Analysing different machine learning models in order to estimate Airbnb prices in Amsterdam

Manon Dubost (2675627), Benedikt Gagro (2610225), Sebastian Keil (2664850), Paulo Maia (2594494), and Roshini Ramasamy (2610864)

VU University Amsterdam

Abstract. The goal of the paper is to find the best machine learning models to estimate the price of an Airbnb in Amsterdam using a dataset from kaggle. Three different feature sets were compiled and the data set was studied with heat maps plotted according to feature correlation. Highly correlated features were removed, creating a lower dimensional feature space. Another feature set was extracted using Principal Component Analysis. Using cross-validation in combination with grid and evolutionary search, the best hyper parameters for each model were estimated and performance was measured. The following regression models were found to be performing best in the optimization phase, and thus chosen to be evaluated on the test data: Lasso Regressor, KNN Regressor, SVM Regressor, Neural Network, Random Forest Regressor, Extra Trees Regressor and Gradient Boosting Regressor. The metric used to determine the best fit was the Root Mean Square Error. The models scores varied between 42.35 and 46.79 in which, one model outperformed. From the findings of the paper, the hypothesis was supported.

Keywords: Machine Learning · Airbnb Prices · Estimation · Linear Models · k-Nearest Neighbors Regression · Support Vector Machine · Ensemble and Boosting Regression · Neural Networks

1 Introduction

The topic of this research paper is the optimization and comparison of different machine learning methods in the setting of a classic regression problem. These algorithms make use of statistical, probabilistic and tree-based optimization techniques [1] to find meaningful relations and patterns inside a large set of data. The problem that was chosen is the estimation of Airbnb prices in Amsterdam. Our goal is to accurately predict the price of an apartment depending on where it is located in Amsterdam, as well as other factors.

The data set used in this project was taken from kaggle.com. It is comprised of 16 features such as number of bedrooms and bathrooms, distance to the centre, room type, number of guests, and minimum nights. Further, the total number of entries is 14.997, which should provide us with enough data to reliably optimize and test the performance of different models. In order to achieve high accuracy for our price prediction model, multiple machine learning algorithms will be utilized,optimized and compared at the end.

1.1 Motivation

Airbnb is one of the most popular used services for short-term stay. They offer low as well as high priced apartment options located all over the city. It is a serious competitor to the hotel business

and offers a very unique experience to its customers. This is due to the fact that the apartments are being subleased from local residents. Meaning that the local residents are free to make up their own prices without being subject to municipality rules or regulations. This is due to the fact that Airbnb acts as a private entity. To make sure that customers only pay as much as the apartment is actually worth, we try to create a true value estimator based on an apartment's features by using the power of machine learning. This way we optimize the price-quality ratio for every customer's particular requirements.

1.2 Hypothesis

We will be comparing multiple algorithms written in python in order to determine the one with the highest performance. The classification methods used are kNN regression, support vector machine, ensembles and boosting methods, simple linear model, and neural networks. We assume the data set to be normally distributed and thus to contain only a few outliers, which will be tested during data exploration.

Our intuition is that it will be fairly easy to identify trends in our data, as there are some key features that contribute heavily towards determining the price of an apartment. The expectation is that the best model will be one of the ensembles type methods since it makes use of a number of machine learning techniques such as bagging, boosting, and stacking. Additionally, we predict that the neural networks will be outperformed by the ensemble models as neural networks tend to overfit the data whereas ensembles do not.

1.3 Literature review

There are a number of research papers that attempt to estimate the rental price of an apartment. A few of them will be described in the following paragraphs.

To our surprise, many research papers chose a similar approach to ours. Namely, selecting a specific set of features and then comparing a number of machine learning algorithms. The research paper by Ye et al. [2] on "Customized Regression Model for Airbnb Dynamic Pricing" incorporated the dynamic pricing notion, where businesses set flexible prices for products and services based on current market demands. Their pricing system initially develops a binary classification model to predict the booking probability and then proceeds by creating a regression model that estimates the optimal price trained by minimizing a customized loss function.

