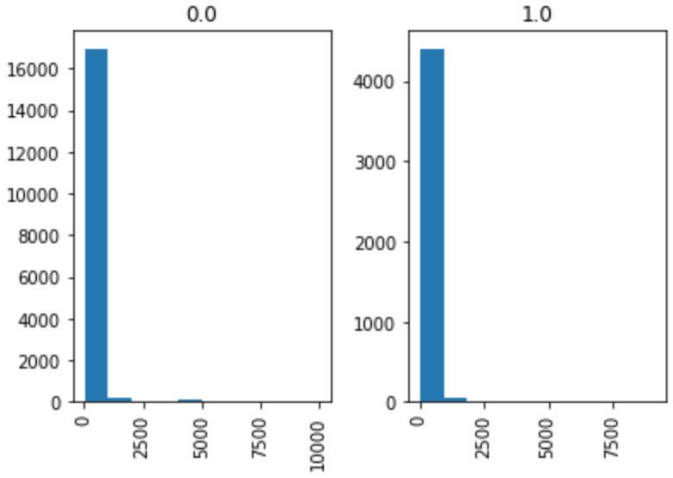
Annex 127: Histogram for Price and Secure



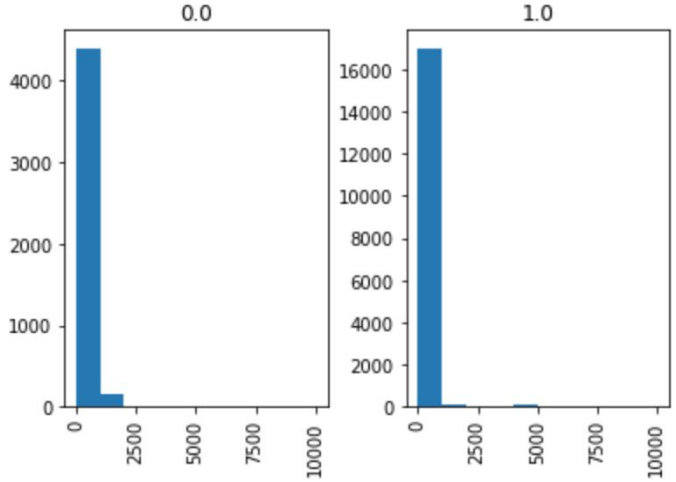
Annex 128: Histogram for Price and Self_check_in



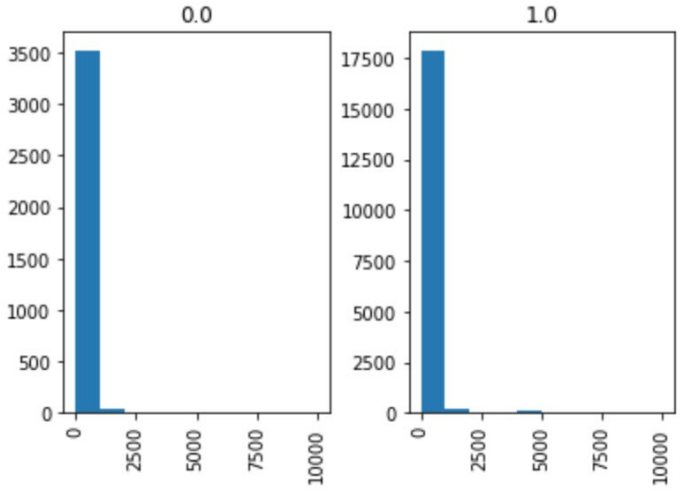
Annex 129: Histogram for Price and Smoking_allowed



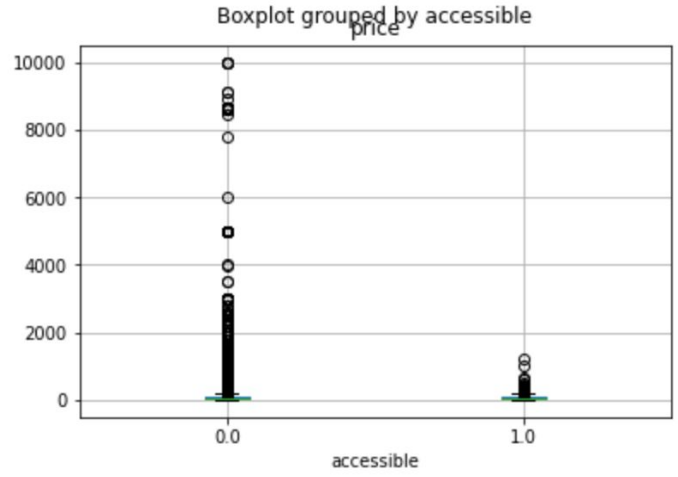
Annex 130: Histogram for Price and TV



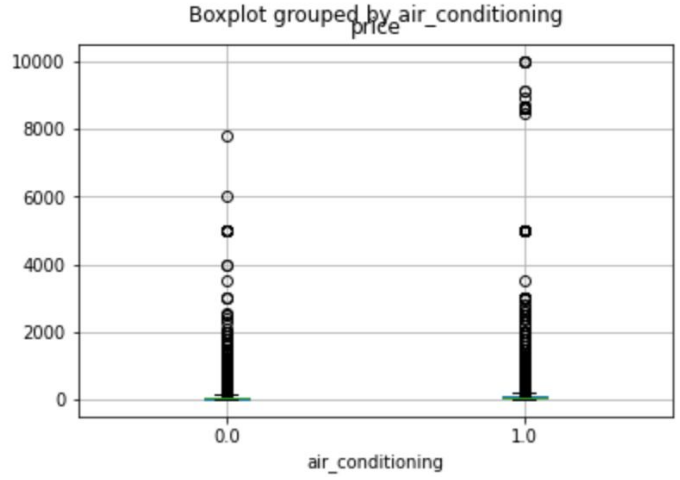
Annex 131: Histogram for Price and White_goods



