Defne Yaz Kılıç - 22102167
Boran Kılıç - 22103444

EEE485 Statistical Learning and Data Analytics
First Report

Introduction

The objective of this project is to develop machine learning algorithms to predict rental prices of Airbnb houses based on several features of the houses on the platform. Airbnb is a short-term rental platform, which brings together the travelers who are looking for a place to stay and the hosts who have a place to rent [1]. Predicting the rental prices of the houses based on their features is important both for hosts and guests. For instance, people looking for affordable housing with specific features may use machine learning algorithms to predict the prices. On the other hand, a host may use machine learning algorithms in order to determine the price of the house that they plan to rent.
In this project, three different machine learning algorithms are chosen to predict the Airbnb rental prices which are Linear Regression, Decision Tree and Neural Network. For the first report and demonstration Linear Regression and Decision Tree algorithms are implemented and the success of the methods are evaluated.
This report consists of dataset description, dataset pre-procesing, review of the chosen models and evaluation of the performance matrices of the models.

Dataset Description

The chosen dataset contains various features that can be linked to the prices of the houses. The 'Airbnb Price Dataset' can be accessed through here[2]. With 70111 data records and total number of 29 categorical and numerical columns for various features of the houses such as the city, the number of beds and bedrooms, amenities, cancellation policy, etc. The log_price feature is the natural logarithm of the actual prices which we plan to predict. Implementing regression algorithms with logarithm of the prices is beneficial in the sense that it helps linearizing the price variable as well as obtaining a more normally distributed price variable [3].

Methodology

1. Pre-processing Dataset :

The pre-processing was rather easy for our dataset since it was already structured. Firstly the columns are separated as numerical and categorical columns. Among them, there were 19 categorical and 10 numerical columns. The categorical columns contain string values such as True/False value of a feature, amenities list, city name, bed type etc. Some of the columns were unnecessary to include in our models which are 'id', 'description', 'name', 'thumbnail_url' and 'zipcode' since including them in the models would not make any logical sense.
Another problem was the empty entries. As can be seen from Fig. 1.1 several columns contained a lot of empty entries that can not be overlooked.

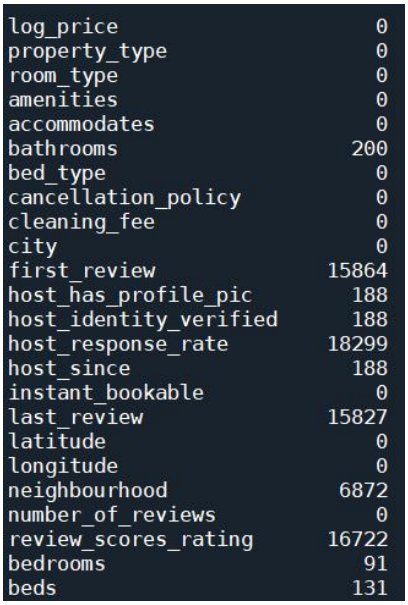


Fig. 1.1 Number of empty entries for each column

To overcome this problem, the columns containing significantly large empty entries first_review, last_review, host_response_rate and review_scores_rating are dropped. For the categorical columns that contain negligibly empty entries, firstly we converted all the categorical columns into numerical ones. The conversion from categorical to numerical was one of the most crucial parts of the preprocessing step. In order to assign consistent values to categorical entries we used different approaches for each column. For the entries that can be represented as binary, we assigned 1 to the 'True' values and 0 to 'False' values. Those columns were 'cleaning_fee', 'instant_bookable', 'host_has_profile_pic', 'host_identity_verified'. For the columns that have natural order which are 'cancellation_policy', 'room_type' values have replaced with integers in the ascending or descending order. For the columns that could not be represented as binary, which are 'city', 'property_type', 'bed_type' and 'neighbourhood', we used Target Encoding. Target encoding is replacing each category of a feature with that category's average price.

After converting every categorical column to numerical we have deleted the rows that has the value 0 for 'log_price' since it was not logical for an Airbnb house to have price 0. For some columns with few number of empty entries, such as 'host_has_profile_pic' , 'host_identity_verified', 'host_since' we deleted the corresponding rows.