In another paper, by Kalehbasti, Nikolenko, and Rezaei, called "Airbnb Price Prediction Using Machine Learning and Sentiment Analysis" [3], the main focus was on previous customer reviews. The models that were used are sentiment analysis, linear regression, tree-based models, SVR and neural networks. Further, a study on Airbnb apartments in Montreal by Pow, Janulewicz, and Liu [4], has been carried out using the following models: linear regression, support vector regression (SVR), k-nearest neighbours (KNN) and regression tree/random forest regression. Research done by Cai, Han, and Wu [5] are focusing on Airbnb rentals in Melbourne, utilizing models such as linear regression, ridge regression, support vector regression, random forest regression and gradient boosting. Lastly, the paper by Ma, Zhang, Ihler and Pan [6] called "Estimating Warehouse Rental Price using Machine Learning Techniques" compared a linear regression, a regression tree, a random forest regression and a gradient boosting regression tree to estimate warehouse rental prices.

The conclusion for most of the papers is that the linear regression models tend to be the worst performing ones, with kNN, random forest regression and gradient boosting performing significantly

better. In contrast to the papers discussed above, our paper will consider all principal features available and not just a specific subset. Moreover, we choose a bigger set of machine learning algorithm classes to determine the best performing one when compared to previous research.

1.4 Methodology

In order to find appropriate models and hyper parameters for our regression problem, we will use a broad variety of methods. The main metric of interest will be the root mean squared error (RMSE), as it gives us a value that is representative of how well our models do. For instance, a RMSE of 50 means would correspond to a model that deviates from the actual Air-BnB price by 50 Euros. We decided to test models from different classes of algorithms, so we can get a broad view on what kind of method is best-suited for our problem. These classes are linear models, lazy algorithms (i.e. KNN), support vector machines, ensembles and neural networks. For each of these classes, we will test a variety of modifications using 5-fold cross-validation and compare the validation score (RMSE value). For the neural networks a single 30 percent validation set is used.

Regarding hyper-parameter optimization, we are mainly working with three kinds of methods. These are experimentation, based on our intuition, a grid search and an evolutionary search, or any combination of these. To implement evolutionary search in our parameter space, we will use TPOT, which stands for Tree-based Pipeline Optimization Tool. TPOT is a genetic programming library that automates the search of models in the pipeline space. A pipeline is an abstraction of the entire data science process, including feature extraction and parameter optimization. It uses evolutionary methods like selection, cross-over and mutation [7].

However, we will not use TPOT to search the entire pipeline space, but rather modify the algorithm in such a way that it only searches over the hyper-parameters for any given model. The exact workings of evolutionary programming are beyond the scope of this paper, however it is important to note that we used a population and offspring size of 20 across 5 generations. These values were chosen in a trade-off between computational complexity and our ambition to find solid hyper-parameters. The number of generated models is calculated by population size + generations x offspring size.

2 Data exploration

In machine learning it is generally desirable to reduce the complexity of the feature space, as long as this does not impair the model performance. The idea that we should prefer simple solution over complex ones is most neatly summarized by the dictum of 14th century scholar William of Ockham, who notoriously states that "Entities should not be multiplied without necessity." [8] This simple guiding problem-solving principle certainly applies to machine learning, where we try to reduce both model and feature complexity. This part focuses on how we can achieve the latter. The goal is that we construct three different features spaces, ranging from high variability (low bias) to low variability (high bias). These distinct sets can then be compared throughout the optimization process, providing valuable insights on where we might be running into common pitfalls such as overfitting.

2.1 The data set

The data set used in this paper was taken from kaggle, which is a data science website that offers a variety of free data sets. It consists of 14.997 entries in total. The data set is already split into

training and test data. The number of items inside the training data is 10.498 and the number of items inside the test data is 4.499. This represents a split at approximately 30% test data and 70% training data. After doing some exploratory data analysis, we find that the data set is already quite 'clean', meaning that there are no missing values and categorical features have already been encoded into numbers (e.g. using one-hot encoding). All models will be exclusively trained on the training data. The test data is only used at the end when evaluating the performance of the most promising models. The data set includes following features:

- Accommodates
- Number of bathrooms
- Number of bedrooms
- Calculated host listings
- Guests included
- Host listings count
- Latitude longitude
- Minimum nights
- Number of reviews
- Distance to centre
- Instantly bookable (true, false)
- Room type (entire apartment/house, private room, shared room)

The features "instantly bookable" and "room type" are categorical, whereas the rest is numerical.