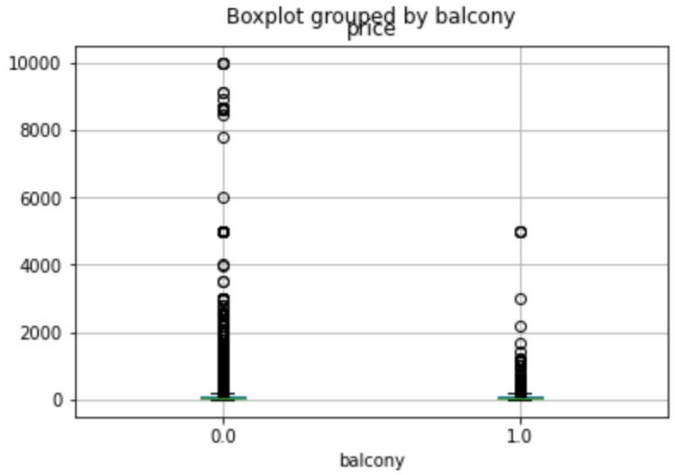
Annex 132: Boxplot for Price and Accessible



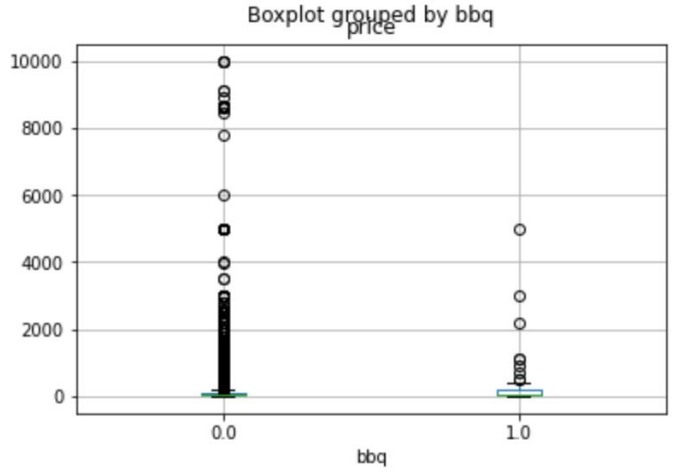
Annex 133: Boxplot for Price and Air_conditioning



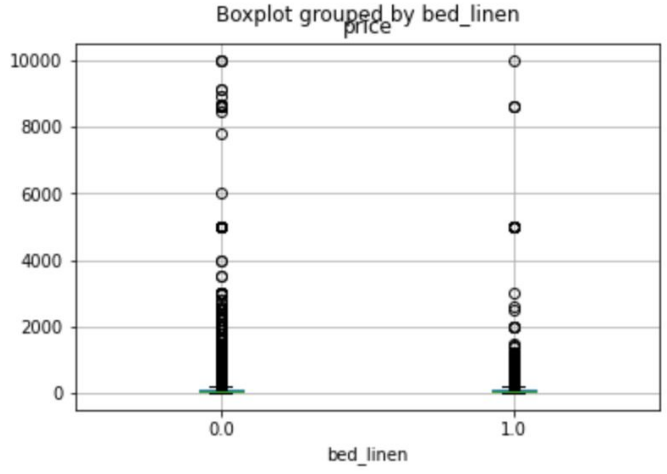
Annex 134: Boxplot for Price and Balcony



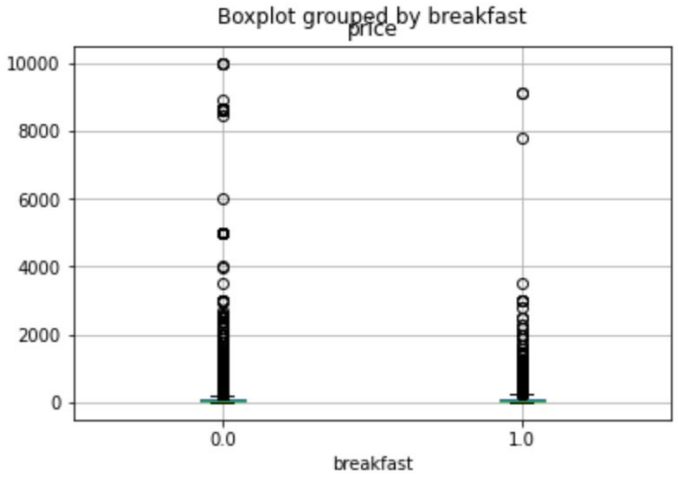
Annex 135: Boxplot for Price and BBQ



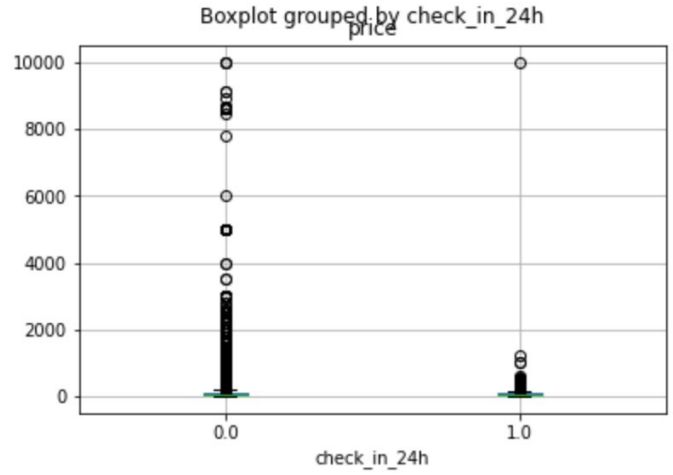
Annex 136: Boxplot for Price and Bed_linen



