Airbnb Listings in European Cities

Negru Bogdan, Constantinescu Andrei-Eduard, Bădescu Mădălina,
Sabo Vlad-Andrei, Drobnitchi Daniel, Suto Robert-Lucian (411)

May, 2023

1 Introduction

The rapid growth of the economy has revolutionized the travel and accommodation industry, with Airbnb emerging as a prominent player. As an online platform that facilitates short-term rentals, Airbnb has gained popularity worldwide, offering a wide range of accommodation options for travelers. In recent years, European cities have witnessed a significant surge in Airbnb listings, prompting the need to explore and understand the dynamics of pricing in this context.

This project aims to provide a comprehensive analysis of Airbnb listings in some of the most popular European cities. By examining the determinants that influence pricing, we seek to unravel the complexities behind the variations in rates across different locations. Understanding these factors can shed light on the broader implications for travelers, hosts, and policymakers alike.

To accomplish this, we are working with a comprehensive dataset that encompasses a diverse range of attributes associated with Airbnb listings. These attributes include room types, cleanliness and satisfaction ratings, number of bedrooms, distance from the city center, and more. By leveraging spatial econometric methods, we will delve into the dataset to identify the key determinants that contribute to variations in Airbnb prices across the selected European cities.

2 Description of the Dataset

In this dataset [4], a comprehensive examination of Airbnb listings in some of the most popular European cities is presented. The information is gathered for the research paper "Determinants of Airbnb prices in European cities: A spatial econometrics approach"[3]. Each listing undergoes an assessment of various characteristics, including room types, cleanliness and satisfaction ratings, number of bedrooms, distance from the city center, and more. These factors are presented in detail so that it is possible to gain a profound understanding of Airbnb prices.

The data is collected from European cities and, in order to be used for some generalization purposes, it represents a large part of the continent. There are 10 cities contained in the data set, with information collected from Airbnbs for weekdays and weekends:

- Amsterdam
- Athens
- Barcelona
- Berlin
- Budapest
- Lisbon
- London
- Paris
- Rome
- Vienna

The data is separated between weekdays and weekends. A complete list of the 19 factors taken into consideration in the data set, for all cities and both parts of the week, can be seen in Table 1.

$attr\_index_j$ represents the attraction index for listing k, based on K points of interest, and is computed with the formula

$$attr\_index_j = \sum_{k=1}^{K} \frac{R_k}{d_{jk}}$$

Table 1: Description of the Columns from the dataset

Column Name	**Description**
realSum	The total price of the Airbnb listing. (Numeric)
room_type	The type of room being offered (e.g., private, shared, etc.). (Categorical)
room_shared	Whether the room is shared or not. (Boolean)
room_private	Whether the room is private or not. (Boolean)
person_capacity	The maximum number of people that can stay in the room. (Numeric)
host_is_superhost	Whether the host is a superhost or not. (Boolean)
multi	Whether the listing is for multiple rooms or not. (Boolean)
biz	Whether the listing is for business purposes or not. (Boolean)
cleanliness_rating	The cleanliness rating of the listing. (Numeric)
guest_satisfaction_overall	The overall guest satisfaction rating of the listing. (Numeric)
bedrooms	The number of bedrooms in the listing. (Numeric)
dist	The distance from the city centre. (Numeric)
metro_dist	The distance from the nearest metro station. (Numeric)
lng	The longitude of the listing. (Numeric)
lat	The latitude of the listing. (Numeric)
attr_index	Attraction index of the listing location. (Numeric)
attr_index_norm	Normalized attraction index. (Numeric, between 0 - 100)
rest_index	Attraction index of the listing location. (Numeric)
rest_index_norm	Normalized attraction index. (Numeric, between 0 - 100)

where R is the number of reviews for attraction k, and d_{jk} is the distance between the listing and point k. The normalized attraction index is equal to the attraction index divided by the maximum value in a given city and multiplied by 100. The restaurant index is computed in the same manner.

The offers were collected four to six weeks in advance of the travel dates, and the collected prices refer to the full amount due for the accommodation, including the reservation fee and cleaning fee. All prices in the samples refer to accommodation for two people and data set contains only data for listings that accommodate a maximum of 6 people.

An overview of all the columns can be seen in Figure 1.

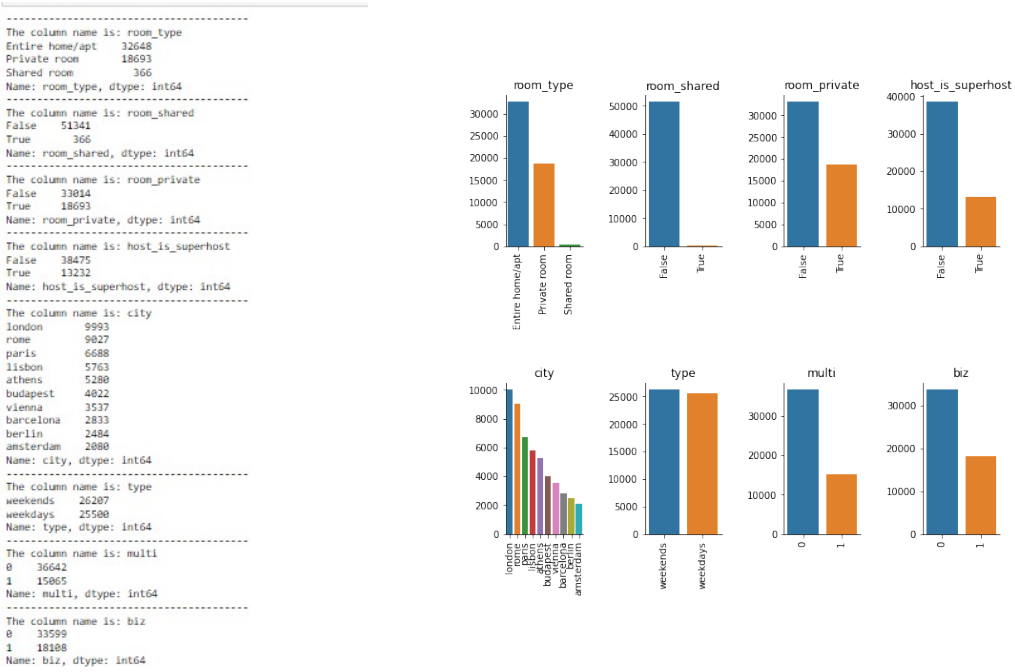

Figure 1: Data Summary

3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that involves examining and summarizing a dataset to gain insights and identify patterns, trends, and relationships within the data. It is a primarily visual and statistical approach that helps analysts understand the underlying structure of the data before formal modeling or hypothesis testing.

Big data is characterized by its high volume, velocity, and variety, making it challenging to analyze using traditional techniques. EDA provides a means to explore and make sense of vast amounts of data by leveraging visualizations, summary statistics, and useful graphics. It helps in detecting outliers, understanding data quality issues, identifying missing values, and revealing potential correlations or patterns that might not be immediately apparent, especially when working with a large quantity of information.

3.1 Distance from the center of the city

As a first impression and from the little we know about rental properties, we wanted to observe how the location of a listing affects the price, more specifically, if the properties that are closer to the center of the city are more expensive than the others. We plotted them accordingly on the map of their respective cities and in Figure 2 to Figure 11 we illustrate some of the results.

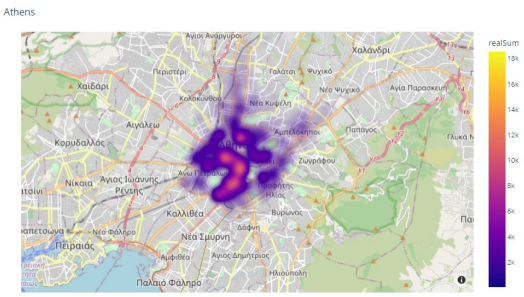

Figure 2: Athens realSum plot

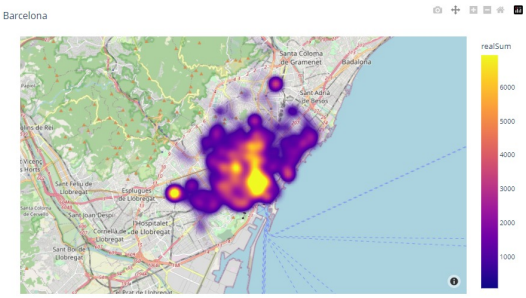

Figure 3: Barcelona realSum plot

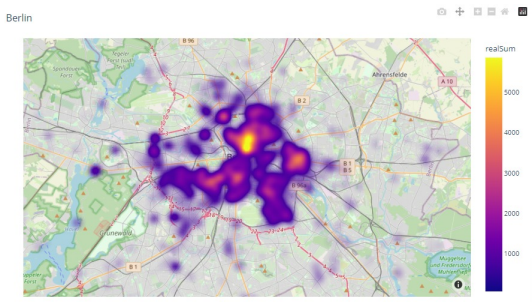

Figure 4: Berlin realSum plot

Figure 5: Budapest realSum plot

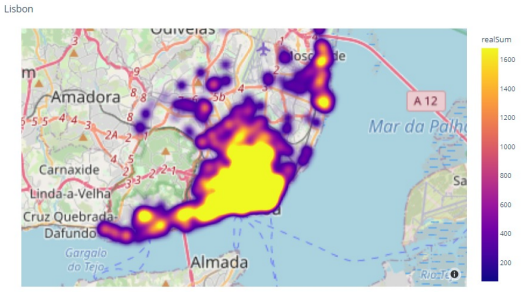

Figure 6: Lisbon realSum plot

Figure 7: London realSum plot

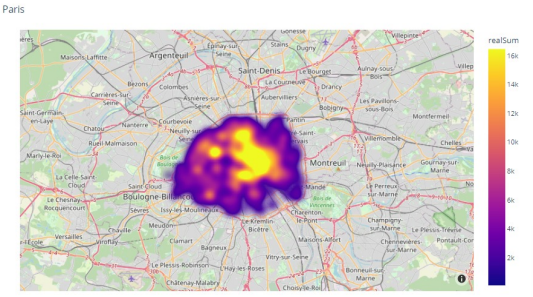

Figure 8: Paris realSum plot

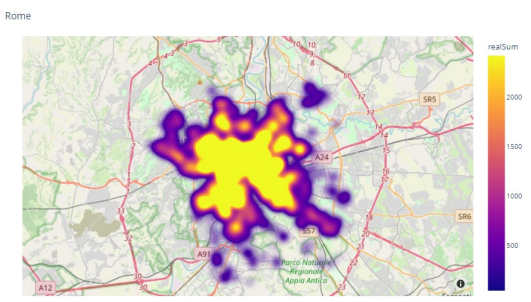

Figure 9: Rome realSum plot

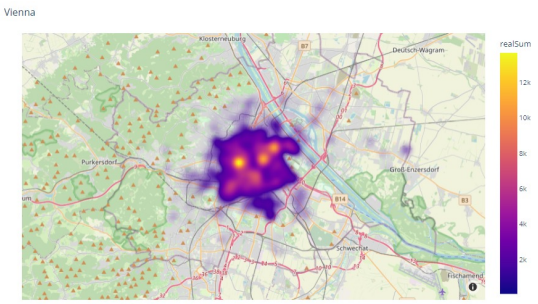

Figure 10: Vienna realSum plot

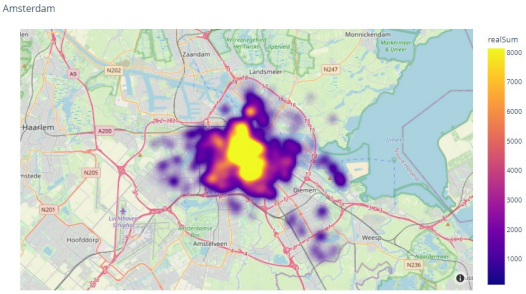

Figure 11: Amsterdam realSum plot

As we can see, it's pretty clear that the closer a property is to the center of the city, the more expensive it is. However, there are some outliers, and exceptions to this. More specifically, rentals from Lisbon and Rome seem to be expensive, regardless of the location.