2.2 Feature selection

In order to determine what the relevant features of the data set are, we will first assess how much information each feature provides [9]. Each feature is being plotted against all the other features in a heat map (see Fig 1).

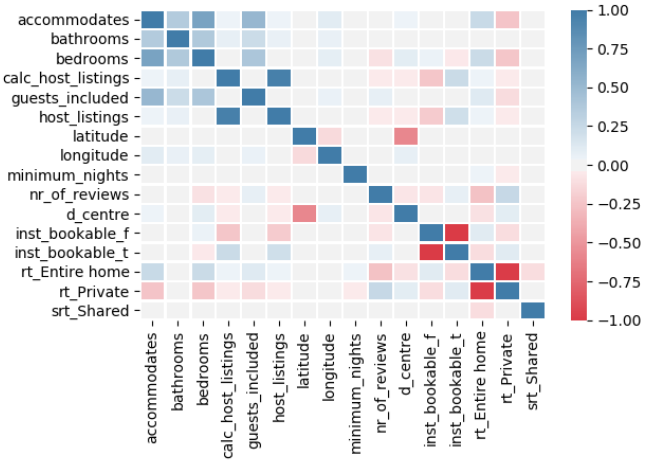


Fig. 1. Correlation plot of 16 x 16 features

The darker the color in the plot the more do these features correlate with each other, denoting that they convey the same information. Using two features that are highly correlated with each other may be redundant. Therefore those features will be eliminated (excluding the features that are being plotted against themselves). This leaves us with the following subset of features: number of bathrooms, number of bedrooms, host listings count, guests included, instantly bookable false, instantly bookable true, room type home/apt, room type private room, room type shared room.

2.3 Feature extraction

The data set discussed above still contains too many features. Not dealing with this may result in models not being able to handle it. As we want to retain as much relevant information that is provided by the data set, we are going to use Principal Component Analysis to reduce the number of features. Like this we may lose the interpretability of the newly derived features but keep as much of the original information as possible. Further, reducing the complexity of our feature space is going to reduce variance of the model performance and thus prevent overfitting. This will hopefully allow us to keep variance low and yield a more accurate predictive performance for our future models.

To start with the principle component analysis the information of our data set must first be standardized. Meaning our features have to be mapped to the standard normal distribution. Now, the data is ready for Principal Component Analysis. The new features yield the following scree plot (see Fig.2):

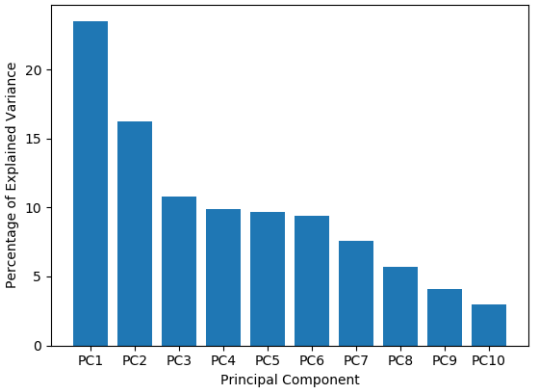


Fig. 2. Scree plot of PCA components

A clear "knee point" can be seen between principal component 6 and principal component 7. As principle component 7 only accounts for 5% of the variance, we will drop all the components after principal component 6. In summary, we were able to reduce the feature space of initially 16 features to only 6 principle components that contain the most essential information.

3 Models

3.1 Linear Models

Linear Regression (i.e. ordinary least-squares) The chosen linear models are Linear regression, Ridge regression and Lasso regression [10]. Ridge and Lasso regression are essentially modifications

of ordinary least-squares regression models which penalize the size of the weights. This means that they reward less complex models, effectively increasing bias and decreasing variance. This is achieved by adding a regularization term λ to the regular regression model. For Ridge Regression, the weight magnitude in the error term is squared, which makes it a L2 regularizer.

$$min(w) \quad \|Xw - y\|^2 + \lambda\|w\|^2 \tag{1}$$

Lasso Regression utilizes L1 regularization, which also rewards models with fewer weights.

$$min(w) \quad \frac{1}{2n}\|Xw - y\|^2 + \lambda\|w\| \tag{2}$$

In order to find the optimal linear model, we experiment with different regularizers and values of λ (note: λ is here called alpha).