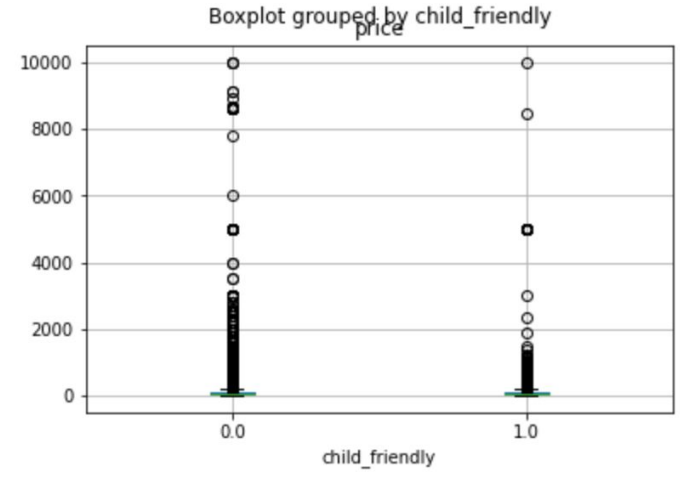
Annex 137: Boxplot for Price and Breakfast



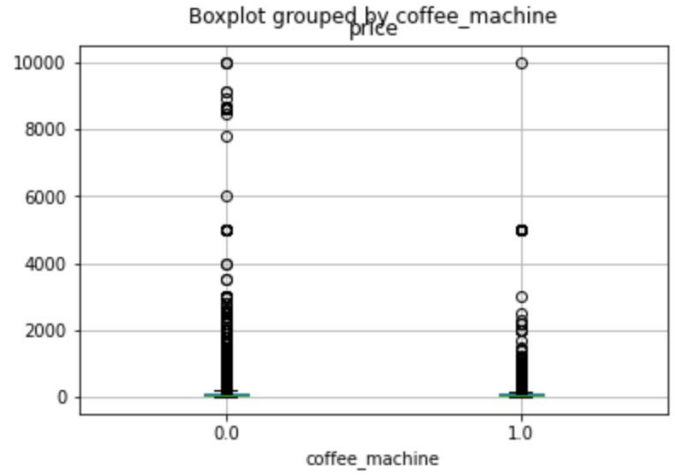
Annex 138: Boxplot for Price and Check_in_24hr



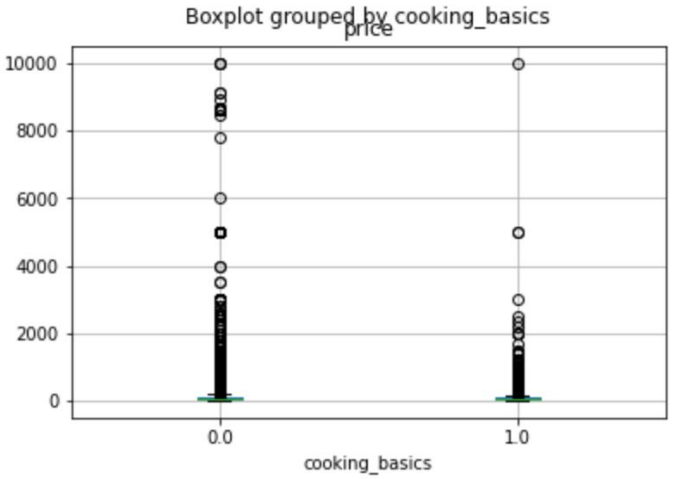
Annex 139: Boxplot for Price and Child_friendly



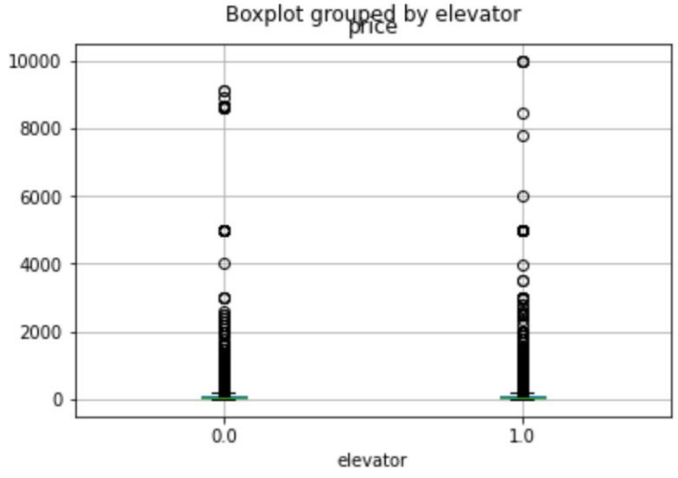
Annex 140: Boxplot for Price and Coffee_machine



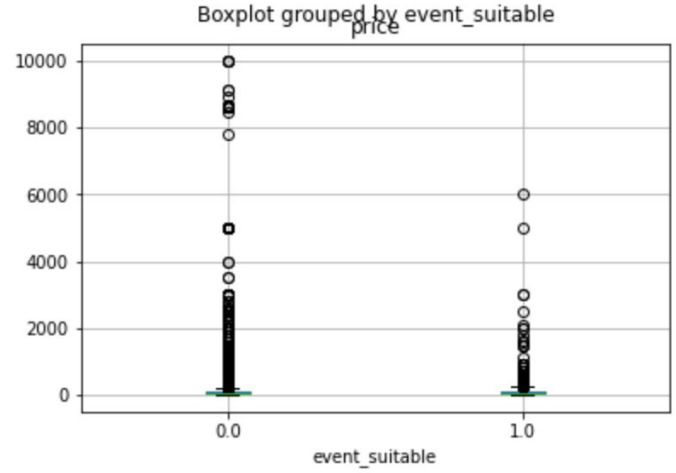
Annex 141: Boxplot for Price and Cooking_basics