3.2 Various relationships between features

Next, we wanted to observe if any other feature can influence the price of the property. As presented in Table 1, there are a bunch of features to an Airbnb rental that could influence its price, such as the type of the room, capacity, cleanliness, distance from various attractions/restaurants/metro, etc. The influences are normal, and they can be seen in other types of rental properties and also in hotels, motels, etc. Below are some plots (from Figure 12 to Figure 16) that describe these specific relationships between the features of the data set:

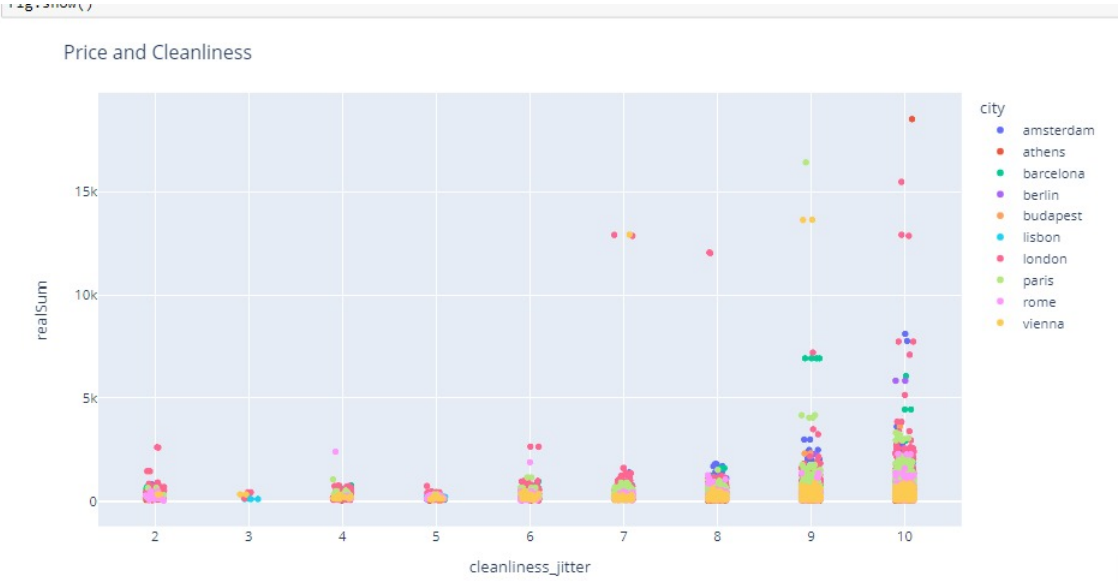

Figure 12: Cleanliness and price

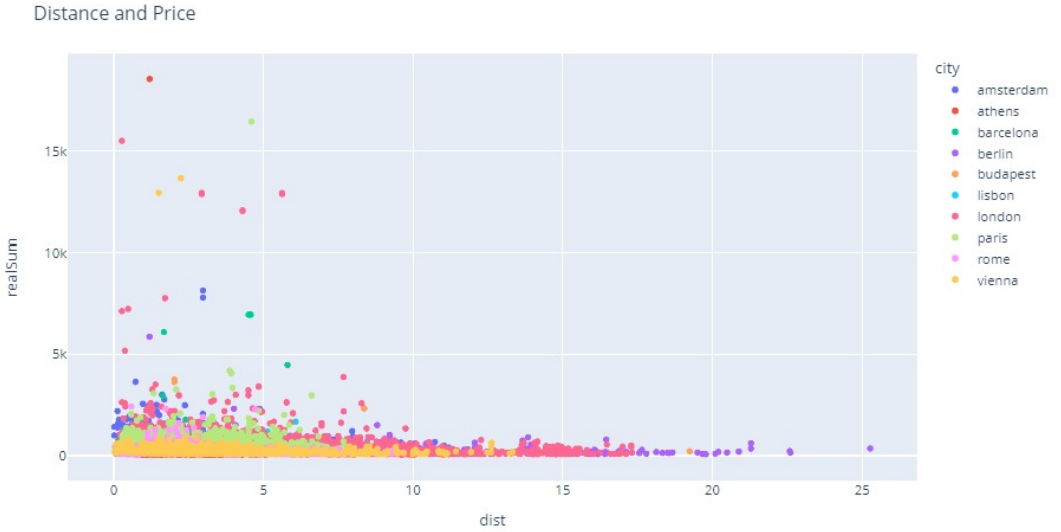

Figure 13: City and price

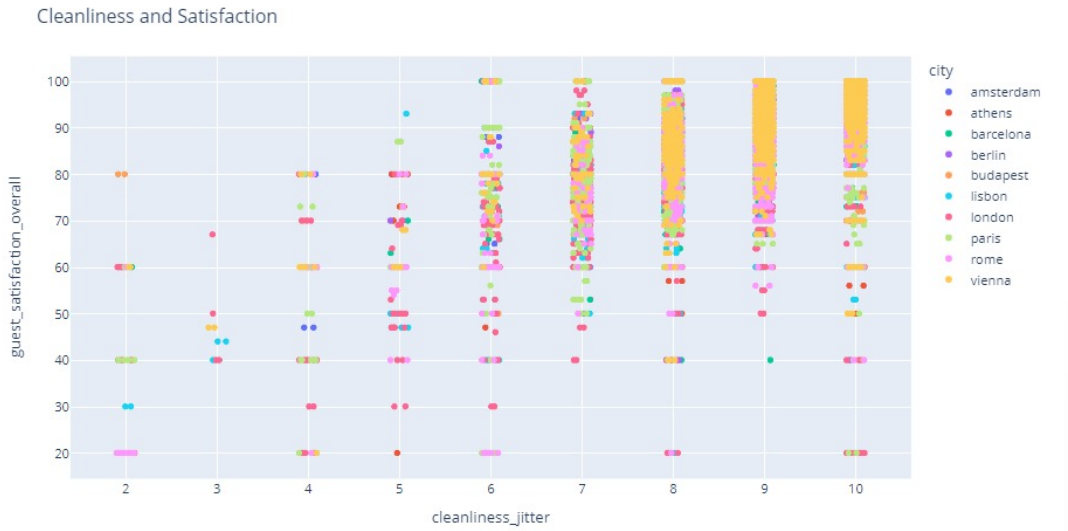

Figure 14: Clean and satisfaction

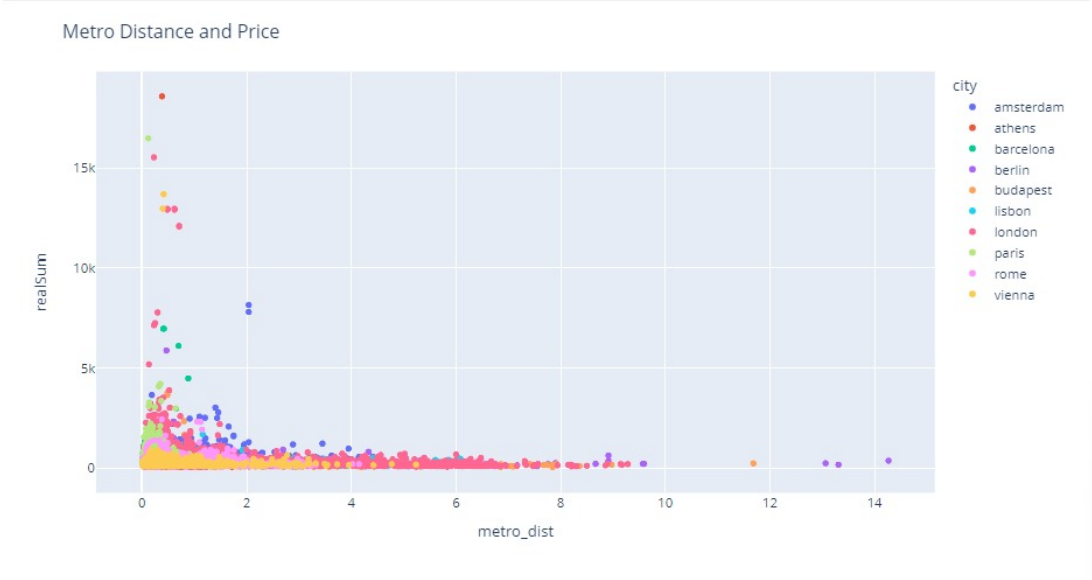

Figure 15: Distance from Metro and price

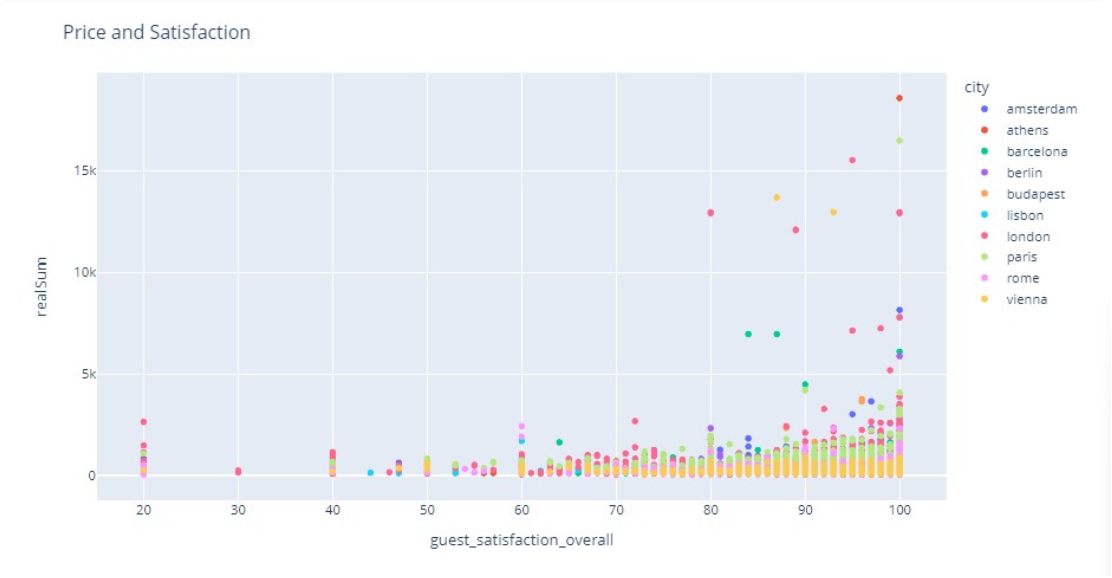

Figure 16: Satisfaction Metro and price

Some things we can observe from the plots so far are:

- Most of the listings are within 7 km of the city center while there are relatively few listings that are 7 - 10 km from the city center;
- Most of the listings are within 3 km of the closest metro while there are relatively few listings that are 3 - 5 km from the closest metro;
- The majority of the cleanliness ratings are 8-10

3.3 Data Cleaning

Data cleaning is a critical step in the data preprocessing pipeline that involves identifying and rectifying or removing errors, inconsistencies, and inaccuracies within a dataset. Outliers, in particular, are data points that significantly deviate from the expected patterns or distribution of the rest of the data. Deleting outliers is the approach we took for handling them during the data cleaning process, because later, when training the models, we had better results without outliers.

As we mentioned and observed above, we can clearly see that there are a bunch of outliers in the data. We can also see this if we look at the distributions of the

data (from Figure 17 to Figure 19).