For the 'neighbourhood' column even though it has significant number of empty entries instead of columns we deleted the corresponding rows for that empty entries. The reason for that is the 'neighbourhood' column shows significant correlation with the price of the Airbnb houses as it can be seen form Fig. 1.2. This correlation makes logical sense and can be beneficial for our regression models.

For non-binary categorical columns which has few number of empty values, 'beds', 'bedrooms', and 'bathrooms', the empty entries were filled with the mean value of that column.

For the feature 'amenities' we used the total number of different amenities of each datum as their corresponding numerical value.

After these processes we created a correlation matrix with the processed dataset to observe the correlation of each feature with the price.

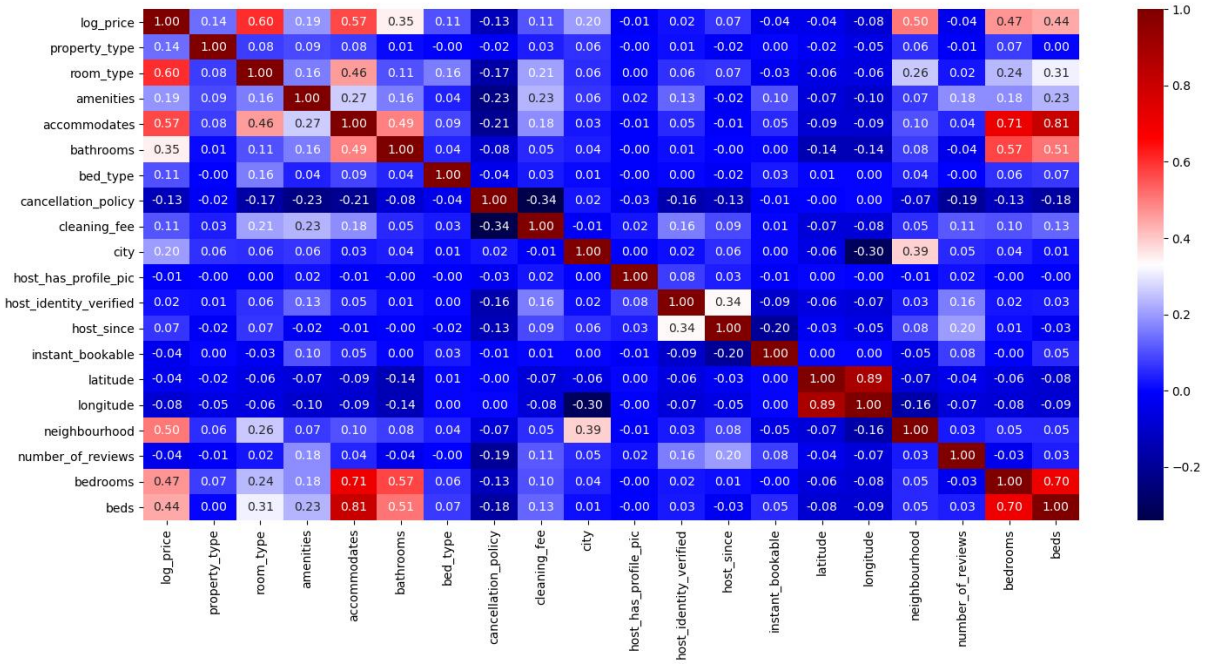


Fig. 1.2 Heatmap of the Processed Dataset

For the last part of the prepossessing step we dropped the features which shows very small even almost zero correlation with the Airbnb price.

A portion of the resulting processed dataset with logarithmic price and the new heatmap can be seen below.

Index	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy	cleaning_fee	city	neighbourhood	bedrooms	beds
0	5.01064	4.75885	2	9	3	1	4.79417	0	1	4.71934	5.0133	1	1
1	5.1299	4.75885	2	15	7	1	4.79417	0	1	4.71934	5.13127	3	3
2	4.97673	4.75885	2	19	5	1	4.79417	1	1	4.71934	4.6364	1	3
3	6.62007	4.79711	2	15	4	1	4.79417	2	1	5.17001	5.26301	2	2
4	4.74493	4.75885	2	12	2	1	4.79417	1	1	4.9868	4.88798	0	1
5	4.44265	4.75885	1	10	2	1	4.79417	0	1	5.17001	5.28294	1	1