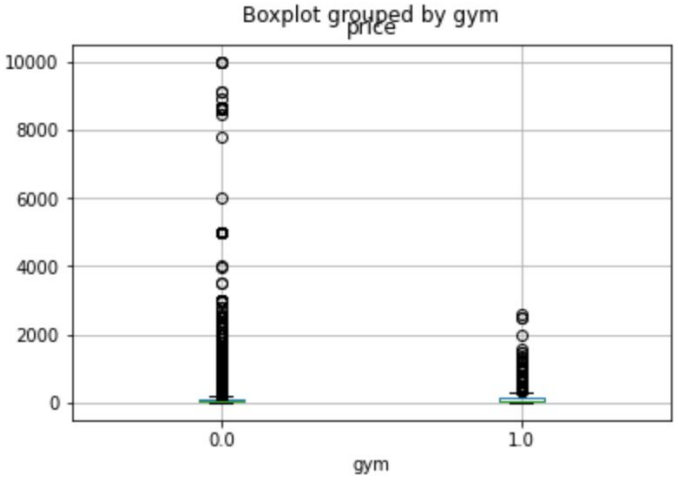
Annex 142: Boxplot for Price and Elevator



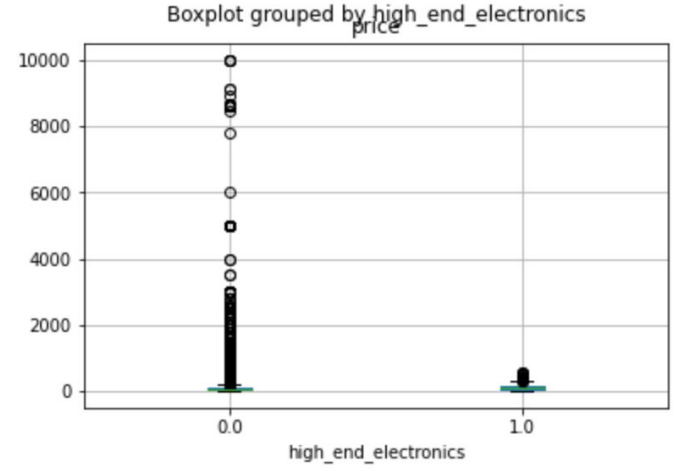
Annex 143: Boxplot for Price and Event_suitable



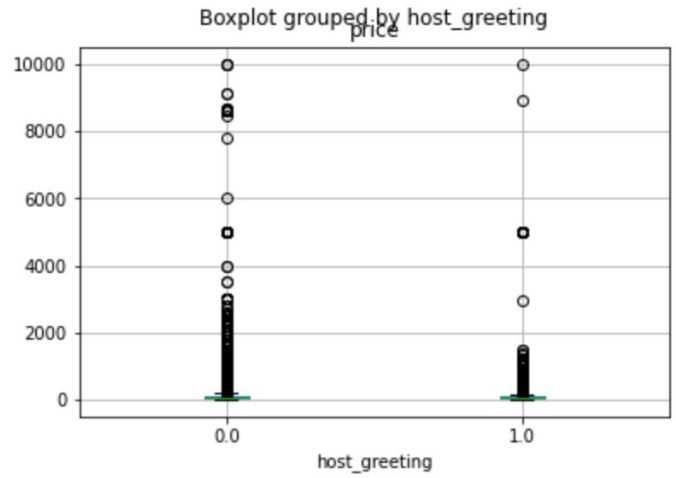
Annex 144: Boxplot for Price and Gym



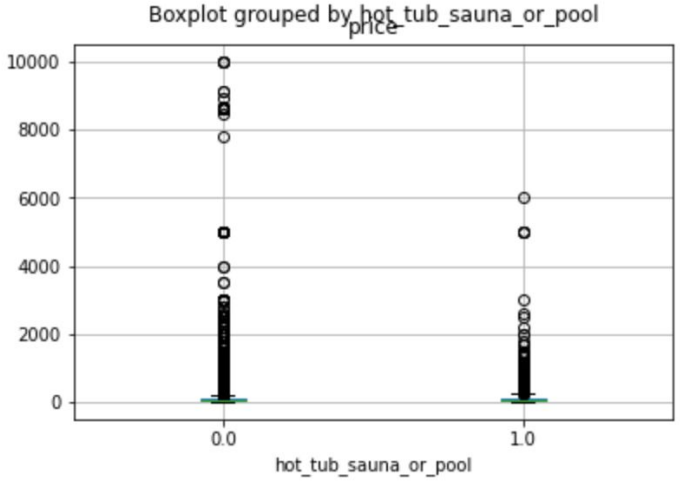
Annex 145: Boxplot for Price and High_end_electronics



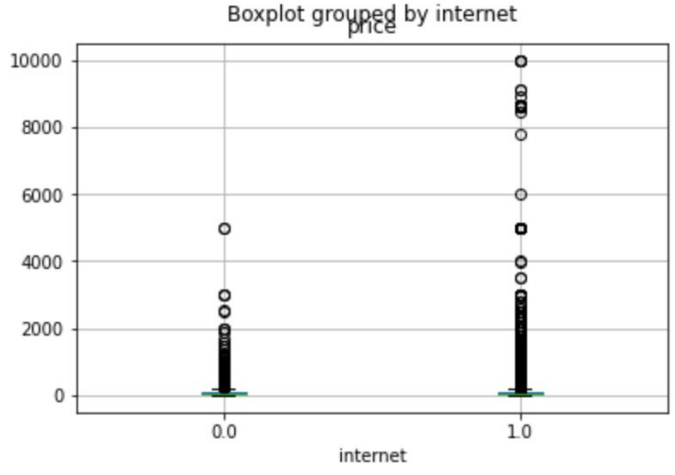
Annex 146: Boxplot for Price and Host_greeting