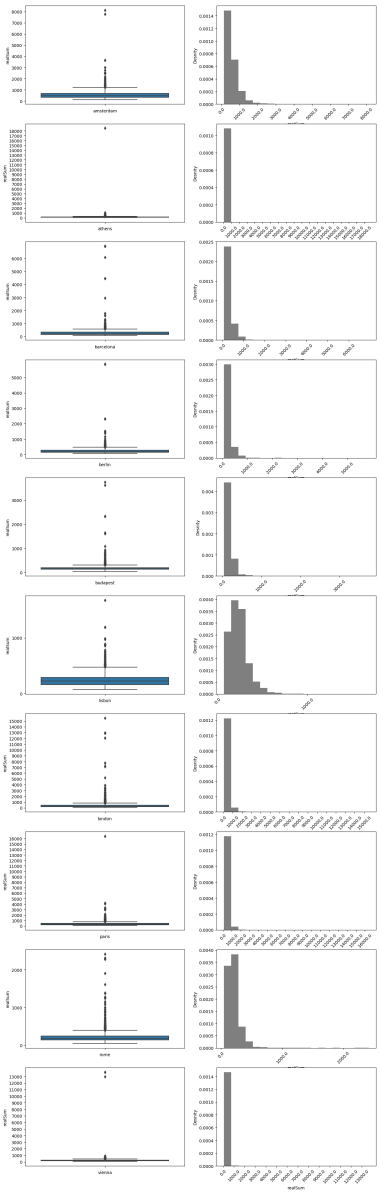

Figure 17: Initial Distribution

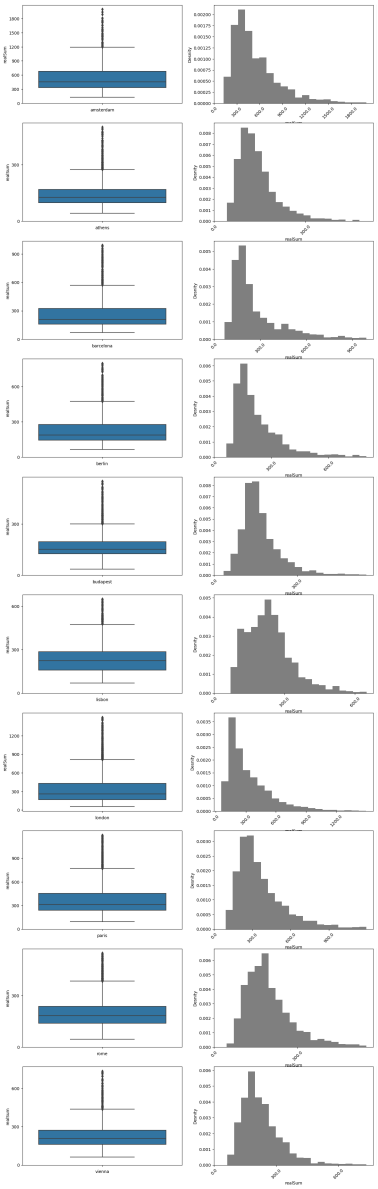

Figure 18: Distribution after removing outlines

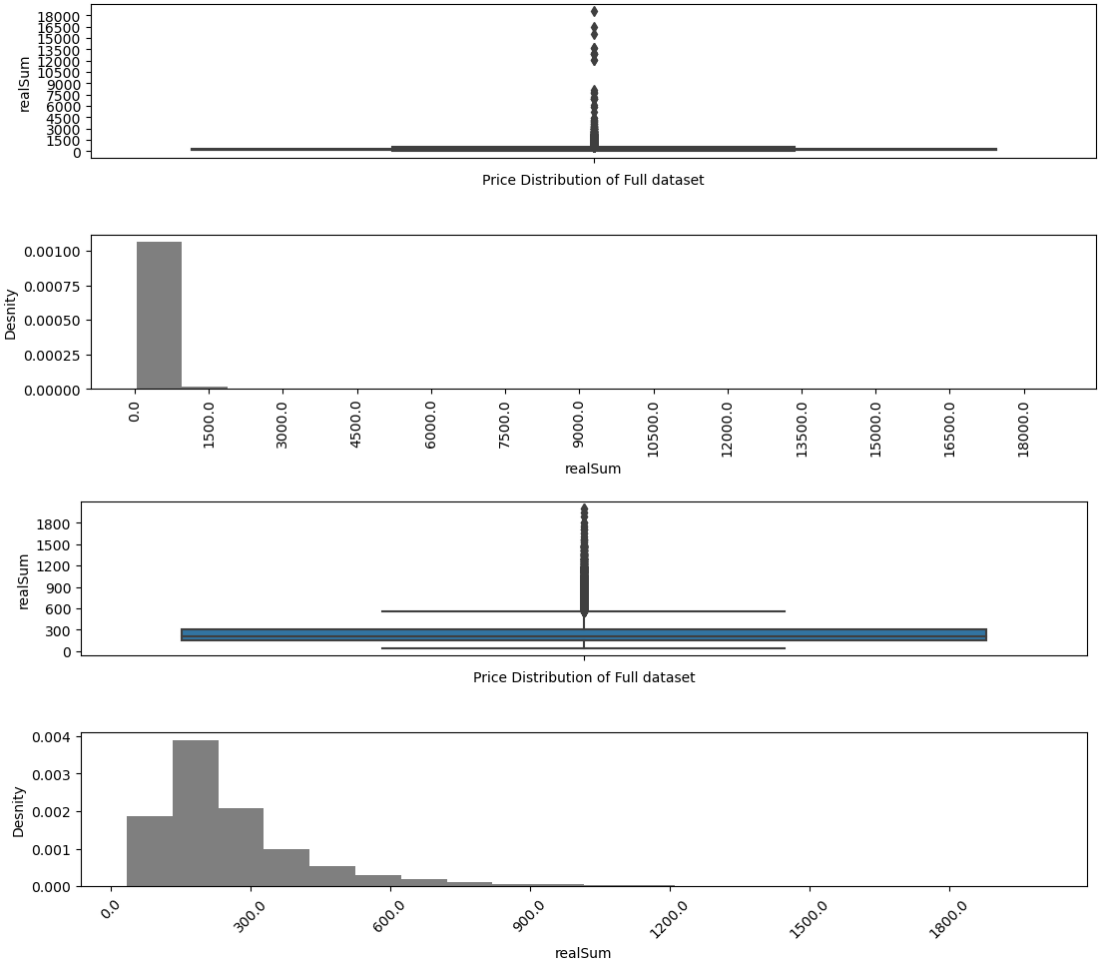

Figure 19: Initial distribution and after removing outlines (combined dataset)

Another feature that can potentially increase a property's price is the period of the week it is available. Although this doesn't come as a feature, the dataset is split between weekdays and weekend entries so it's easy to just create one. Although one would think that on the weekends, the demand would be higher, hence the price for renting would be higher, surprisingly, there is almost no difference between realSum on weekends and weekdays. This observation is illustrated in Figure 20.

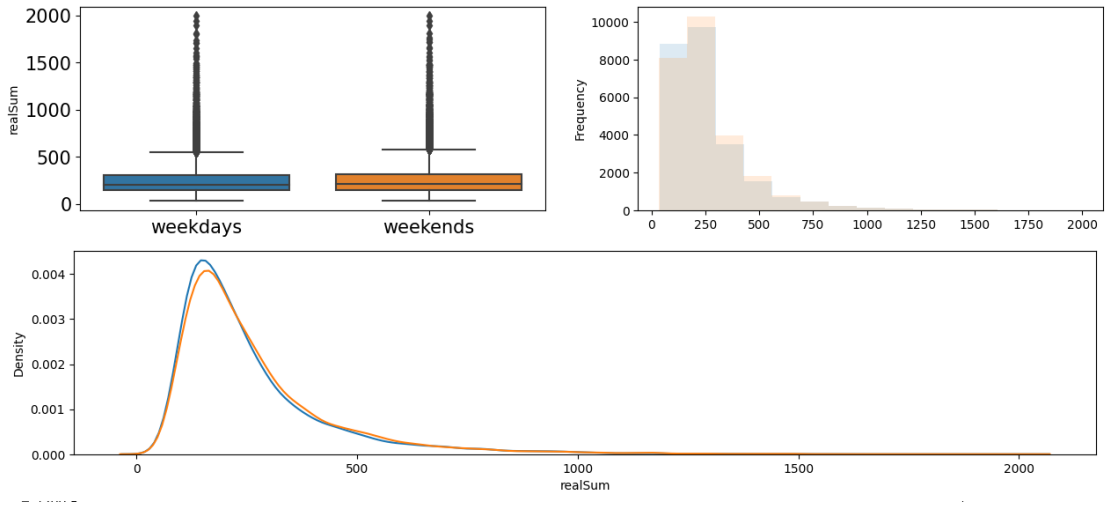

Figure 20: Price Plots between weekdays and weekend rental properties

Even though we can see that, on a general scale, there is no significant difference between prices for listings on weekdays versus weekends, we want to explore the possibility that this observation is not true for every particular city. Thus, we computed the graphics from Figure 21 to Figure 30. The figures illustrate the fact that, even though at a city level there exist differences that are a bit bigger than can be seen on a general scale, those are still not very significant. The largest changes in price between weekdays and weekends can be seen in Budapest, followed by Lisbon. In cities like Athens or Rome, there are close to no differences. Because the majority of the results don't signal big changes, we decided to continue the analysis of the dataset by merging the entries from weekdays and weekends.

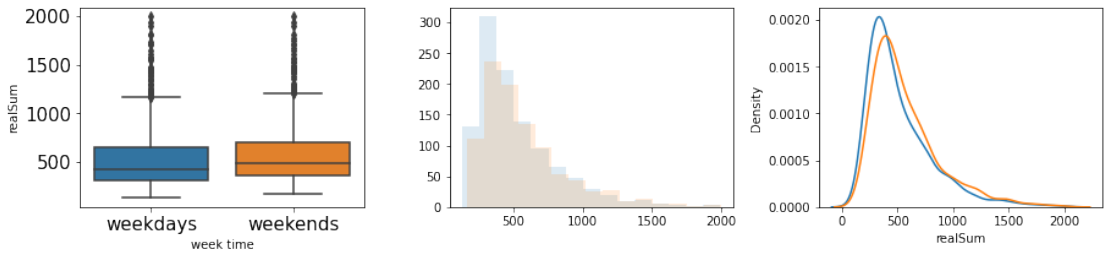

Figure 21: Price Plots between weekdays and weekend rental properties in Amsterdam

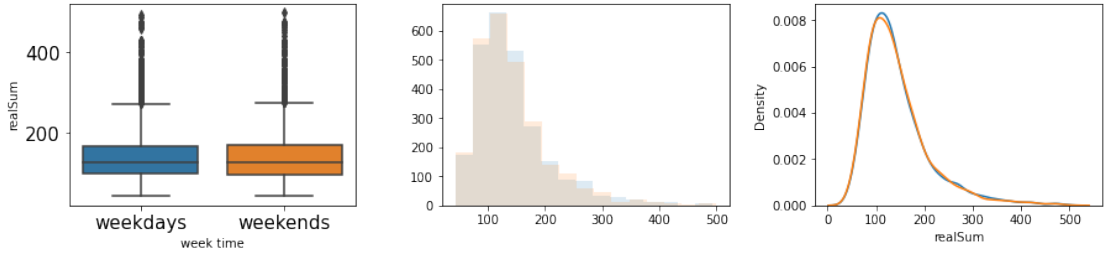

Figure 22: Price Plots between weekdays and weekend rental properties in Athens

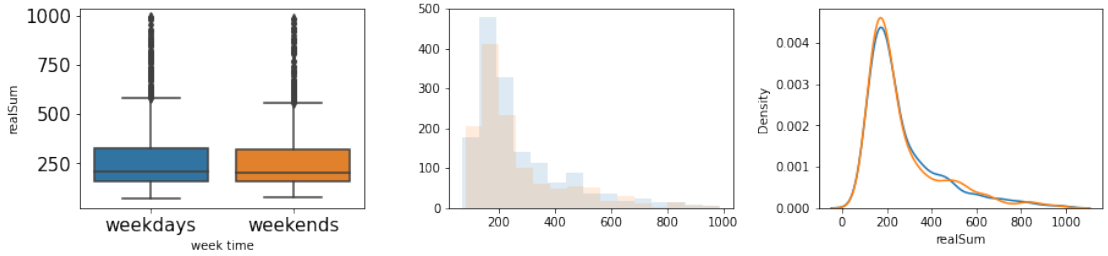

Figure 23: Price Plots between weekdays and weekend rental properties in Barcelona

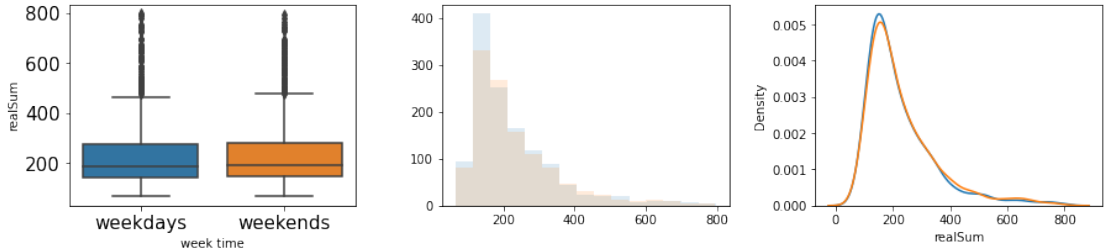

Figure 24: Price Plots between weekdays and weekend rental properties in Berlin

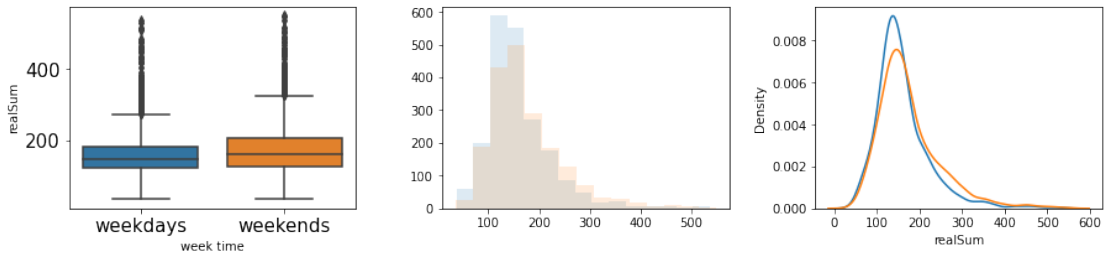

Figure 25: Price Plots between weekdays and weekend rental properties in Budapest