Fig. 1.3 First five entries of the processed Dataset

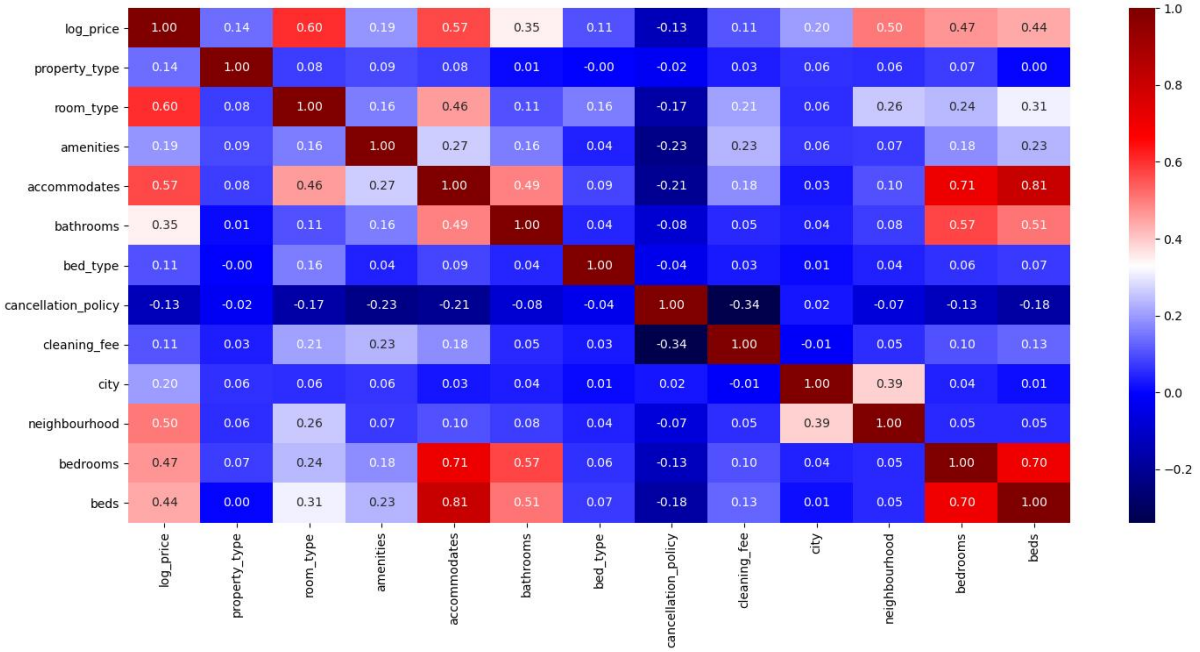


Fig. 1.4 Final Heatmap of the Processed Dataset

For the first phase we did not standardize our data since it would be unnecessary for Linear Regression and Decision Tree. However, for the final phase we plan to standardize the dataset before implementing Neural Network.

2. Linear Regression

As the first prediction method, linear regression is chosen. Although it is a simple model, due to its wide range of applications and easy-to-interpret nature, linear regression is one of the most fundamental methods of machine learning and data science. With linear regression, the data is fitted to a line in the hyperplane by taking the weighted sum of each data point as well as a bias term [4]. The coefficients that minimize the residual sum of squares are chosen as the weights. For multiple linear regression with p features, the predicted variable can be expressed as follows:

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j X_j \tag{1}$$

If $\hat{\beta} = [\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_p]^T$ and $X = [1, X_1, ..., X_p]^T$ then eq.1 can be expressed as:

$$\hat{Y} = X^T \hat{\beta} \tag{2}$$

Where the $\hat{\beta}$ vector can be found by the least squares method:

$$(\hat{\beta}_{RSS}) = \arg\min_{\beta \in R^{p+1}} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 \tag{3}$$