Table 1. 5-fold cross validation results, using grid search

	Full Features		Selected Features		PCA Features	
	RMSE	r-squared	RMSE	r-squared	RMSE	r-squared
LinearReg	46.19	0.4746	48.1	0.4302	52.55	0.3201
RidgeReg (alpha=0.1)	46.19	0.4745	48.1	0.4303	52.55	0.3201
LassoReg (alpha=0.1)	46.18	0.4746	48.1	0.4303	52.55	0.3201

3.2 KNN Regression

The KNN Regression Model is one of the simplest models out there. It is however very efficient at certain tasks. It can be used for classification and regression problems. In our case we will use it for regression. The KNN model uses feature similarity to attribute values to the data points. The way it works is very straightforward. A new instance will be attributed a value depending on its k closest neighbors. Let's say K=5, then the value attributed will be the mean of its 5 closest neighbours. For our problem KNN regression has proven to be very effective.

Table 2. Results for the KNN Regression Model

	Full Features		Selected Features		PCA Features	
	RMSE	r-squared	RMSE	r-squared	RMSE	r-squared
k=1	59.6	0.1247	61.07	0.0796	64.47	-0.0251
k=5	45.83	0.4825	48.1	0.4298	49.75	0.3904
k=10	44.43	0.5137	46.54	0.4664	48.28	0.4259
k=14	44.06	0.5217	46.28	0.4724	47.88	0.4354
k=20	44.02	0.5225	46.05	0.4776	47.68	0.4400

It is instructive to examine how the cross-validated error converges with the differently chosen values for k. We can see that the point of interest lies around k=14 (See Fig. 3 in the appendix).

3.3 Support Vector Machine

A Support Vector Machine (SVM) is a linear supervised machine learning model which can be used in classification and regression problems, both linear and non-linear using the kernel trick. The model looks at the extremes of the data and draws a hyperplane near the boundaries. The points closest to the decision boundary are called the support vectors as they can be used to describe the decision boundary and the distance to the support vectors is called the margin. It identifies the hyperplane that has the maximum margin from the closest point which therefore maximises the minimum distance. Using these support vectors, we maximize the margin of the classifier.

For pattern analysis and to get comparative data, three different kernel types were used for the data namely a radial basis function kernel, polynomial kernel and a linear kernel using evolutionary search. Each one was assigned different hyperparameters depending on the specific kernel. The results are shown in the figure below.

Table 3. SVM regression using rbf-kernel

Rank	RMSE	C	gamma
1	45.63	10	0.05
2	46.8	5	0.005
3	48.85	7	0.001
4	49.5	1	0.005
5	49.73	5	0.001

After conducting the evolutionary search, we further optimize the value of γ, using grid search around the values found by the evolutionary algorithm (see Appendix for the full result tables). We find that $\gamma = 0.03$ gives the the best-performing model.

3.4 Ensembles and Boosting Methods

Random Forest Regression This model works on the idea of "the wisdom of crowds". In other words the information in multiple uncorrelated trees put together will outperform an individual model. This can be very powerful as the trees protect each other from their own mistakes. Together they can point us in the right direction. To find good hyper-parameters for this model, we search for different number of trees and whether or not to use bootstrapping, using evolutionary search.