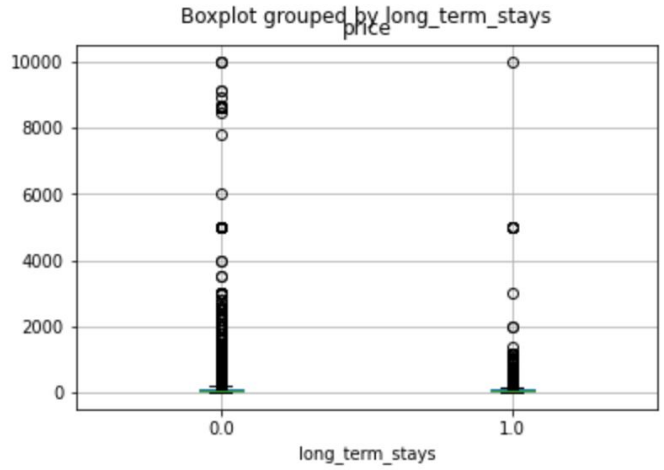
Annex 147: Boxplot for Price and Hot_tub_suana_or_pool



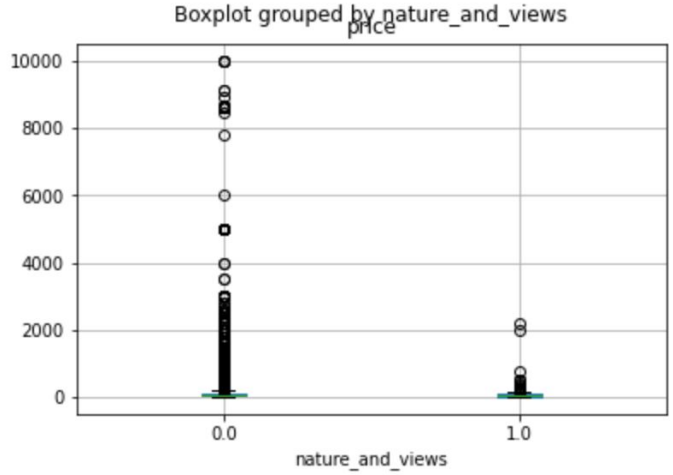
Annex 148: Boxplot for Price and Internet



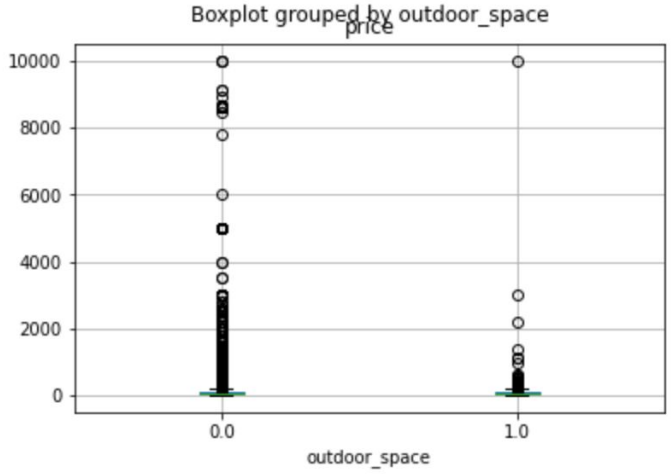
Annex 149: Boxplot for Price and Long_term_stays



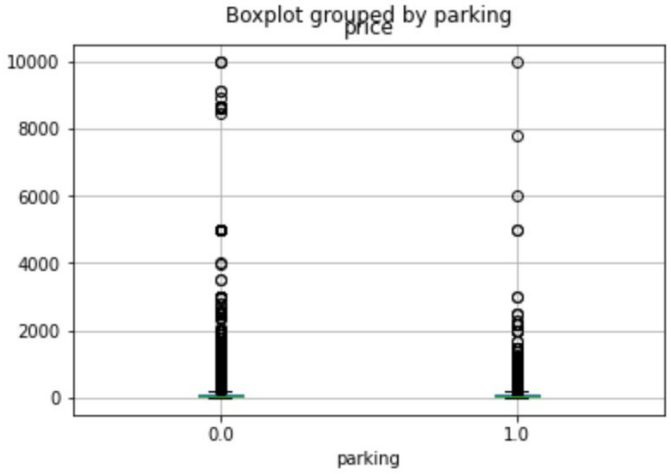
Annex 150: Boxplot for Price and Nature_and_views



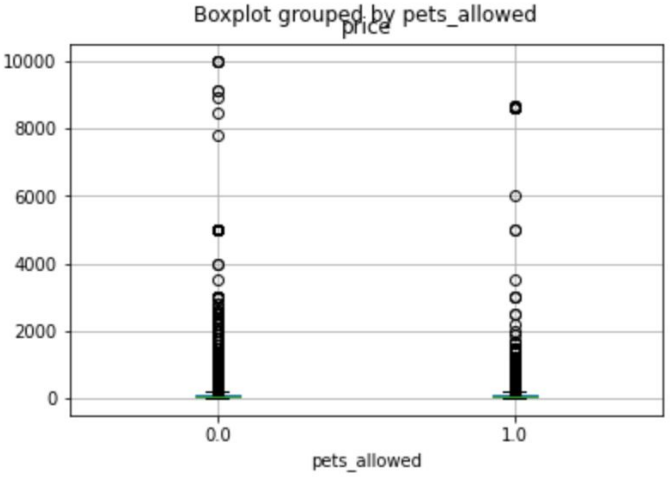
Annex 151: Boxplot for Price and Outdoor_space