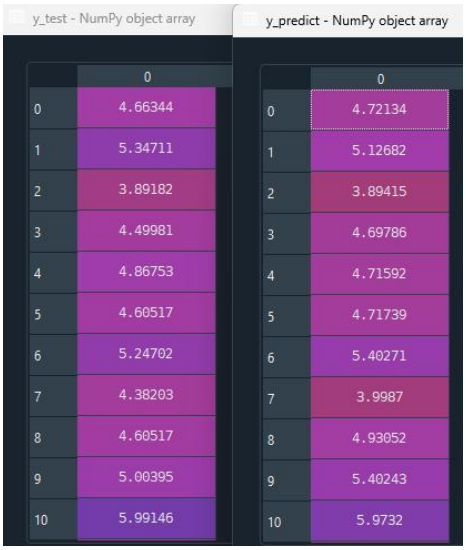
Assuming X has full column rank:

$$\hat{\beta}_{RSS} = X(X^T X)^{-1} X^T y \tag{4}$$

$$\hat{y} = X\hat{\beta}_{RSS} = X(X^T X)^{-1} X^T y \tag{5}$$

When this method was applied to Airbnb price prediction, first we dropped the 'log_price' column and labeled the remainder matrix as x and we labeled the 'log_price' column as y. After that, the dataset

was divided into two parts for training and testing purposes. The data was randomly shuffled and then split such that 80% was utilized for training and the remaining 20% for testing. Due to the strictness of the obtained linear relation, the fitted line cannot be easily modified according to the dataset requirements. However this method still gives consistent results with moderate error values. The training time is also very low. The performance metrics and example results of linear regression are given below:



y_test - NumPy object array

	0
0	4.66344
1	5.34711
2	3.89182
3	4.49981
4	4.86753
5	4.60517
6	5.24702
7	4.38203
8	4.60517
9	5.00395
10	5.99146

y_predict - NumPy object array

	0
0	4.72134
1	5.12682
2	3.89415
3	4.69786
4	4.71592
5	4.71739
6	5.40271
7	3.9987
8	4.93052
9	5.40243
10	5.9732

Fig. 2.1 Example Result of Linear Regression Predictions

The performance of the algorithm can be interpreted from the Fig. 2.2 which contains the performance metrics.

```
Training time of linear regression: 0.00400090217590332 seconds
Mean Absolute Error of Linear Regression   :  0.3145189409195166
Mean Squarred Error of Linear Regression     :  0.18313075127950232
Root Mean Squarred Error of Linear Regression:  0.42793778902955315
R2 Score of Linear Regression                :  0.6405583180106527
```

Fig. 2.2 Performance metrics of Linear Regression

The result are discussed in the results and conclusion part.

3. Decision Tree

The second method implemented within the scope of this project is the decision tree algorithm. A decision tree consists of leaf nodes, a root node and internal nodes with branches. Together, they form a hierarchical tree structure that can be utilized for both classification and regression tasks. As a non-parametric supervised learning algorithm, a decision tree is formed by recursively splitting the dataset into two subgroups with respect to features [5]. Since the aim of the project is to do price predictions, regression tree, a type of decision tree that works well for continuous value prediction can be used. In order to illustrate, a sample regression tree is given in Fig. 3.1.