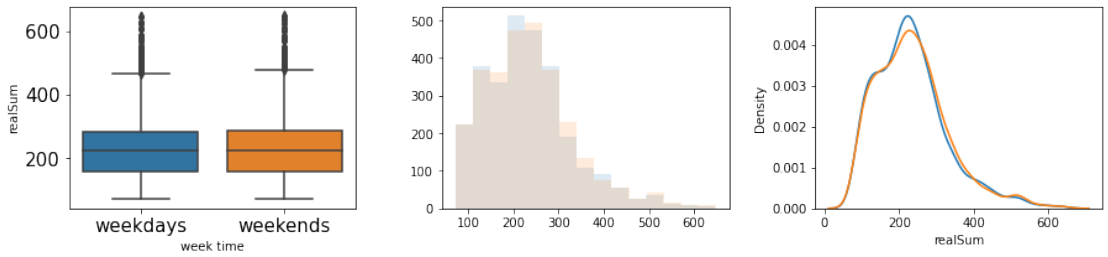

Figure 26: Price Plots between weekdays and weekend rental properties in Lisbon

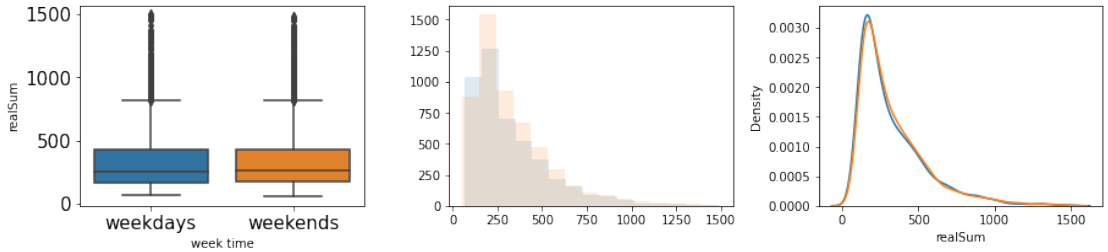

Figure 27: Price Plots between weekdays and weekend rental properties in London

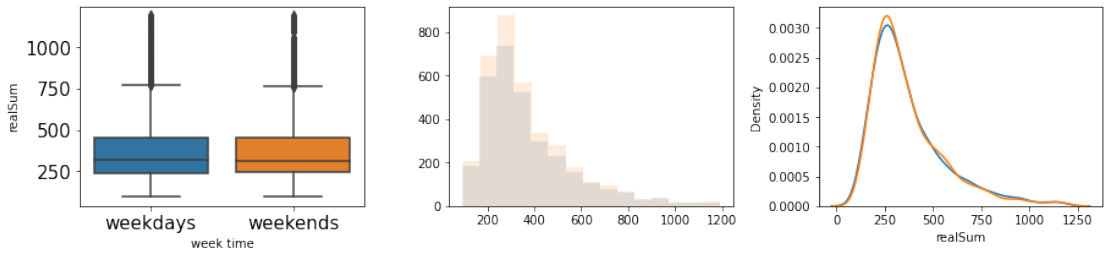

Figure 28: Price Plots between weekdays and weekend rental properties in Paris

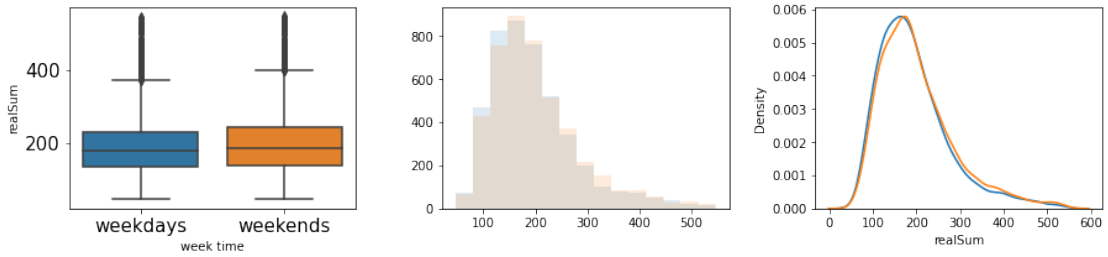

Figure 29: Price Plots between weekdays and weekend rental properties in Rome

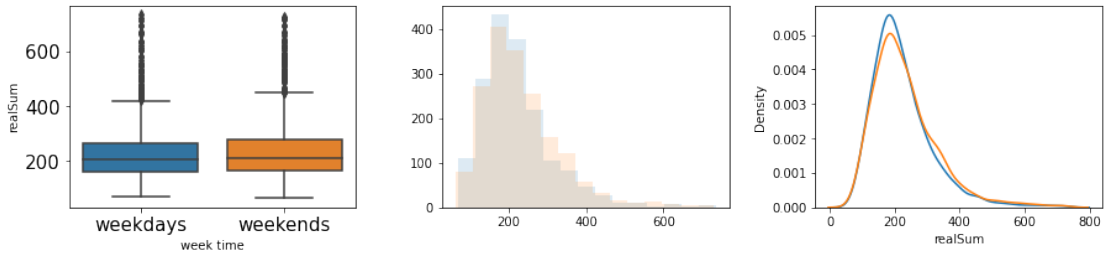

Figure 30: Price Plots between weekdays and weekend rental properties in Vienna

3.4 Data Pre-Processing

The aim of data preprocessing is to transform raw data into a clean, consistent, and meaningful format that can be efficiently analyzed. In the case of big data, preprocessing plays an even more significant role due to the sheer size and complexity of the datasets. It involves tasks such as feature selection, scaling, and

normalization. Data preprocessing in the big data context is essential for extracting valuable insights and patterns, improving the accuracy of models, and enabling efficient and scalable analysis of massive datasets.

The first step we took was replacing True and False variables with 1 and 0, this is not something huge but it helps when training the models. After that, we one-hot encoded the categorical variables (room type, week time and city). The main reason for encoding the city variable is the slight variances and differences in the median of realSum in different cities.

Next, we computed the heatmap of all the features that we are working with in Figure 31.

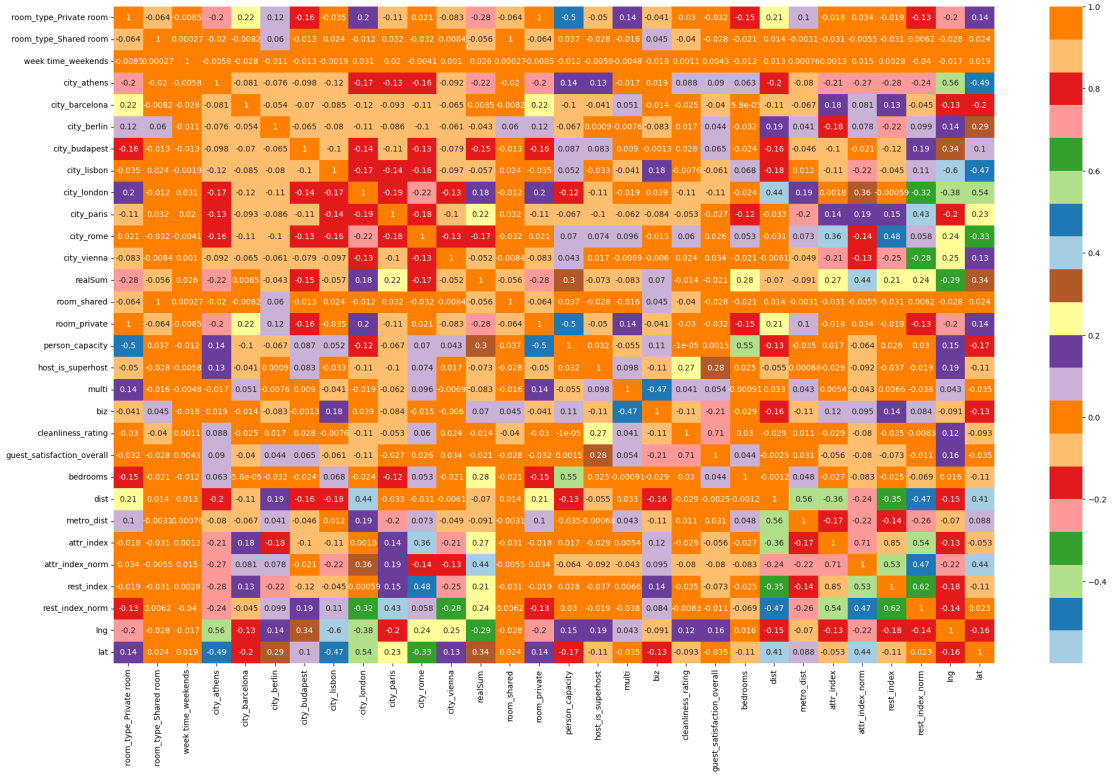

Figure 31: Heatmap of Data

With a close look at the heatmap, we can see that there are features that have perfect correlation, thus these are the redundant features and are safe to remove (i.e. room_shared, room_private). We also removed rest_index_norm and

attr_index_norm, since we have non-normalized forms of these features that provide the same information.

One more thing we did was combine the latitude and longitude features into a single one, calculated using the haversine distance formula between the latitude and longitude of the listing and co-ordinates of the null island (i.e. the point on earth where the latitude and longitude are 0). Obviously, we dropped the latitude and longitude features in favor of this one.

As a last step, we employed a Standard Scaler, for scaling, mainly, all of the features that weren't one-hot encoded. The **realSum, biz, multi, and superhost** were also not encoded, providing good info in their initial state.

4 Machine Learning Approaches (Regression Models)

4.1 First Experiment

As a first experiment, we employed regular regression models in the context of the problem.

4.1.1 Linear Regression

Linear regression is a statistical modeling technique used to establish a relationship between a dependent variable and the independent variables. It assumes a linear relationship and aims to find the best-fit line that predicts the dependent variable based on the independent variables. This model is widely applied in fields such as economics, finance, and machine learning. The coefficients of the linear equation, representing the slope and intercept, are estimated using a statistical approach. By estimating these coefficients, the model can predict the dependent variable for new values of the independent variables [1].

Table 3 shows that most of the results are around 0.45 - 0.57. Although not included, we have observed that the standard deviation for all of the experiments is close to 0.011.

4.1.2 Ridge Regression

Ridge regression is a regression technique that adds a regularization term to the loss function, making it suitable for addressing multicollinearity. It helps stabilize the model by minimizing the sum of squared residuals while considering the magnitude of the coefficients.[2]

We also concluded a grid search for this type of regression. To sum up the results, the accuracy was almost always around 0.577, while the standard deviation was around 0.011, which are pretty similar values to the ones obtained by the Linear Regression model.

The best result was obtained by the following parameters (Table 2):

alpha	**fit_intercept**	**normalize**	**solver**	R^2 **score**
0.1	True	False	sag	0.5775663075604424

Table 2: Ridge Regression results

Table 3: Linear Regression results

copy_X	**fit_intercept**	**n_jobs**	**normalize**	**positive**	R^2 **score**
True	True	None	True	False	0.5775545576941945
True	True	None	True	True	0.426381646912403
True	True	None	False	False	0.5775545576941948
True	True	None	False	True	0.4263816469124027
True	True	-1	True	False	0.5775545576941945
True	True	-1	True	True	0.426381646912403
True	True	-1	False	False	0.5775545576941948
True	True	-1	False	True	0.4263816469124027
True	False	None	True	False	0.5774749837833154
True	False	None	True	True	0.426408937291854
True	False	None	False	False	0.5774749837833154
True	False	None	False	True	0.426408937291854
True	False	-1	True	False	0.5774749837833154
True	False	-1	True	True	0.426408937291854
True	False	-1	False	False	0.5774749837833154
True	False	-1	False	True	0.426408937291854
False	True	None	True	False	0.5775545576941945
False	True	None	True	True	0.426381646912403
False	True	None	False	False	0.5775545576941948
False	True	None	False	True	0.4263816469124027
False	True	-1	True	False	0.5775545576941945
False	True	-1	True	True	0.426381646912403
False	True	-1	False	False	0.5775545576941948
False	True	-1	False	True	0.4263816469124027
False	False	None	True	False	0.5774749837833154
False	False	None	True	True	0.426408937291854
False	False	None	False	False	0.5774749837833154
False	False	None	False	True	0.426408937291854
False	False	-1	True	False	0.5774749837833154
False	False	-1	True	True	0.426408937291854
False	False	-1	False	False	0.5774749837833154
False	False	-1	False	True	0.426408937291854

4.1.3 Lasso Regression

Lasso Regression, or Least Absolute Shrinkage and Selection Operator Regression, is a powerful technique widely used in statistics and machine learning [10]. It is particularly effective in situations where datasets have a large number of features. By simultaneously performing variable selection and regularization, Lasso Regression helps to estimate coefficients in a linear regression model. It achieves this by adding an L1 regularization term, proportional to the sum of absolute coefficient values, to the standard linear regression cost function. The regularization parameter, lambda (λ), controls the degree of shrinkage and encourages sparsity in the coefficient values.

The results are pretty similar to both of the regressions presented above, with the best being shown in Table 4:

alpha	**fit_intercept**	**max_iter**	**normalize**	**selection**	R^2 **score**
0.1	True	5000	False	random	0.5775663075604424

Table 4: Lasso Regression results

4.1.4 Decision Tree Regression

Decision Tree Regression is a widely used machine learning technique that offers an intuitive and interpretable approach to solving regression problems. It is based on the concept of a hierarchical structure resembling a tree, where each internal node represents a decision based on a feature, and each leaf node corresponds to a predicted value. In Decision Tree Regression, the goal is to partition the feature space into regions that minimize the mean squared error between the predicted and actual values [14]. Decision Tree Regression is particularly advantageous in situations where interpretability is important, as it allows for a straightforward understanding of the decision-making process and can handle both continuous and discrete output variables.

The best score achieved is shown in Table 5:

max_depth	**min_samples_leaf**	**min_samples_split**	R^2 **score**
None	1	5	0.781214

Table 5: Decision Tree Regression results

We can clearly see that this type of regression is more efficient than the other ones.

4.1.5 Random Forest Regression

Random Forest Regression is a powerful and widely used machine learning technique that combines the concepts of decision trees and ensemble learning to tackle regression problems. It is based on the principle of creating an ensemble, or a collection, of multiple decision trees, known as a random forest [8]. Each tree in the random forest is trained on a different subset of the data and a random subset of features. During the prediction phase, the random forest combines the predictions of individual trees to obtain a final aggregated prediction. This ensemble approach helps to improve the accuracy and robustness of the regression model, reducing the risk of overfitting. Random Forest Regression is particularly suitable for handling complex datasets with a large number of features, as it can effectively capture non-linear relationships, handle missing data, and provide insights into feature importance. It is a versatile and widely adopted technique in various domains, offering a balance between accuracy, interpretability, and computational efficiency.

The best result for our grid search, which is also the best result we achieved in this experiment, was Table 6:

n_estimators	**max_depth**	**min_samples_split**	**min_samples_leaf**	R^2 **score**
50	60	2	2	0.854140

Table 6: Random Forest Regression results

4.1.6 ElasticNet Regression

ElasticNet Regression is a powerful regression technique that combines the strengths of both Ridge Regression and Lasso Regression [9]. It is designed to overcome some limitations of these individual methods while incorporating their beneficial features. ElasticNet Regression employs a hybrid regularization term that consists of both L1 (Lasso) and L2 (Ridge) penalties. This allows ElasticNet Regression to simultaneously perform feature selection and control the magnitude of the coefficients. By adjusting a mixing parameter, it is possible to control the balance between L1 and L2 regularization, enabling fine-tuning of the model's

behavior.

Although it sounds good, the results aren't very pleasing with ElasticNet Regression, having the lowest R^2 score out of the 6 models, at about **0.39** accuracy.

4.2 Experiment 2

As a second experiment, we employed different dimension reduction methods. We also set a threshold of 50 euros, so that we can consider approximate results also correct. The reasoning behind this is the fact that, without proper knowledge of more conditions, the price may seem random. Even for a human, the lack of images and an impression of the look of the properties puts us at an impasse, with numerical data not being sufficient for justifying some of the prices.