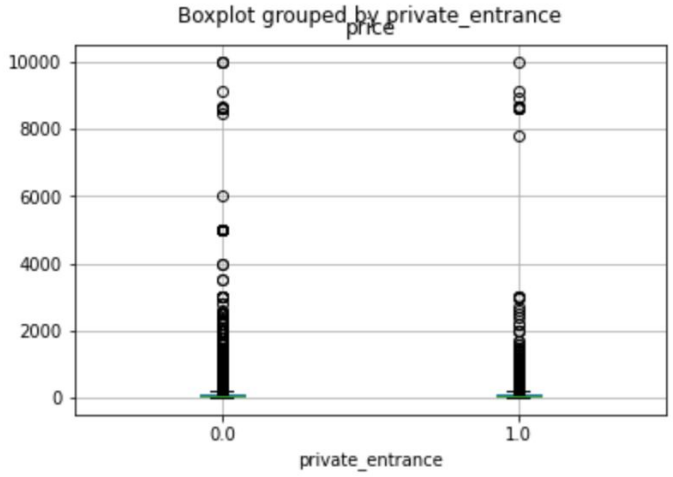
Annex 152: Boxplot for Price and Parking



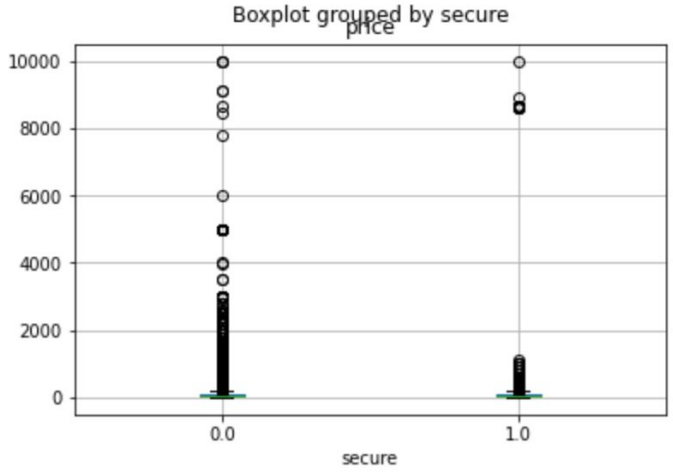
Annex 153: Boxplot for Price and Pets_allowed



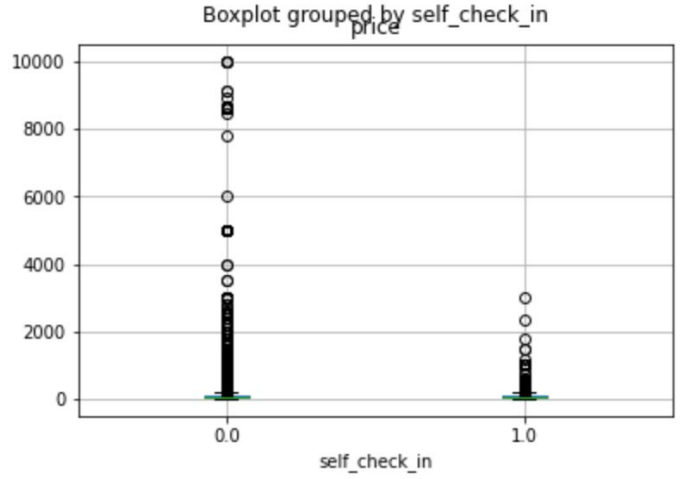
Annex 154: Boxplot for Price and Private_entrance



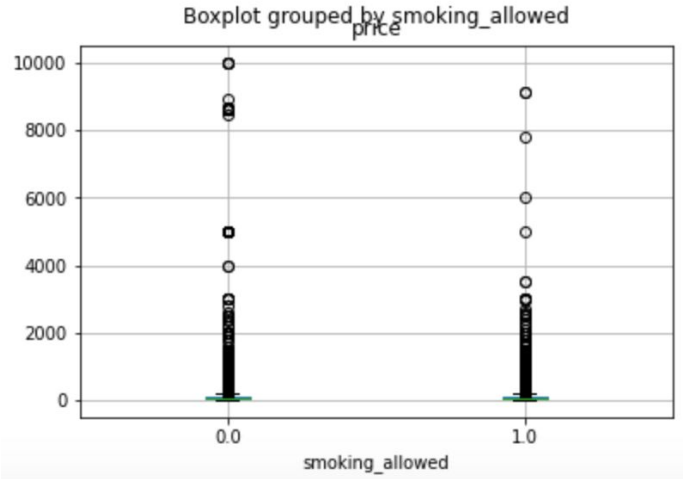
Annex 155: Boxplot for Price and Secure



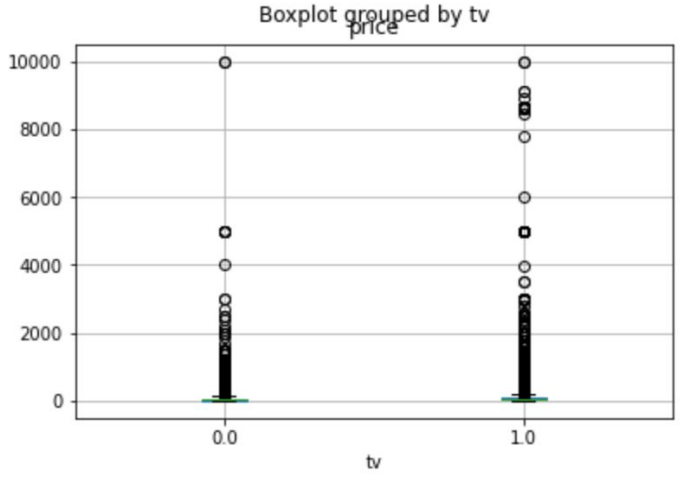
Annex 156: Boxplot for Price and Self_check_in