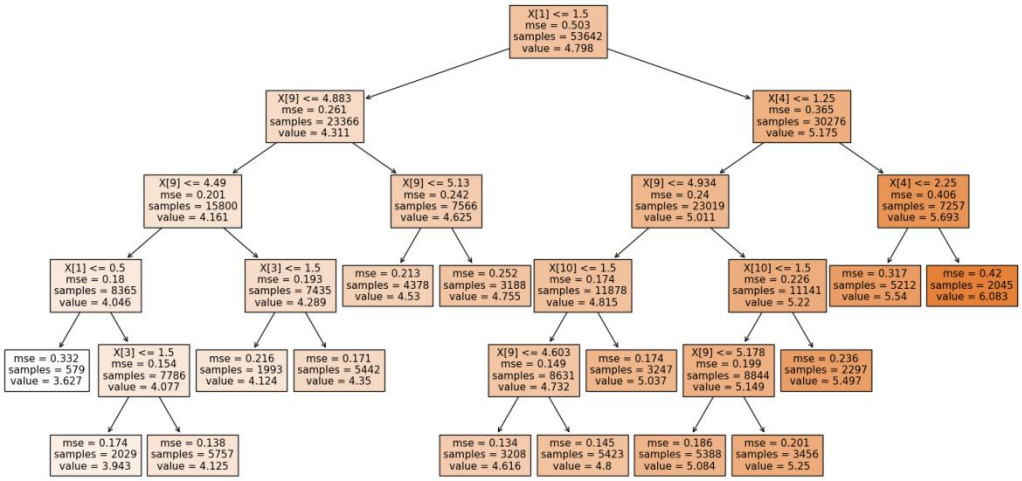


Fig. 3.1 Sample Decision Tree

The decision tree algorithm calculates thresholds for each feature on each node and chooses the "best" one and repeats this process until the tree is formed. The "best" threshold is the one that minimizes the residual sum of squares (RSS). The value of each node is the mean value of the data instances contained in that node. RSS is calculated as follows by the eq.6 where y_i denotes the mean value of the corresponding node:

$$RSS = \sum_{i=1}^{k} (\hat{y}_i - y_i)^2 \tag{6}$$

This way the dataset is splitted into subgroups therefore assigning every data to one of the leaf nodes in the end. However, if this algorithm is run without intervening the algorithm will eventually split every data instance in the training dataset to a unique leaf node which may create over-fitting. In order to prevent over-fitting two hyperparameters are defined. One of them is maximum depth which is the maximum distance of any leaf node to the root node i.e. the number of nodes until the farthest leaf node. This hyperparameter enables to restrict the size of the tree. The other hyperparameter is the minmum number of samples on a node, when a node has that number of samples or lower on it after a split, the node does not split further therefore becomes a leaf node. Both parameters are important due to prevent over-fitting and reducing computation time. In order to decide on the hyperparameters we run an algorithm which assigns different values to each hyperparameter, trains the tree, makes predictions and calculates the mean squared error for each hyperparameter duo. After this algorithm is completed running we plotted the MSE w.r.t. max depth and min samples split and decided on the hyper parameters.

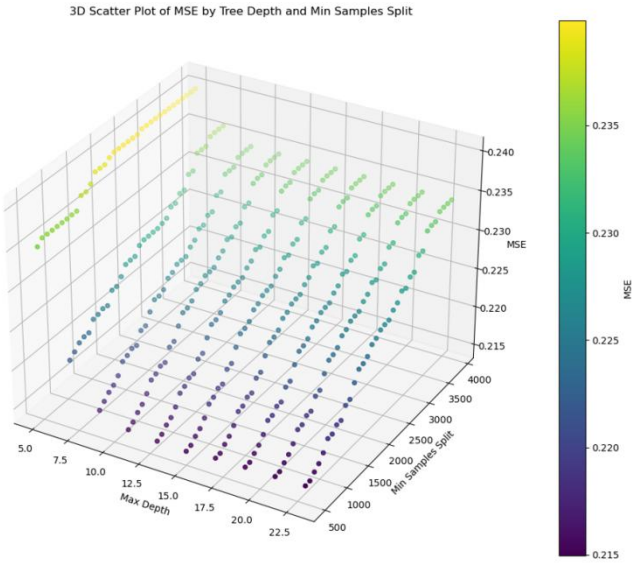


Fig. 3.2 3D Scatter Plot of MSE by Tree Depth and Min Samples Split

From Fig. 3.2 it can be seen that MSE does not decrease much after max depth = 10 for constant min samples split therefore max depth has chosen as 10 in order to minimize the error while also minimizing computation time. On the other hand, it can be seen that, as minimum sample size per split decreases MSE decreases as well. However, in order to avoid over-fitting and minimize the computation time the min samples split is chosen rather high but sufficient for the size of the dataset which is 500.

After the hyperprameters are determined we trained the decision tree with our training dataset which is 80% of the randomly shuffled dataset and made predictions with our test data set which is remaining 20% of the randomly shuffled dataset. The sample predictions are given below in Fig 3.3.

y_test - NumPy object array

	0
0	4.66344
1	5.34711
2	3.89182
3	4.49981
4	4.86753
5	4.60517
6	5.24702
7	4.38203
8	4.60517
9	5.00395
10	5.99146

y_predict - NumPy object array

	0
0	4.72921
1	5.13841
2	3.93907
3	4.61026
4	4.7714
5	4.7714
6	5.2672
7	4.10052
8	4.8795
9	5.4724
10	6.25953

Fig. 3.3 Example Result of Decision Tree Predictions

The performance of the algorithm can be interpreted from the Fig. 3.4 which contains the performance metrics.