Table 4. Random Forest Regression on Full Feature set

Rank	RMSE	Bootstrap	N Estimators
1	43.33	True	200
2	43.39	False	10
3	43.49	False	100
4	43.5	True	140
5	43.57	True	100

Gradient Boosting Regression Gradient Boosting is a powerful, tree-based ensemble method, which boosts a number of weak learners (fixed sized decision trees) and is thus able to effectively capture complex patterns in the data. Gradient Boosting uses greedy search to build an additive model of the form:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{3}$$

where $h_m(x)$ is the function that describes individual weak learners and γ_m describes the step length. Each additionally added tries to minimize the loss of the previous ensemble $F_m - 1$:

$$h_m = arg \quad min(h) \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i)) \tag{4}$$

The step length is calculated via the derivative of the individual weak learner loss functions:

$$\gamma_m = arg \quad min(\gamma) \quad \sum_{i=1}^{n} (y_i, F_{m-1}(x_i) - \gamma \frac{\partial L(y_i, F_{m-1}(x_i)}{\partial F_{m-1}(x_i)} \tag{5}$$

Gradient Boosting also utilizes a learning rate, which modifies the contribution of each tree. Since there is a trade-off between the learning rate and the number of trees, we search over a variety of values for these parameters using evolutionary search:

Table 5. Results for Gradient Boosting Regression on Full Feature set

Rank	RMSE	Learning Rate	n estimators
1	41.84	0.1	200
2	41.85	0.1	250
3	41.91	0.1	150
4	41.93	0.1	50
5	42.16	1	100

Extra Trees Regression Extra Trees Regression works very similarly to Random Forest, with the main difference being in how features are split. While Random Forest splits features on the most telling threshold (i.e. the one that gives the highest information gain), Extra Trees randomly draws a variety of thresholds randomly and selects the best-performing option. This additional element of randomness tends to slightly reduce the variance of the model. To find the optimal hyper parameters for this model, we consider whether or not samples should be bootstrapped, the number of trees in the forest (n estimators) and minimum number of samples required to be at a leaf node. We find those parameters using evolutionary search.

Table 6. Results for Extra Tree Regression on Full Feature set

Rank	RMSE	Bootstrap	N Estimators	Min Samples Leaf
1	42.5	True	100	5
2	42.52	True	260	5
3	42.53	True	240	10
4	42.6	False	160	25
5	42.61	True	100	5

3.5 Neural networks

We construct different (fully-connected) Neural Nets, all using the reLU activation function and rmsprop optimization [11]. Rmsprop is an optimizer that is commonly used to combat vanishing gradients in full-batch regression problems. It essentially works by combining the ideas of only using the sign of each gradient and adapting the step size for each weight individually. We then we seek to make changes in the number of hidden layers and number of nodes per layer to find the optimal architecture.

Table 7. Results for fully connected NN using the reLU activation function and rmsprop optimization

		Full Features	Selected Features	PCA Features
#Hidden Layers	#Nodes	RMSE	RMSE	RMSE
1	10	47.45	48.59	55.72
1	50	44.29	46.08	52.28
2	hl:1 20 hl:2 10	44.07	45.85	51.43
2	hl:1 30 hl:2 20	43.53	45.78	51.12
2	hl:1 30 hl:2 30	43.68	45.64	50.26

4 Results

4.1 Evaluation

After extensively cross-validating and optimizing different models and hyper parameters, we can finally measure the performance of our most promising models on the test data. From each class of models, we choose the modification that yielded the best results during the optimization phase. Given that the ensembles performed on average better than all the other models, we will chose one modification of each ensemble to compete in the final testing rounds. These are the models we chose (and their associated hyper parameters)

- Lasso Regression (alpha=0.1)
- KNN Regression (k=14)
- SVM Regression (C=10, gamma=0.03)

– Neural Network (h1:30, h2:20)
– Random Forest Regression (Bootstrap=True, n est.=200)
– Extra Trees Regression (Bootstrap=True, n est.=100, min samples leaf=5)
– Gradient Boosting Regression (learning rate= 0.1, n estimators=200)

As throughout the optimization phase, we compare the test-set performance on our three distinct feature spaces. For all test runs, we compute the RMSE and R2 metrics (except for the Neural Network, where we unfortunately cannot provide the R2). These are our final results:

4.2 Conclusion

These are the five best-performing models, based on our evaluation:

1. Gradient Boosting Regressor
2. Extra Trees Regressor
3. Random Forest Regressor
4. KNN Regressor
5. SVM Regressor

Not too surprisingly perhaps, the ensembles outperformed all the other models, with the Gradient Boosting Regressor coming out on top. Ensembles are known to achieve high variance, while being tree-based and elements of randomness tend to aid them in preventing overfitting. Although the Neural Network performed quite poorly on the test set, it is important to note that the modification we tested here barely touches the surface of what deep learning has to offer. Throughout the optimization phase it already became evident that all models, regardless of their complexity, seem to do best with the full set of features. The PCA feature set consistently proved to be the worst option. This pattern seems to result from the fact the correlation between the different features and our target value is quite low, which makes any further sacrifice of explained variance quite detrimental to the overall model performance.