4.2.1 PCA

Principal component analysis (PCA) is a popular technique for analyzing large datasets containing a high number of dimensions/features per observation, increasing the interpretability of data while preserving the maximum amount of information, and enabling the visualization of multidimensional data [6]. Formally, PCA is a statistical technique for reducing the dimensionality of a dataset.

We applied the PCA transformation (**with all components kept**) and used the models presented above, with the best parameters found in our grid searches. The number of components chosen computed the best results (all smaller numbers had smaller scores). The results are as follows (Table 7):

Model	R^2 **score**
LinearRegression	0.6311179567346376
Ridge	0.631232901905791
Lasso	0.6395549322972942
DecisionTreeRegressor	1
RandomForestRegressor	0.9720149190878059
ElasticNet	0.38517041350115566

Table 7: The models' results after PCA

4.2.2 Truncated SVD

This transformer performs linear dimensionality reduction by means of truncated singular value decomposition (SVD) [7]. Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with sparse matrices efficiently.

The results obtained are very similar to the ones obtained with the PCA. In Table 8 we can observe the scores for SVD (**with 23 components**). The number of components chosen computed the best results (all smaller numbers had smaller scores).

Model	R^2 **score**
LinearRegression	0.6311869238373295
Ridge	0.631232901905791
Lasso	0.6372790179084576
DecisionTreeRegressor	1
RandomForestRegressor	0.9596236083370102
ElasticNet	0.38331779026735224

Table 8: The models' results after SVD

4.2.3 UMAP

UMAP (Uniform Manifold Approximation and Projection) is a flexible and scalable dimensionality reduction technique that captures both local and global structures in high-dimensional data [5]. Unlike traditional methods, UMAP constructs a topological representation of the data and optimizes a low-dimensional embedding to preserve the connectivity patterns. By leveraging graph-based techniques and optimization methods, UMAP reveals intricate structures and maintains separation between classes or clusters. It excels at preserving both large-scale trends and small-scale neighborhoods, making it a valuable tool for exploratory data analysis and pattern recognition. With its scalability to large datasets, UMAP allows for efficient exploration and visualization, enabling researchers and data scientists to gain insights into the underlying structure of complex data.

The results for UMAP (**with 2 components**) are as follows (Table 8):

Model	R^2 score
LinearRegression	0.4403319616542909
Ridge	0.5782662036401145
Lasso	0.5530410794425895
DecisionTreeRegressor	1
RandomForestRegressor	0.9596521839309764
ElasticNet	0.38331779026735235

Table 9: The models' results after UMAP

5 Machine Learning Approaches (Classification Models)

Since the training time for our models was very big, due to the size of the data, we decided to apply dimensionality reduction methods directly.

5.1 Experiment 3

For this experiment, we divide the prices into ranges. We chose as intervals the categories: 0-124, 125-149, 150-199, 200-249, 250-299, 300-349, 350-399, 400-499, 500-749, 750-999, 1000-1499, 1500-2999, 3000+. This approach is followed with the intent of trying to use also classifying models with this data set. These results were computed after applying the Lasso Regression to the data set. The selected features after the Lasso Regression are room type, person capacity, bedrooms, dist, attr index norm, rest index norm, lng-lat.

5.1.1 Logistic Regression Classifier

A powerful method for binary classification, logistic regression [11] is a supervised learning method. What the algorithm does is to determine whether or not a data point is part of a class by assigning it a probability. As an extra preprocessing step, we normalised the data using a normaliser with the l2 norm, available in Scikit-learn. We also decided to tune the following hyperparameters:

- Regularisation parameter, C $\in \{1, 2, 3, 4, 5, 0.5, 10\}$
- Solver used by the model, solver $\in \{lbfgs,\ newton-cg,\ saga\}$

We performed a grid search with 5 folds for 21 candidates for the hyperparameters of the logistic regression model. In Table 10 we present the best 10 results.

Table 10: Top 10 Scores with Parameters

C	Solver	Score
1	lbfgs	0.333
1	newton-cg	0.333
1	saga	0.333
2	lbfgs	0.333
2	newton-cg	0.333
2	saga	0.333
3	lbfgs	0.333
3	newton-cg	0.333
3	saga	0.333
4	lbfgs	0.333

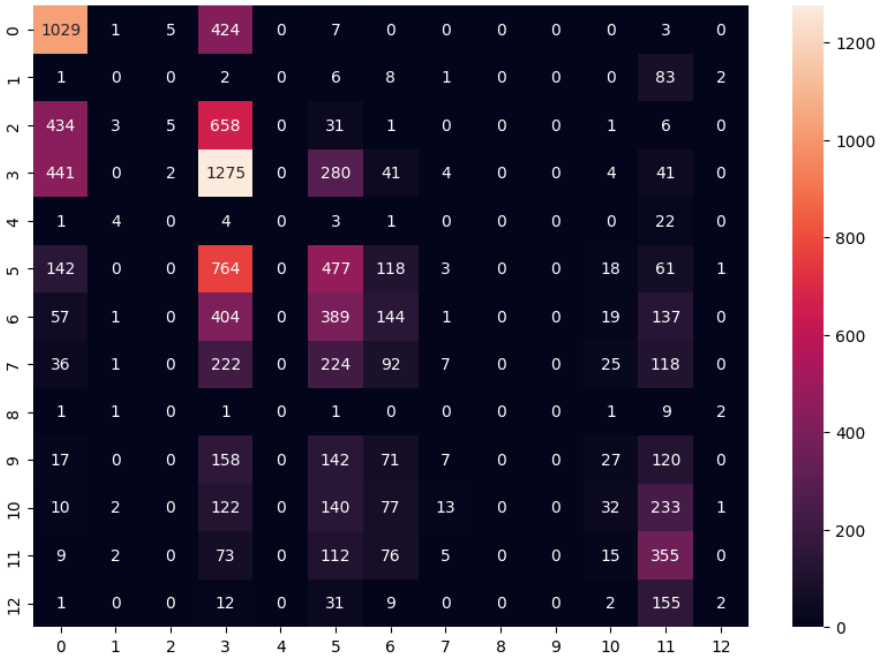

Figure 32: Confusion Matrix for the best Logistic Regression model

5.1.2 KNN

A simple classification algorithm, k-nearest neighbors (KNN) is [12] a supervised learning method. This algorithm considers each labeled data point as part of a vector space. From there, the class of a new data point is determined by choosing the most prevalent class amongst the k closest labeled data points according to some chosen metric. We have chosen to use the usual Euclidean distance as our metric, leaving the number of neighbors k as the only hyperparameter to be tuned.

We performed a grid search with 5 folds for 12 candidates for the hyperparameters of the KNN model. In Table 11 we present the best 15 results.

Leaf Size	**Number of Neighbors**	**Score**
1	3	0.431
2	3	0.431
3	3	0.431
1	5	0.409
2	5	0.409
3	5	0.409
1	7	0.399
1	10	0.390
2	10	0.390
3	10	0.390

Table 11: The best 10 results for the KNN model

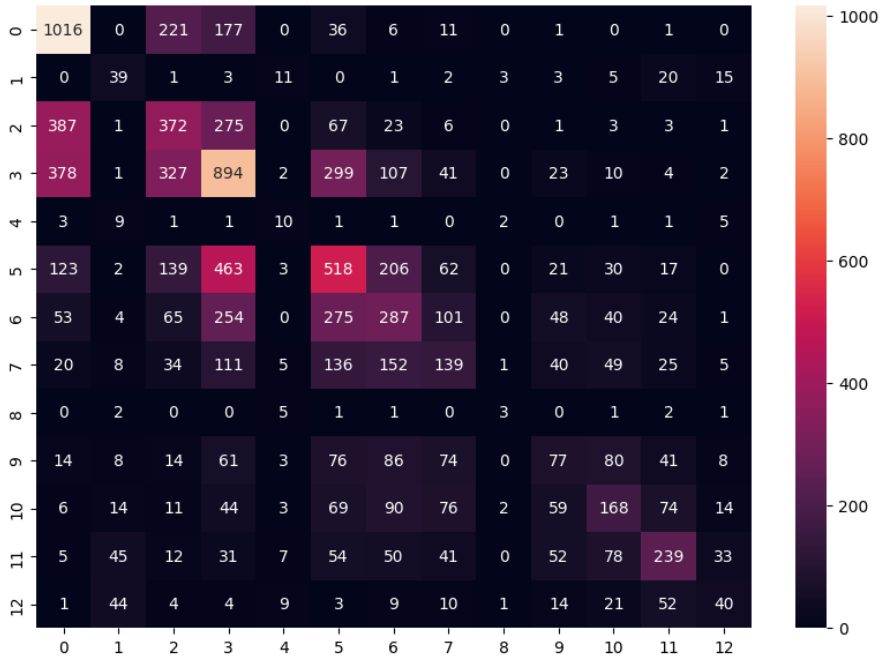

Figure 33: Confusion Matrix for the best KNN model

5.1.3 Decision Tree Classifier

A decision tree classifier[14] is a type of supervised machine learning model used for classification problems. It works by creating a tree-like model of decisions and their possible consequences. The tree is constructed by breaking down the data into smaller subsets, and at each step, a decision is made about which feature to split the data on, based on which split will result in the greatest reduction in impurity. The process is repeated recursively until a stopping criterion is met, such as a minimum number of samples in a leaf node.

Each internal node in the tree represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label. The final class label for a new sample is determined by traversing the tree from the root to a leaf, making the decisions at each internal node based on the values of the sample's features.

We conducted a grid search to optimize the performance of our classifier by tuning a crucial parameter known as the criterion. We considered two potential criteria, namely gini and entropy. After a thorough evaluation, we determined

that the 'entropy' criterion emerged as the superior choice, achieving an accuracy of 0.55.

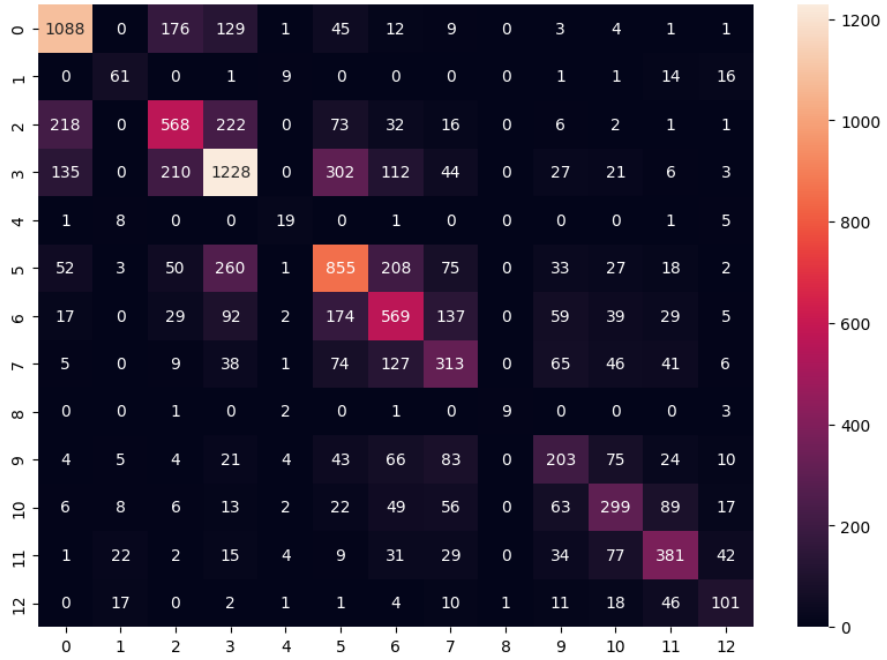

Figure 34: Confusion Matrix for the best Decision Tree Classifier model

5.1.4 Random Forest Classifier

[8] The random forest classifier consists of a large number of individual decision trees that operate as an ensemble, each individual tree in the random forest outputs a class prediction and the class with the most votes becomes the final prediction. It is a supervised learning algorithm that produces even without hyperparameter tuning great results.

One big advantage of this algorithm is that it can be used for both regression and classification problems. Being a derivative of the decision tree classifier, it uses almost the same hyperparameters in a training batch.

Random forest adds additional randomness to the model while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

The main limitation of random forest is that a large number of trees can make the algorithm too slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained.

We performed a grid search with 3 folds on 10 candidates for the hyperparameters on random forest. Following this we present the best results.