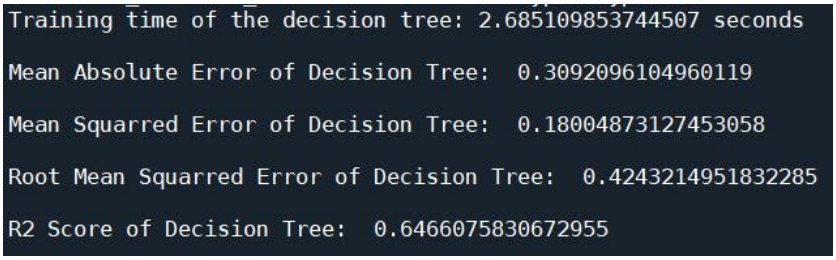


Fig. 3.4 Performance metrics of Decision Tree

The result are discussed in the results and conclusion part.

Results and Conclusion

From the results which are visible in the Figures 2.2 and 3.4 it can be seen that training time for Linear Regression was 0.004 seconds whereas for Decision Tree it was 2.68 seconds which is longer. R^2 score of Linear Regression was 0.6406 whereas for Decision Tree it was 0.6466. Higher R^2 score corresponds to better predictions which means the model with the highest R^2 score can be considered a more accurate model. In our case, R^2 scores were pretty close to each other which means both models have similar accuracy rate. On the other hand, the training time for Decision Tree is significantly larger then the training time of Linear Regression. In the light of these findings it can be concluded that linear regression model is a better choise for our problem because training time of the Decision Tree algorithm takes significantly longer with the same accuracy rate. Training time is an important factor while working with large datasets, since generally larger datasets may require more time to train.

This project was beneficial in the sense that understanding the math behind the regression algorithms and practising the implementation of regression algorithms in price prediction. For the final phase we plan to complete the project by completing the Neural Network algorithm and comparing the results with the algorithms that we already implemented.

References

[1] Andy, “Logarithmic transformation in linear regression models: Why & when,” DEV Community, [Online]. Available: https://dev.to/rokaandy/logarithmic-transformation-in-linear-regression-models-why-when-3a7c.[Accessed Apr. 20, 2024].

[2]“About airbnb: What it is and how it works - airbnb help center,” Airbnb, [Online]. Available: https://www.airbnb.com/help/article/2503 [Accessed Apr. 19, 2024].

[3] R. S. Rana, “Airbnb price dataset,” Kaggle, [Online]. Available: https://www.kaggle.com/datasets/rupindersinghrana/airbnb-price-dataset [Accessed Apr. 19, 2024].

[4] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning with Applications in Python*. Cham: Springer International Publishing, 2023.

[5] R. Gupta, "Regression Trees: Decision Tree for Regression Machine Learning," *Analytics Vidhya*, 2021. [Online]. Available: https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047 [Accessed: Apr. 18, 2024].

Appendices

Appendix A
Gannt Chart:

Fig. A.1: Gannt Chart

Appendix B
Distributions:

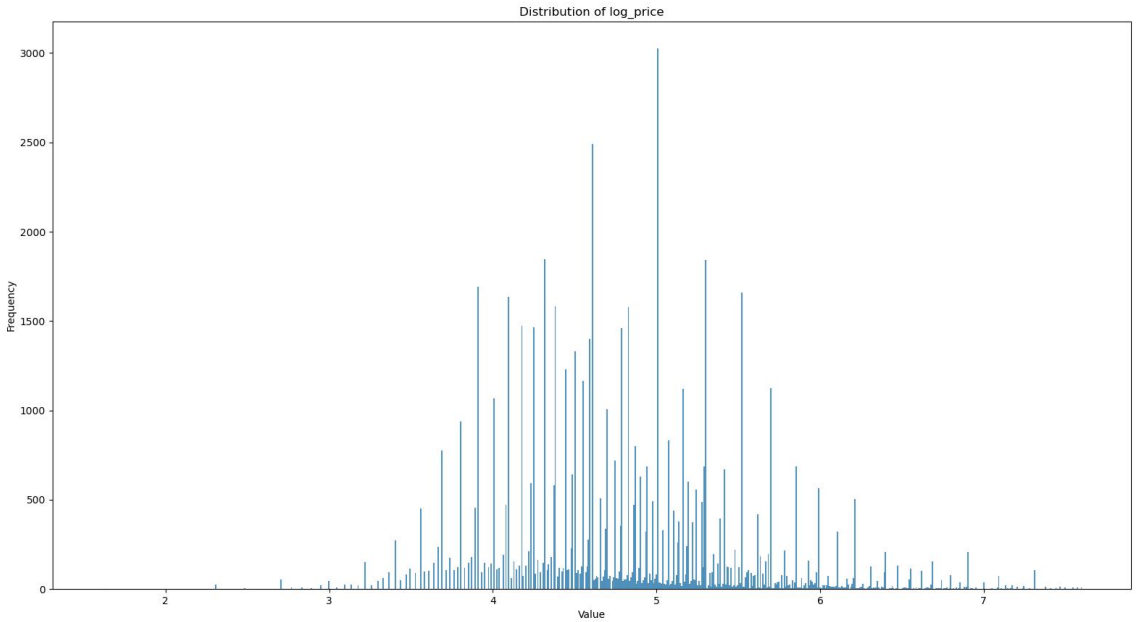


Fig. B.1: Distribution of the log price

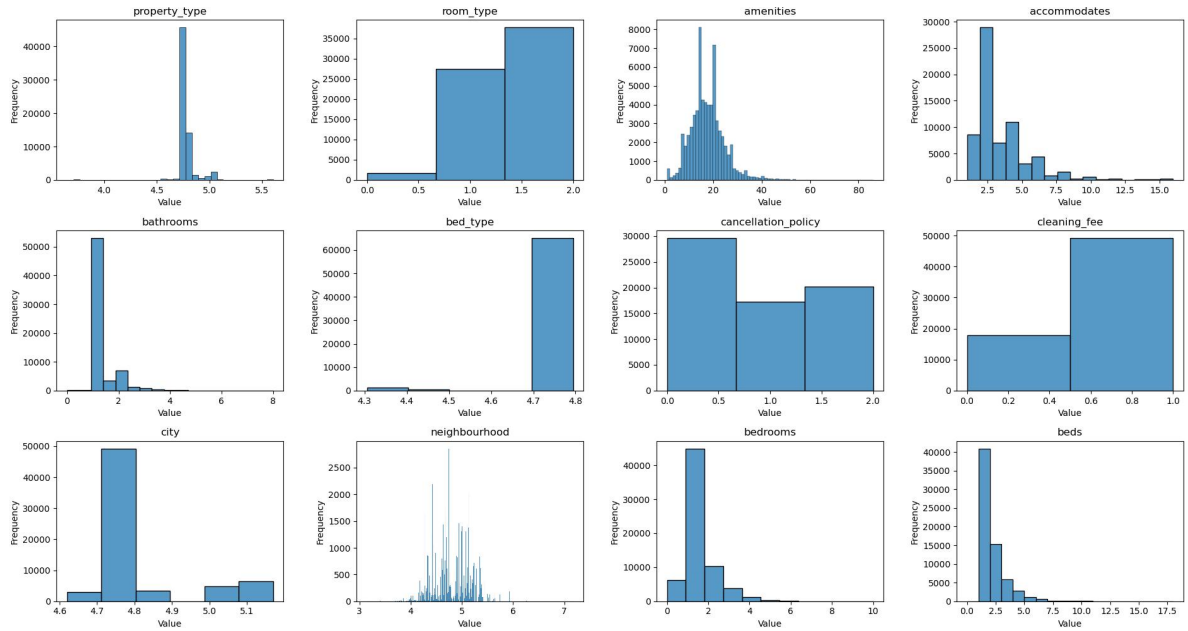


Fig. B.2: Distributions of the features

Appendix C

Pre-processing code:

preprocessing.py

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def target_encoding(new_data, columnname):
    mean_prices = new_data.groupby(columnname)['log_price'].mean()
    new_data[columnname] = new_data[columnname].replace(mean_prices)
    return new_data

data = pd.read_csv("Airbnb_Data.csv")