Although the data used for this research comprised a decent number of features, there are many other factors that may influence the price of Airbnb places in Amsterdam, and which might be investigated in further machine learning projects. For instance, one might focus on customers reviews and analyze them through sentiment analysis, using natural language processing technology. Geographical features such as, the nearest metro station, supermarket or police station might also provide valuable information that could generate predictive power using a create approach to machine learning.

Table 8. Final result table with top being worst performing model and bottom best performing model

	Full Features		Selected Features		PCA Features	
	RMSE	r-squared	RMSE	r-squared	RMSE	r-squared
Lasso Regressor	46.79	0.4648	49.04	0.4121	53.37	0.3037
KNN Regressor	44.02	0.5264	46.83	0.464	49.36	0.4044
SVM Regressor	45.07	0.5035	48.25	0.431	51.79	0.3443
Neural Network	45.66	-	47.24	-	52.68	-
Random Forest Regressor	43.28	0.5422	46.78	0.465	49.51	0.4007
Extra Trees Regressor	42.44	0.5598	46.13	0.4799	48.73	0.4197
Gradient Boosting Regressor	42.35	0.5616	45.59	0.4919	49.65	0.3974

References

1. J. A. Cruz and D. S. Wishart, "Applications of machine learning in cancer prediction and prognosis," Available at https://dl.acm.org/doi/abs/10.1145/3219819.3219830 (2006/01/01).
2. Ye, Peng, Qian, Julian, Chen, Jieying, Wu, Chen-hung, Zhou, Yitong, D. Mars, Spencer, Yang, Frank, Zhang, and Li, "Customized regression model for airbnb dynamic pricing," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, p. 932–940, 2018.
3. P. R. Kalehbasti, L. Nikolenko, and H. Rezaei, "Airbnb price prediction using machine learning and sentiment analysis," Available at https://arxiv.org/pdf/1907.12665.pdf (2019/29/07).
4. N. Pow, E. Janulewicz, and L. D. Liu, "Airbnb price prediction using machine learning and sentiment analysis," Available at http://rl.cs.mcgill.ca/comp598/fall2014 (2014/01/09).
5. T. Cai, K. Han, and H. Wu, "Melbourne airbnb price prediction," Available at http://cs229.stanford.edu/proj2019aut(2019/12/19).
6. Y. Ma, Z. Zhang, A. Ihler, and B. Pan, "Estimating warehouse rental price using machine learning techniques," Available at https://pdfs.semanticscholar.org/287f/aa34d15555728032db44a0d7dd73a5bd8886.pdf(2019/12/19).
7. R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science. in proceedings of the genetic and evolutionary computation conference 2016 (gecco '16)," *Association for Computing Machinery, New York*, p. 485–492, 2016.
8. J. Schaffer, "What not to multiply without necessity," pp. 644–664, 2015.
9. H. Motoda and H. Liu, "Feature selection, extraction and construction," Available at https://pdfs.semanticscholar.org (2002/01/01).
10. Sci-kitLearn, "Package used for machine learning algortihms," Available at http://scikit-learn.org/stable (2019/18/04).
11. J. Schmidt-Hieber, "Nonparametric regression using deep neural networks with relu activation function," Available at https://arxiv.org/abs/1708.06633 (2019/16/04).