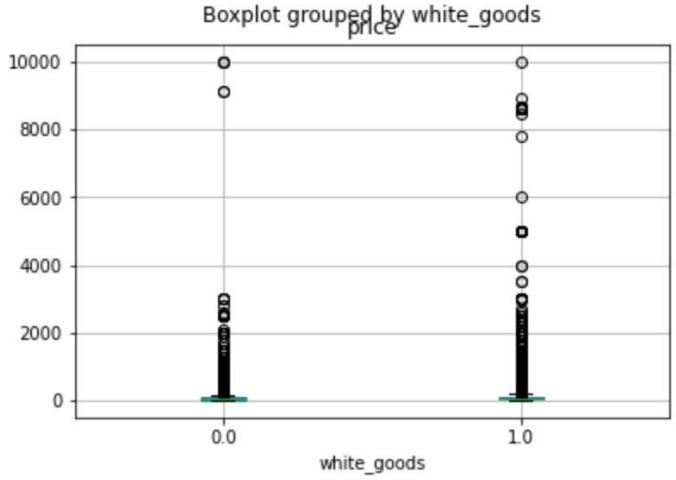
Annex 157: Boxplot for Price and Smoking_allowed



Annex 158: Boxplot for Price and TV



Annex 159: Boxplot for Price and White_goods



Annex 160: KBest: Chi-Square Feature Selection for Area_name

	Specs	Score
25	area_name_san_blas	2078.425402
24	area_name_salamanca	1185.200440
6	area_name_city_center	1157.898224
16	area_name_madrid	968.042695
31	area_name_villaverde	877.061216
22	area_name_puente_de_vallecas	780.090115
23	area_name_retiro	769.099136
14	area_name_latina	763.424443
28	area_name_usera	762.603878
7	area_name_ciudad_lineal	725.269925

Annex 161: KBest: Chi-Square Feature Selection for Amenities

	Specs	Score
13	child_friendly	1683.564792
1	air_conditioning	1558.077619
2	high_end_electronics	1519.640748
9	coffee_machine	1400.518232
10	cooking_basics	1174.781801
17	long_term_stays	1095.461281
7	breakfast	931.420967
20	smoking_allowed	927.173635
4	balcony	876.609404
0	check_in_24h	854.063123

Annex 162: KBest: Chi-Square Feature Selection for Host_verifications

	Specs	Score
4	manual	844.482210
7	selfie	798.648772
6	reviews	726.845081
1	facebook	591.883187

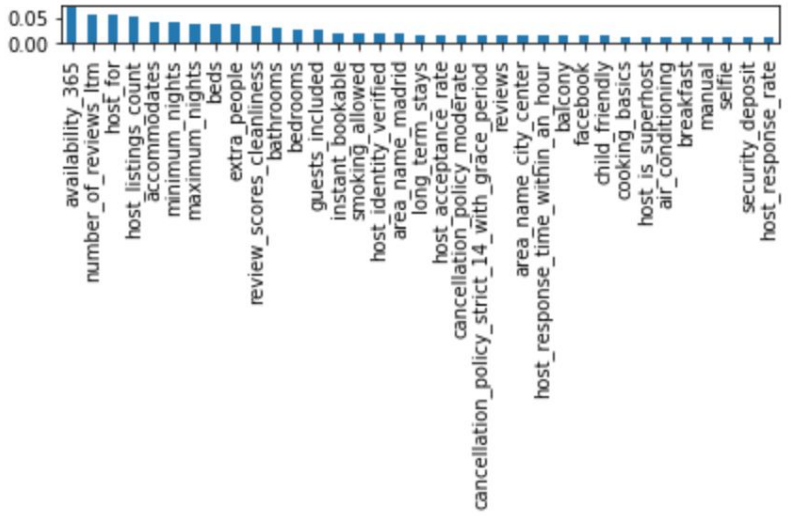
Annex 163: KBest: Chi-Square Feature Selection

	Specs	Score
15	maximum_nights	3.914360e+10
3	host_listings_count	6.947369e+05
22	host_for	5.856590e+05
16	availability_365	1.664129e+05
13	extra_people	5.482247e+04
17	number_of_reviews_ltm	5.333387e+04
14	minimum_nights	3.353003e+04
6	accommodates	1.005550e+04
41	room_type_private_room	6.757180e+03
9	beds	5.901684e+03
12	guests_included	5.214722e+03
8	bedrooms	3.015230e+03
42	room_type_shared_room	2.734211e+03
10	security_deposit	2.200268e+03
49	cancellation_policy_super_strict_30	2.088087e+03
58	area_name_san_blas	2.078425e+03
40	room_type_hotel_room	1.825135e+03
30	child_friendly	1.683565e+03
24	air_conditioning	1.558078e+03
25	high_end_electronics	1.519641e+03
0	host_response_rate	1.425197e+03
28	coffee_machine	1.400518e+03
11	cleaning_fee	1.196223e+03
57	area_name_salamanca	1.185200e+03
29	cooking_basics	1.174782e+03
51	area_name_city_center	1.157898e+03
31	long_term_stays	1.095461e+03
37	host_response_time_within_a_day	1.069097e+03
7	bathrooms	1.050340e+03
1	host_acceptance_rate	1.049928e+03