Criterion	Estimators	Score
entropy	500	0.635
gini	200	0.630
gini	500	0.629
entropy	200	0.632
gini	100	0.624

Table 12: Top 5 Scores with Parameters of Random Forest Classifier

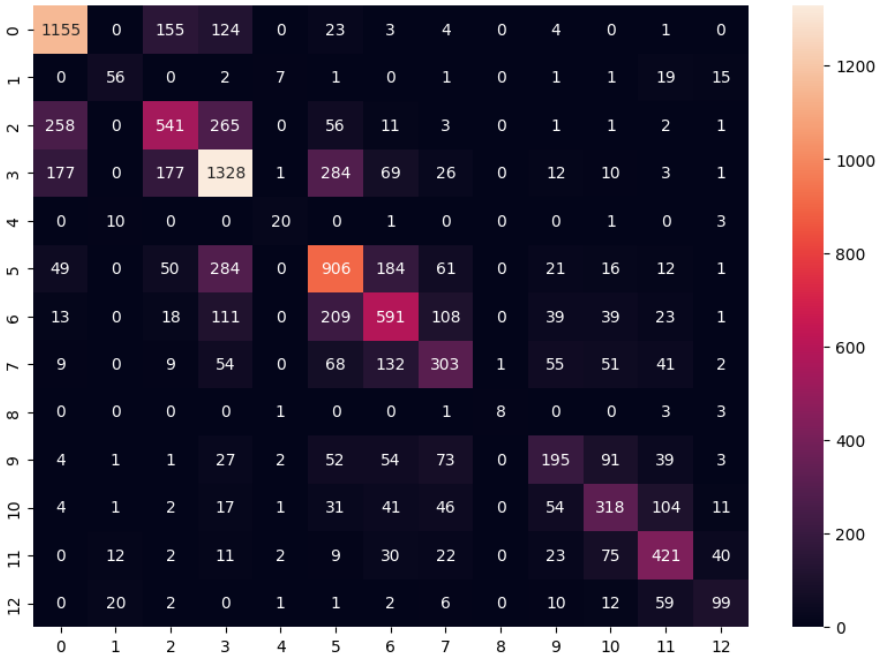

Figure 35: Confusion Matrix for the best Random Forest Classifier model

5.1.5 MLP

The Multilayer Perceptron (MLP) is composed of multiple layers of artificial neurons, also known as perceptrons, arranged in a feedforward manner. Each neuron in the MLP receives inputs, performs a weighted sum of the inputs, applies an activation function to the sum, and passes the result as output to the next layer. The hidden layers in an MLP allow it to learn increasingly abstract representations of the input data, leading to higher-level features. The final layer, known as the output layer, produces the network's prediction or classification. Training an MLP involves iteratively adjusting the weights and biases through backpropagation, which calculates the gradient of the loss function with respect to the network's parameters.

With multi-layer perceptron being a Deep learning algorithm, it strives to 'converge' to a result during its operations, doing so, it might hit the limit of iterations it can go through, thus 'crashing' the learning sequence. The most we could get out of it was an accuracy of 0.369.

In the next table, we will showcase the results of the grid-search:

Score	**Parameters**
0.369	{'activation': 'relu', 'learning_rate': 'adaptive', 'solver': 'adam'}
0.366	{'activation': 'tanh', 'learning_rate': 'constant', 'solver': 'adam'}
0.365	{'activation': 'tanh', 'learning_rate': 'constant', 'solver': 'lbfgs'}
0.362	{'activation': 'tanh', 'learning_rate': 'adaptive', 'solver': 'adam'}
0.361	{'activation': 'relu', 'learning_rate': 'adaptive', 'solver': 'adam'}
0.361	{'activation': 'relu', 'learning_rate': 'adaptive', 'solver': 'sgd'}
0.358	{'activation': 'tanh', 'learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.358	{'activation': 'relu', 'learning_rate': 'adaptive', 'solver': 'lbfgs'}
0.357	{'activation': 'relu', 'learning_rate': 'constant', 'solver': 'adam'}
0.353	{'activation': 'tanh', 'learning_rate': 'constant', 'solver': 'sgd'}

Table 13: Top 10 Scores with Parameters of MLP

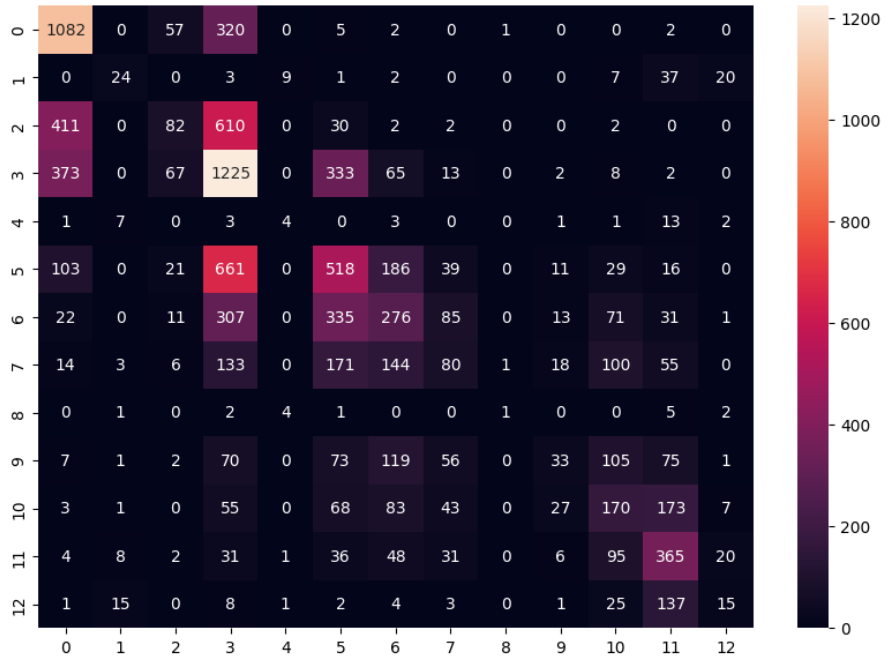

Figure 36: Confusion Matrix for the best MLP model

5.1.6 Support Vector Classification

A Support Vector Classifier, or SVC for short, [13] is a supervised learning algorithm that relies on building hyperplanes between different data points in order to separate them. The planes are chosen based on their distance to their support vectors and the amount of correctly classified points. From here stem the concepts of hard margins(minimum distance to the data points) and soft margins(allows some points on the wrong side of the margin or hyperplane), which are controlled by the regularization parameter. Harder margins tend to overfit, while lower margins tend to misclassify more often. Another thing of note about SVC is, in case the data is not separable, it can be transposed into a plane in which it becomes possible through a technique called kerneling.

Here are the 10 best folds after fitting 3 folds for 27 candidates:

Table 14: Top 10 Scores with Parameters

Score	C	Gamma	Kernel
0.364	2	0.1	rbf
0.360	1	0.01	rbf
0.351	2	0.1	poly
0.334	2	0.001	linear
0.334	1	0.001	linear
0.334	2	0.1	linear
0.328	2	0.001	rbf
0.325	2	0.01	rbf
0.315	1	0.001	rbf
0.314	2	0.1	rbf

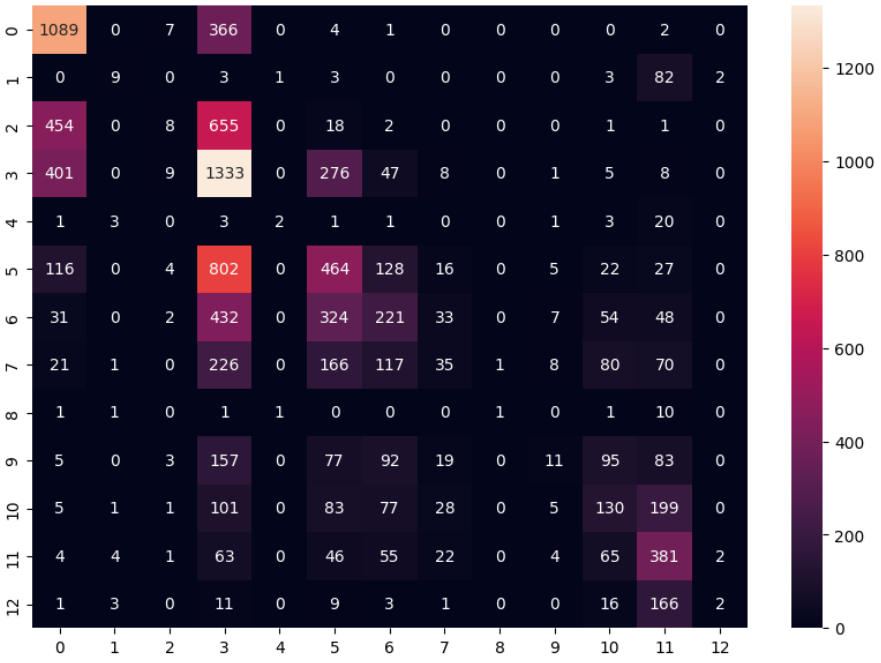

Figure 37: Confusion Matrix for the best SVM model

5.2 Experiment 4

In this experiment, we continue with the bins approach, and we apply PCA on the set. What stands out in this experiment is the fact that we computed the models also on the data that had outliers, not only on the cleaned version. We also split the data into training and testing sets. Since grid search performs k-fold validation on the training set itself, no further splitting is necessary.

5.2.1 PCA with Outliers

We allowed PCA to create 19 feature vectors, which explain the variance as follows:

Component	**Value (%)**
PC0	18.05%
PC1	14.07%
PC2	10.28%
PC3	7.84%
PC4	6.68%
PC5	6.24%
PC6	5.38%
PC7	5.23%
PC8	4.62%
PC9	4.26%
PC10	3.82%
PC11	3.14%
PC12	2.78%
PC13	2.05%
PC14	1.99%
PC15	1.64%
PC16	1.42%
PC17	0.48%
PC18	Very close to 0%

Table 15: Principal components on data with outliers

As can be seen in the table, the first 10 principal components explain roughly 80% of the variance, as such we can drop the rest of the generated features for a significant reduction in size.

We wanted to generate the best results, so we applied grid searches for all the models that we previously used for classification. The best results yielded by our grid search were:

Model	**Parameters**
Logistic Regression	C=0.5, max_iter=2000, penalty='l2', solver='lbfgs'
KNN	leaf_size = 1, n_neighbors = 10
SVM	C = 2, gamma = 0.001, kernel = 'rbf'
Decision Tree Classifier	criterion ='entropy'
Random Forest Classifier	criterion ='gini', n_estimators = 50
MLP Classifier	activation ='tanh', learning_rate = 'constant', solver='sgd'

Table 16: Models and their best parameters

After conducting an extensive grid search, we evaluated the models with the obtained parameters on the test set.

Model	**Accuracy**
Logistic Regression	0.288
KNN	0.345
SVM	0.288
Decision Tree	0.344
Random Forest	0.399
MLP	0.352

Table 17: Models and their accuracy on data with outliers

Accuracy is not the goal in this experiment however. While it is a useful metric to go by, we are more interested to see how the properties are misclassified. If, for example, a property in bin 5 is assigned to bin 4 or 6, that's not a big price difference (therefore the property may be priced pretty fairly) compared to the property being assigned to bin 0 or bin 10 (undervalued or overvalued). As such, we shall take a look at the obtained confusion matrices.