5 Appendix

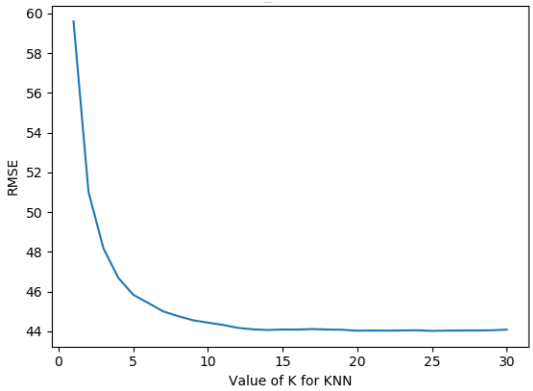
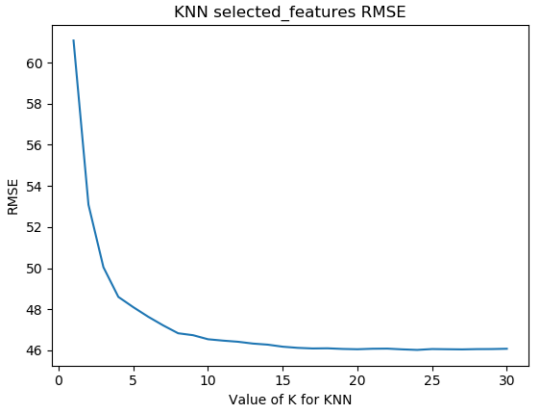
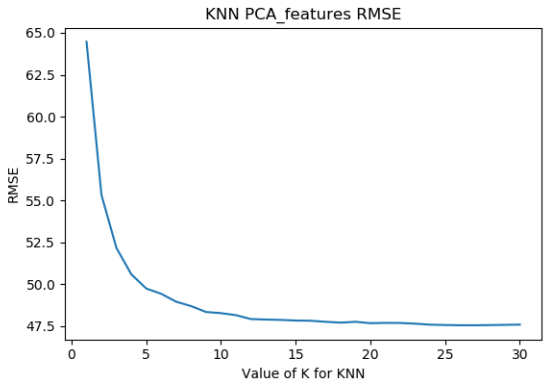
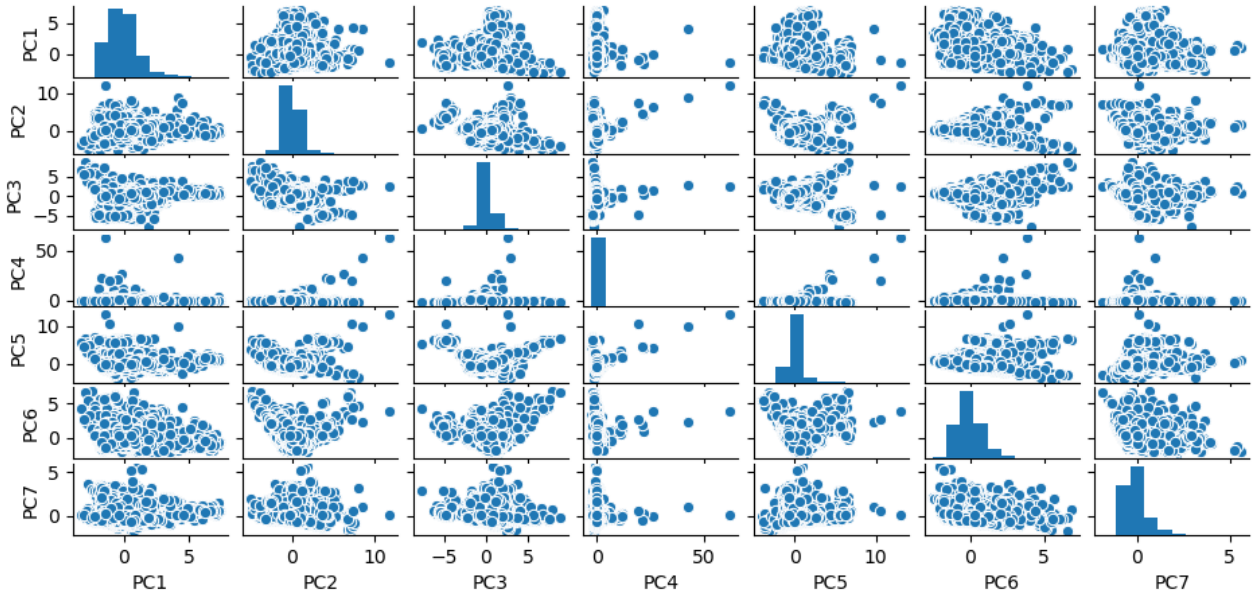
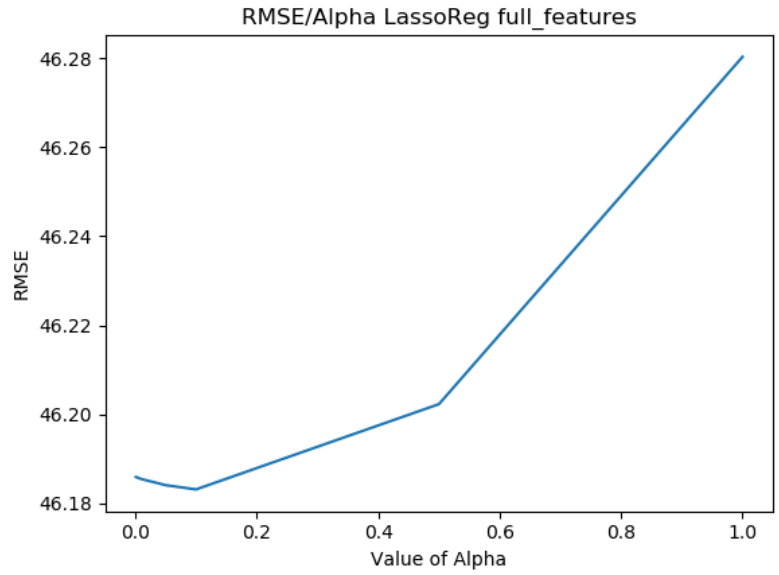