Annex 164: ExtraTreesClassifier

availability_365	0.068624
number_of_reviews_ltm	0.055729
host_for	0.055694
host_listings_count	0.051927
accommodates	0.042289
minimum_nights	0.040749
maximum_nights	0.039263
beds	0.039209
extra_people	0.036747
review_scores_cleanliness	0.032397
bathrooms	0.029023
bedrooms	0.028447
guests_included	0.025053
instant_bookable	0.019335
smoking_allowed	0.018810
host_identity_verified	0.018544
area_name_madrid	0.018432
long_term_stays	0.016918
host_acceptance_rate	0.016731
cancellation_policy_moderate	0.016666
cancellation_policy_strict_14_with_grace_period	0.016283
reviews	0.016198
area_name_city_center	0.015619
host_response_time_within_an_hour	0.014891
balcony	0.014847
facebook	0.014772
child_friendly	0.014762
cooking_basics	0.013996
host_is_superhost	0.013939
air_conditioning	0.013832
breakfast	0.013795
manual	0.013337
selfie	0.013274
security_deposit	0.012981
host_response_rate	0.012912

Annex 165: ExtraTreesClassifier Graph



Annex 166: Final Features

```
17       number_of_reviews_ltm
15              maximum_nights
13                extra_people
12              guests_included
9                           beds
16            availability_365
8                     bedrooms
27                    host_for
3           host_listings_count
6                  accommodates
14               minimum_nights
7                     bathrooms
61                       secure
11                cleaning_fee
67      room_type_private_room
1         host_acceptance_rate
50                     elevator
49                  white_goods
0            host_response_rate
10             security_deposit
31                   government
23          review_scores_value
101         area_name_salamanca
102          area_name_san_blas
```

Annex 167: LightGBM Preliminary Parameters

```
Params:
  objective: regression
  metric: mape
  verbosity: -1
  boosting_type: gbdt
  lambda_l1: 8.37345418497093
  lambda_l2: 0.0054026995561138776
  num_leaves: 4
  feature_fraction: 0.4
  bagging_fraction: 0.9274928940073707
  bagging_freq: 6
  min_child_samples: 100
```

Annex 168: Random Forest: Best Estimator

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=11, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=4,
                      min_samples_split=4, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

Annex 169: XGBoost: Best Estimator RandomizedSearchCV

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=15,
             min_child_weight=3, missing=nan, monotone_constraints=None,
             n_estimators=8, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

Annex 170: XGBoost: Best Estimator GridSearchCV

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=12,
             min_child_weight=18, missing=nan, monotone_constraints=None,
             n_estimators=20, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

Annex 171: SVR: Best Estimator

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Annex 172: Voting Ensemble: SVR and XGBoost

```
VotingRegressor(estimators=[('svr',
                             SVR(C=1.5, cache_size=200, coef0=0.0, degree=3,
                                 epsilon=0.1, gamma='scale', kernel='poly',
                                 max_iter=-1, shrinking=True, tol=0.001,
                                 verbose=False)),
                            ('xgboost',
                             XGBRegressor(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1,
                                          colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          gpu_id=-1, importance_type='gain',
                                          interaction_constraints...
                                          beta_2=0.999, early_stopping=False,
                                          epsilon=1e-08,
                                          hidden_layer_sizes=(16, 10, 16),
                                          learning_rate='constant',
                                          learning_rate_init=0.001,
                                          max_fun=15000, max_iter=200,
                                          momentum=0.9, n_iter_no_change=10,
                                          nesterovs_momentum=True, power_t=0.5,
                                          random_state=0, shuffle=True,
                                          solver='adam', tol=0.0001,
                                          validation_fraction=0.1,
                                          verbose=False, warm_start=False))],
                 n_jobs=None, weights=None)
```

Annex 173: Voting Ensemble: SVR and MLP

```
VotingRegressor(estimators=[('svr',
                             SVR(C=1.5, cache_size=200, coef0=0.0, degree=3,
                                 epsilon=0.1, gamma='scale', kernel='poly',
                                 max_iter=-1, shrinking=True, tol=0.001,
                                 verbose=False)),
                            ('MLP',
                             MLPRegressor(activation='tanh', alpha=0.0001,
                                          batch_size='auto', beta_1=0.9,
                                          beta_2=0.999, early_stopping=False,
                                          epsilon=1e-08,
                                          hidden_layer_sizes=(16, 10, 16),
                                          learning_rate='constant',
                                          learning_rate_init=0.001,
                                          max_fun=15000, max_iter=200,
                                          momentum=0.9, n_iter_no_change=10,
                                          nesterovs_momentum=True, power_t=0.5,
                                          random_state=0, shuffle=True,
                                          solver='adam', tol=0.0001,
                                          validation_fraction=0.1,
                                          verbose=False, warm_start=False))],
                 n_jobs=None, weights=None)
```

Annex 174: Bagging Regressor: SVR

```
BaggingRegressor(base_estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                                    epsilon=0.1, gamma='scale', kernel='rbf',
                                    max_iter=-1, shrinking=True, tol=0.001,
                                    verbose=False),
                 bootstrap=True, bootstrap_features=False, max_features=1.0,
                 max_samples=1.0, n_estimators=5, n_jobs=None, oob_score=False,
                 random_state=None, verbose=0, warm_start=False)
```

Annex 175: Stacking Regressor

```
StackingRegressor(meta_regressor=SVR(C=1.5, cache_size=200, coef0=0.0, degree=3,
                                     epsilon=0.1, gamma='scale', kernel='poly',
                                     max_iter=-1, shrinking=True, tol=0.001,
                                     verbose=False),
                  refit=True,
                  regressors=[XGBRegressor(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           gpu_id=-1, importance_type='gain',
                                           interaction_co...
                                           hidden_layer_sizes=(16, 10, 16),
                                           learning_rate='constant',
                                           learning_rate_init=0.001,
                                           max_fun=15000, max_iter=200,
                                           momentum=0.9, n_iter_no_change=10,
                                           nesterovs_momentum=True, power_t=0.5,
                                           random_state=0, shuffle=True,
                                           solver='adam', tol=0.0001,
                                           validation_fraction=0.1,
                                           verbose=False, warm_start=False)],
                  store_train_meta_features=False,
                  use_features_in_secondary=False, verbose=0)
```