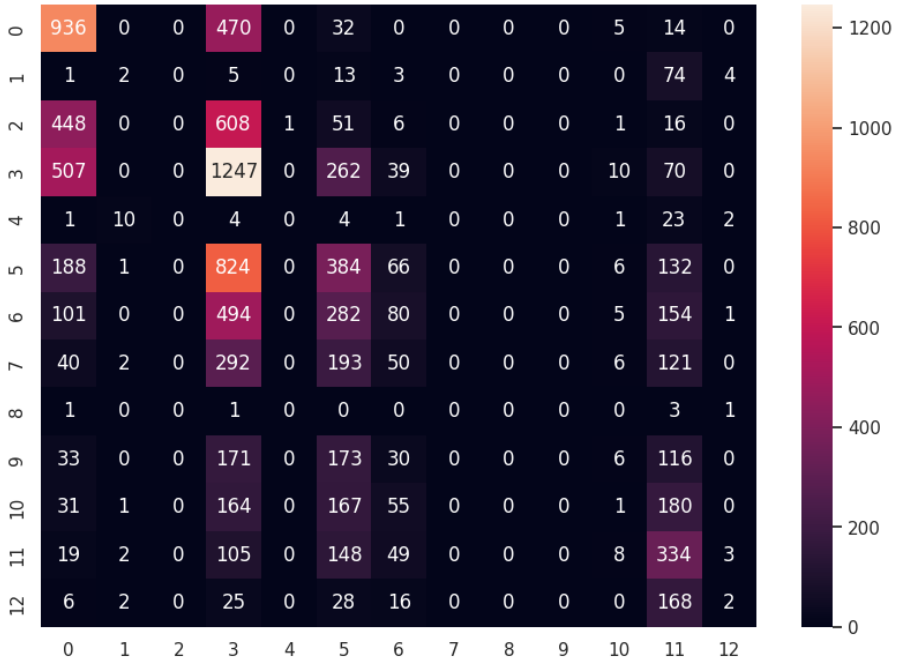

Figure 38: Confusion Matrix for Linear Regression with Outliers

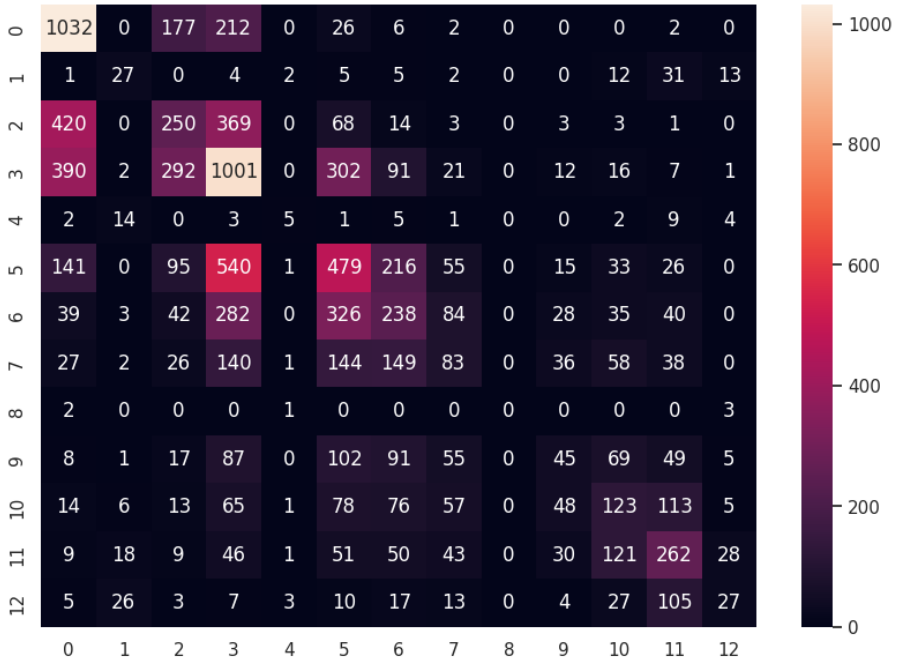

Figure 39: Confusion Matrix for KNN with Outliers

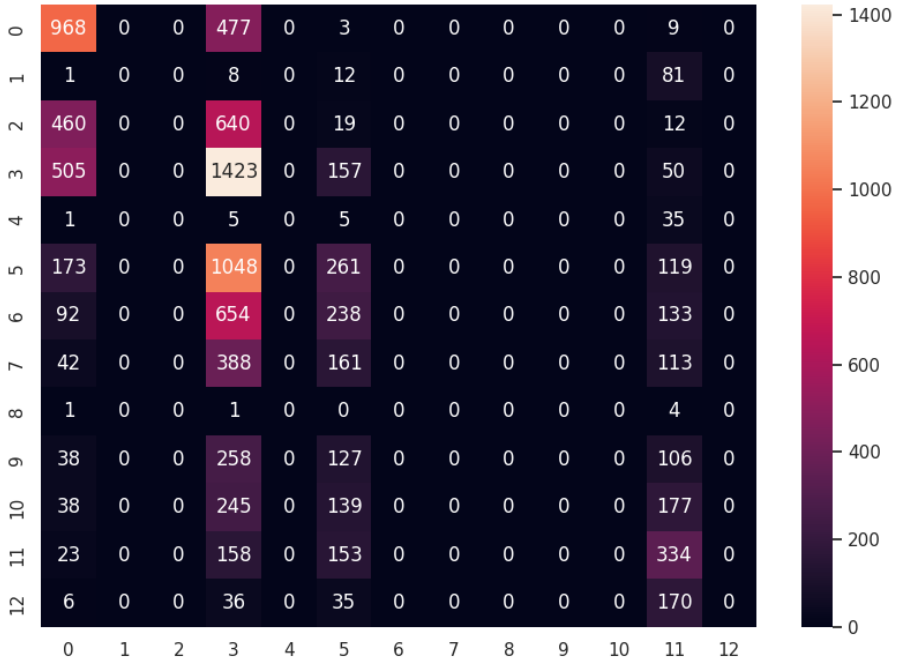

Figure 40: Confusion Matrix for SVM with Outliers

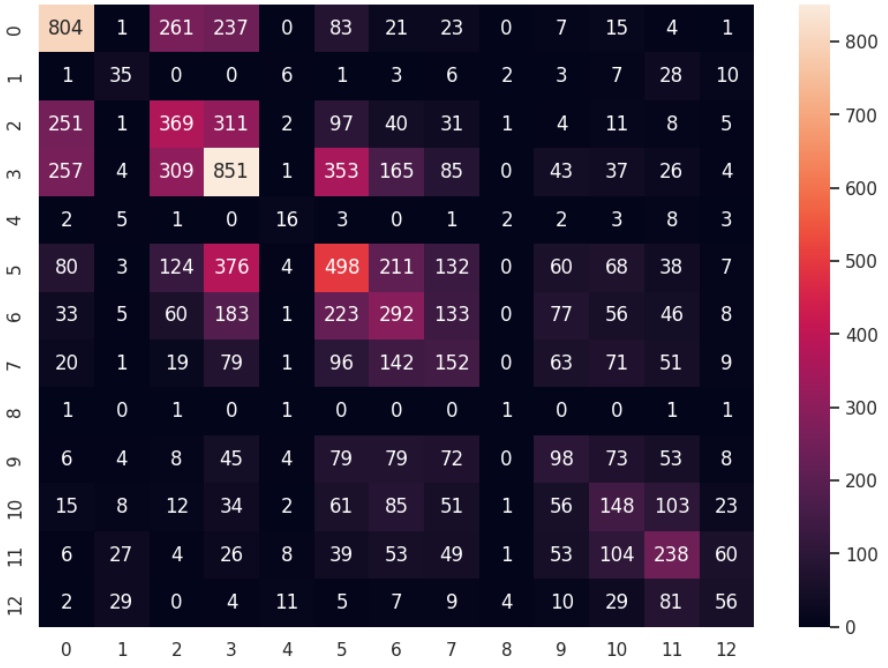

Figure 41: Confusion Matrix for Decision Tree with Outliers

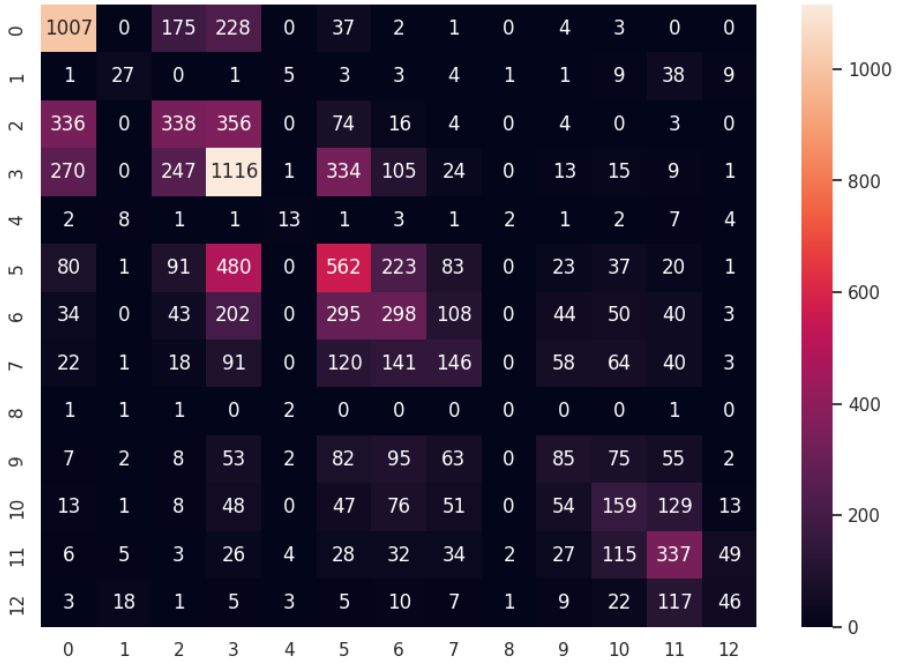

Figure 42: Confusion Matrix for Random Forest with Outliers

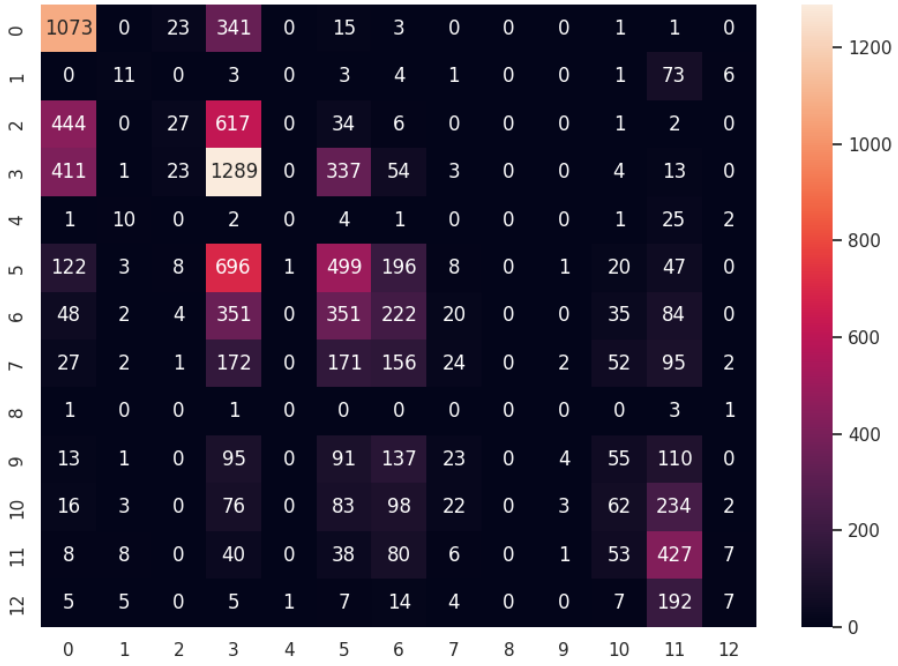

Figure 43: Confusion Matrix for MLP with Outliers

5.2.2 PCA without Outliers

The percentage of the variance explained by the PCA features is a little different than in the previous case:

Component	Value (%)
PC0	18.05%
PC1	14.10%
PC2	10.30%
PC3	7.80%
PC4	6.68%
PC5	6.24%
PC6	5.38%
PC7	5.23%
PC8	4.63%
PC9	4.25%
PC10	3.82%
PC11	3.14%
PC12	2.78%
PC13	2.06%
PC14	1.99%
PC15	1.66%
PC16	1.42%
PC17	0.48%
PC18	Very close to 0%

Table 18: Principal components on data without outliers

However, the first 10 features also explain roughly 80% of the variance, as such, we shall keep the same number of principal components.

Cleaning up the data did improve the accuracy on validation, but did not affect the parameters of the models. As such, the accuracy on the test set for our models is:

Model	Accuracy
Logistic Regression	0.284
KNN	0.342
SVM	0.284
Decision Tree	0.339
Random Forest	0.391
MLP	0.352

Table 19: Models and their accuracy on data without outliers

The confusion matrices we obtained after running our models are as follows:

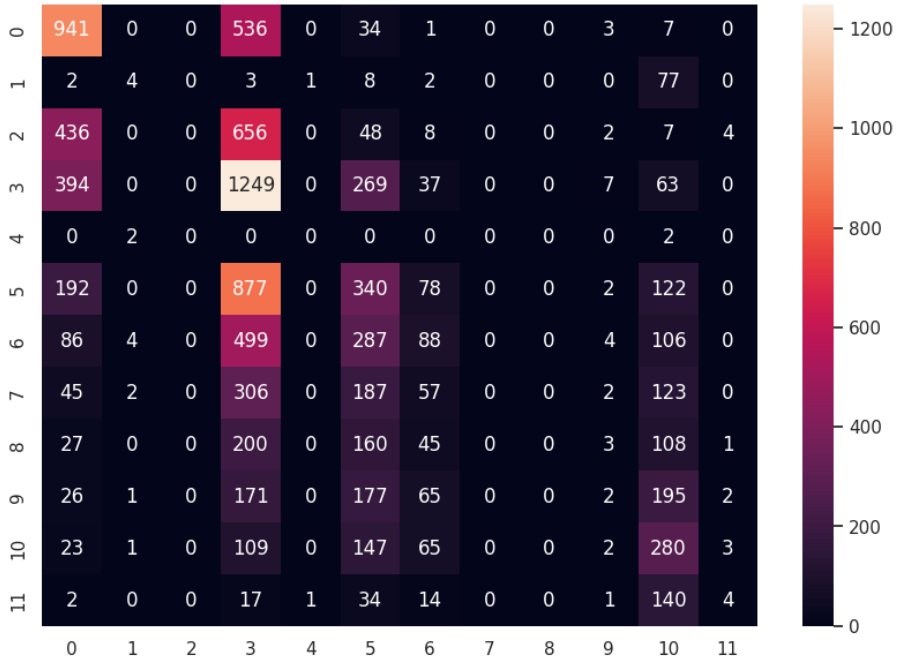

Figure 44: Confusion Matrix for Linear Regression without Outliers

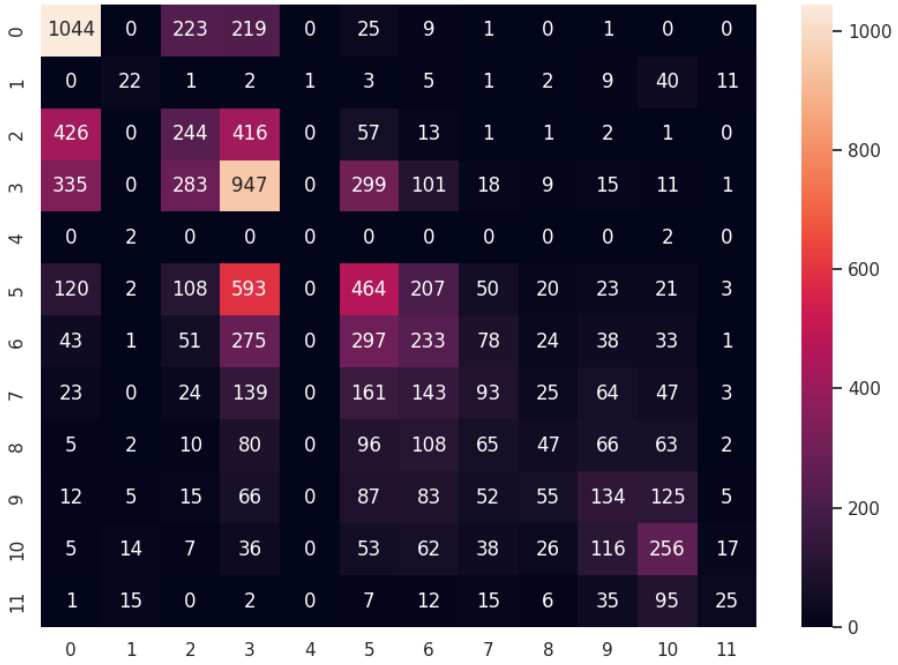

Figure 45: Confusion Matrix for KNN without Outliers

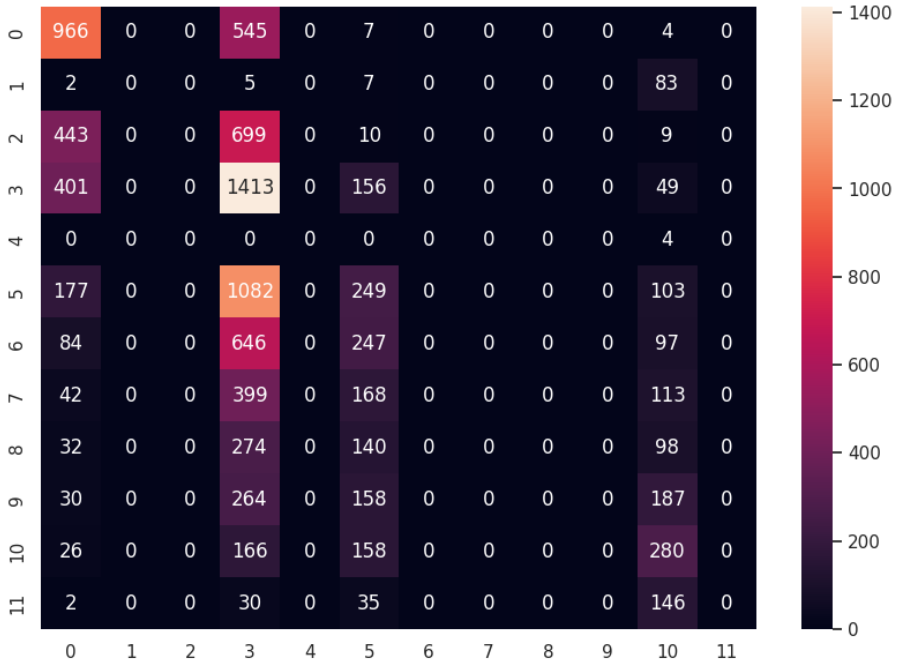

Figure 46: Confusion Matrix for SVM without Outliers

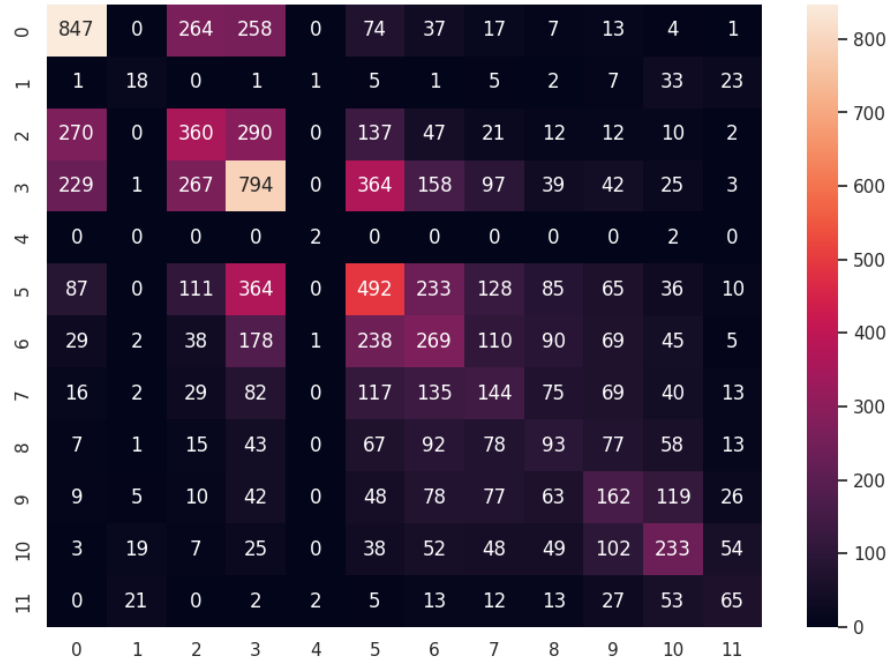

Figure 47: Confusion Matrix for Decision Tree without Outliers

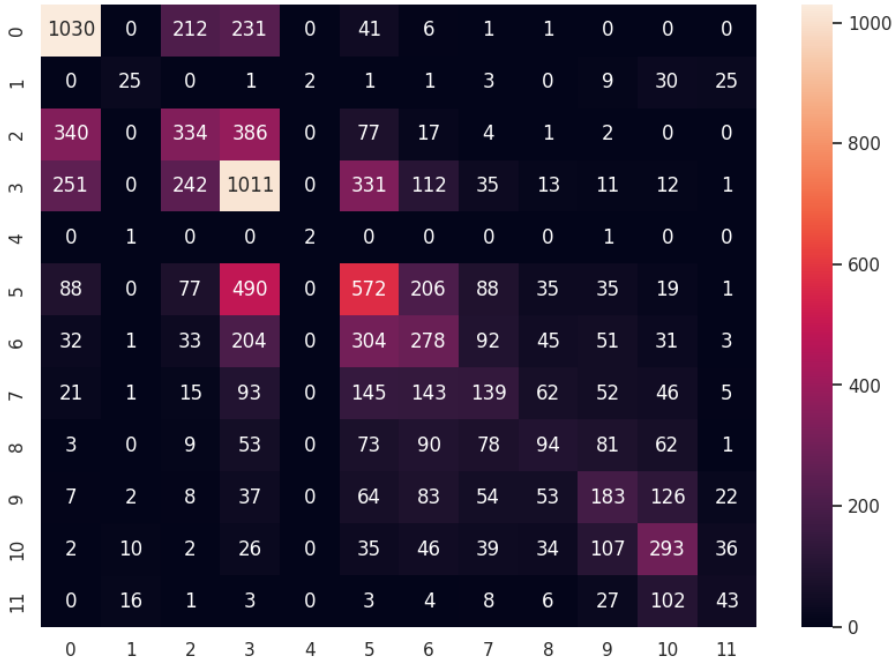

Figure 48: Confusion Matrix for Random Forest without Outliers

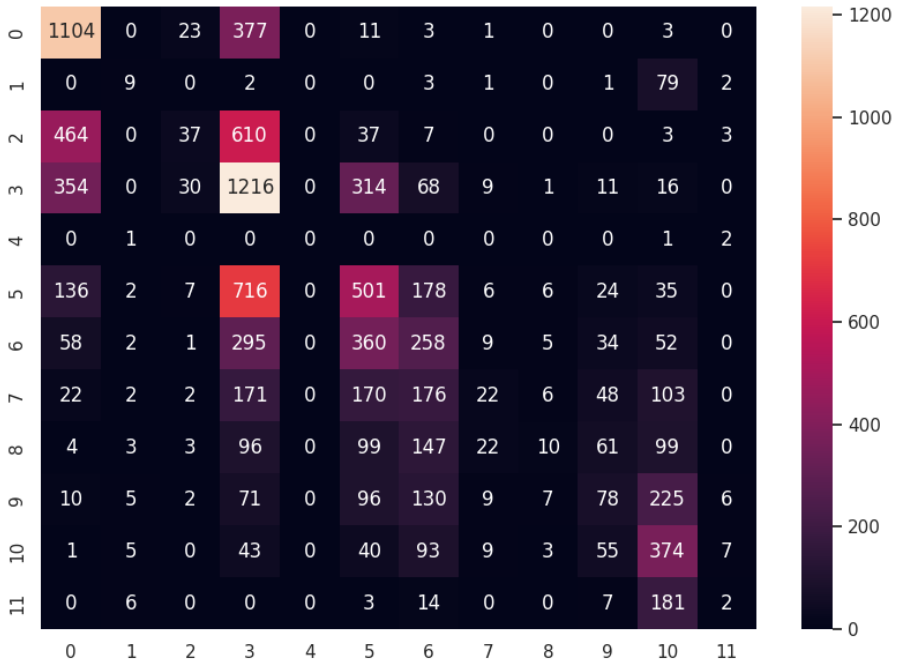

Figure 49: Confusion Matrix for MLP without Outliers

6 Summary of the results

In the next two tables, we present a summarisation of the results obtained during our study.

Regression Model	**Reduction method**	R^2 **score**
Linear Regression	No reduction method	0.57
Linear Regression	PCA	0.63
Linear Regression	SVD	0.63
Linear Regression	UMAP	0.44
Ridge Regression	No reduction method	0.57
Ridge Regression	PCA	0.63
Ridge Regression	SVD	0.63
Ridge Regression	UMAP	0.57
Lasso Regression	No reduction method	0.57
Lasso Regression	PCA	0.63
Lasso Regression	SVD	0.63
Lasso Regression	SVD	0.55
Decision Tree Regression	No reduction method	0.78
Decision Tree Regression	PCA	1
Decision Tree Regression	SVD	1
Decision Tree Regression	UMAP	1
Random Forest Regression	No reduction method	0.85
Random Forest Regression	PCA	0.97
Random Forest Regression	SVD	0.95
Random Forest Regression	UMAP	0.95
ElasticNet Regression	No reduction method	0.39
Elastic Net Regression	PCA	0.38
Elastic Net Regression	SVD	0.38
Elastic Net Regression	UMAP	0.38

Table 20: Regression Models Summary of Results

Regression Model	Reduction method	Outliers	Accuracy
Logistic Regression Classifier	Lasso	Without outliers	0.33
Logistic Regression Classifier	PCA	With outliers	0.28
Logistic Regression Classifier	PCA	Without outliers	0.28
KNN	Lasso	Without outliers	0.43
KNN	PCA	With outliers	0.34
KNN	PCA	Without outliers	0.34
Decision Tree Classifiers	Lasso	Without outliers	0.55
Decision Tree Classifier	PCA	With outliers	0.34
Decision Tree Classifier	PCA	Without outliers	0.33
Random Forest Classifier	Lasso	Without outliers	0.63
Random Forest Classifier	PCA	With outliers	0.39
Random Forest Classifier	PCA	Without outliers	0.39
MLP	Lasso	Without outliers	0.36
MLP	PCA	With outliers	0.35
MLP	PCA	Without outliers	0.35
SVM	Lasso	Without outliers	0.36
SVM	PCA	With outliers	0.28
SVM	PCA	Without outliers	0.28

Table 21: Classification Models Summary of Results

7 Conclusion and Future Work

In conclusion, this paper explored the application of reduction methods for training machine learning (ML) algorithms on big data. The results demonstrate that incorporating reduction methods significantly enhances the performance and effectiveness of ML algorithms compared to scenarios without reduction. The reduction methods employed (PCA, Truncated SVD and UMAP), effectively tackled the challenges posed by big data, including high dimensionality, computational complexity, and information redundancy. By reducing the data size, eliminating irrelevant or redundant features, and optimizing the representation of instances, the reduction methods improved the efficiency, accuracy, and scalability of the ML algorithms.

The best results that we had were computed by the Decision Tree Regressor after the reduction methods were applied (perfect accuracy). A close second was the Random Forest Regressor, which had an accuracy of over 95% with all three reduction methods. The lowest results that we had were computed with the bins experiments, with overall better, but not really significant, results on the cleaned data set. It is clear that for this task, the regression models were better suited.

As for future work, we believe that training the regression models on the data with the outliers, after performing the reduction methods, would be interesting. The training for this experiment takes a lot of time and that is the reason why we didn't do it for the present project, but it is something to take into consideration in the future.

References

[1] BRIAN BEERS. Introduction to linear regression model. https://www.investopedia.com/terms/r/regression.asp. [accessed: 24.05.2023].

[2] Saptashwa Bhattacharyya. Ridge and lasso regression: L1 and l2 regularization. [accessed: 25.05.2023].

[3] Kristóf Gyódi and Łukasz Nawaro. Determinants of airbnb prices in european cities: A spatial econometrics approach. *Tourism Management*, 86:104319, 2021.

[4] Łukasz Gyódi, Kristóf; Nawaro. Determinants of Airbnb prices in European cities. https://zenodo.org/record/4446043#.ZHSGNXZBxPZ, 2021. [Accessed 30.05.2023].

[5] Leland McInnes. sklearn.decomposition.TruncatedSVD Documentation. https://github.com/lmcinnes/umap. [accessed: 27.05.2023].

[6] sklearn. sklearn.decomposition.PCA Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. [accessed: 27.05.2023].

[7] sklearn. sklearn.decomposition.TruncatedSVD Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html. [accessed: 26.05.2023].

[8] sklearn. sklearn.ensemble.RandomForestClassifier Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [accessed: 24.05.2023].

[9] sklearn. sklearn.linear_model.ElasticNet Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html. [accessed: 27.05.2023].

[10] sklearn. sklearn.linear_model.Lasso Documentation. https://scikit-learn.org/stable/modules/linear_model.html#lasso. [accessed: 25.05.2023].

[11] sklearn. sklearn.linear_model.LogisticRegression Documentation. https://imbalanced-https://scikit-learn.org/stable/modules/

generated/sklearn.linear_model.LogisticRegression.html. [Accessed 23.052023].

[12] sklearn. sklearn.neighbors.KNeighborsClassifier Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed 20.052023].

[13] sklearn. sklearn.svm.SVC Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed 28.05.2023].

[14] sklearn. sklearn.tree.DecisionTreeClassifier Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.htmll. [accessed: 26.05.2023].