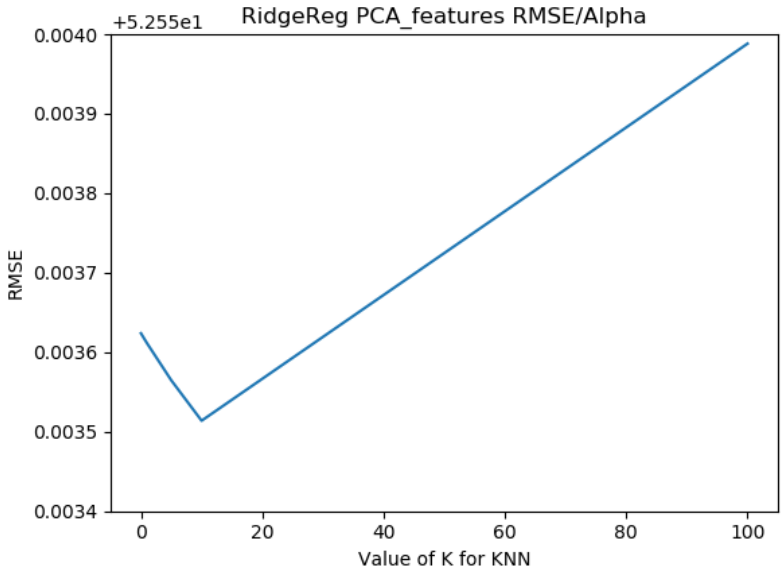
Fig. 3. kNN full feature RMSE




KNN PCA_features RMSE
RMSE
Value of K for KNN
65.0
62.5
60.0
57.5
55.0
52.5
50.0
47.5
0
5
10
15
20
25
30


PC1
PC2
PC3
PC4
PC5
PC6
PC7


RMSE/Alpha LassoReg full_features
46.28
46.26
46.24
46.22
46.20
46.18
RMSE
0.0
0.2
0.4
0.6
0.8
1.0
Value of Alpha


RidgeReg PCA_features RMSE/Alpha
+5.255e1
0.0040
0.0039
0.0038
0.0037
0.0036
0.0035
0.0034
RMSE
0
20
40
60
80
100
Value of K for KNN

Link to the source code: https://github.com/sebkeil/Group20-VU/tree/master/amsterdam-airbnb

Link to the all tables: https://github.com/sebkeil/Group20-VU/blob/master/amsterdam-airbnb/Model%20Optimization.xlsx