############################## non-usable columns dropped ##############################
new_data = data.drop(["id","description","name","thumbnail_url","zipcode"], axis='columns')
number_of_nans_per_column = new_data.isna().sum()
print('\n')
print(number_of_nans_per_column)

############################## columns with too many nan values dropped ################
new_data =
new_data.drop(['first_review','host_response_rate','last_review','review_scores_rating', ],axis = 1)

number_of_nans_per_column = new_data.isna().sum()
print('\n')
print(number_of_nans_per_column)

################## categorical columns converted to numerical ##########################
today = pd.to_datetime('today')
new_data['host_since'] = pd.to_datetime(new_data['host_since'])
new_data['host_since'] = (today - new_data['host_since']).dt.days

#categories with binary values
new_data['cleaning_fee'] = new_data['cleaning_fee'].replace({True: 1, False: 0})
new_data['instant_bookable'] = new_data['instant_bookable'].replace({'t': 1, 'f': 0})
new_data['host_has_profile_pic'] = new_data['host_has_profile_pic'].replace({'t': 1, 'f': 0})
new_data['host_identity_verified'] = new_data['host_identity_verified'].replace({'t': 1, 'f': 0})
#categories with natural order
new_data['cancellation_policy'] = new_data['cancellation_policy'].replace({'strict': 0,'super_strict_30':
0,

'super_strict_60': 0, 'moderate': 1,'flexible': 2})
new_data['room_type'] = new_data['room_type'].replace({'Entire home/apt': 2, 'Private room':
1,'Shared room': 0})

#target encoding for some categories
new_data = target_encoding(new_data,'city')
new_data = target_encoding(new_data,'property_type')
new_data = target_encoding(new_data,'bed_type')
new_data = target_encoding(new_data,"neighbourhood")

################################### NaN values are handled
#some rows with nan values for some features are dropped
```

```
new_data = new_data[new_data['log_price'] != 0]
new_data = new_data.dropna(subset=['host_has_profile_pic'])
new_data = new_data.dropna(subset=['host_identity_verified'])
new_data = new_data.dropna(subset=['host_since'])
new_data = new_data.dropna(subset=['neighbourhood'])

#nan values filled with mean values
new_data['bathrooms'].fillna(round(new_data["bathrooms"].mean()),inplace=True)
new_data['bedrooms'].fillna(round(new_data["bedrooms"].mean()),inplace=True)
new_data["beds"].fillna(round(new_data["beds"].mean()),inplace=True)

###new feature defined
amenities_count = []
for i in new_data["amenities"]:
    amenities_count.append(len(i.split(',')))

new_data["amenities"] = amenities_count

number_of_nans_per_column = new_data.isna().sum()
print("\nAfter\n" )
print(number_of_nans_per_column)

plt.figure(figsize = (40,30))
sns.heatmap(new_data.corr(), annot=True, fmt=".2f", cmap="seismic")
plt.subplots_adjust(left=0.2, bottom=0.3)
plt.show()

new_data = new_data.drop(['host_since','latitude', 'longitude',
                          'host_has_profile_pic', 'host_identity_verified',
                          'instant_bookable','number_of_reviews'], axis='columns')

number_of_nans_per_column = new_data.isna().sum()
print("\nAfter\n" )
print(number_of_nans_per_column)

plt.figure(figsize = (20,10))
sns.heatmap(new_data.corr(), annot=True, fmt=".2f", cmap="seismic")
plt.subplots_adjust(left=0.2, bottom=0.3)
plt.show()

new_data.to_csv('proccessed_airbnb_data.csv', index=False)
```

Separate file of functions for linear regression and decision tree:

functions.py:

```
import numpy as np

############################common functions#######################################
def r2_score (y_test, y_predict):
    nominator = 0
    denominator=0
    for i in range(len(y_test)):
        nominator = nominator + (y_test[i]-y_predict[i])**2
    for i in range(len(y_test)):
        denominator =    denominator + (y_test[i]- y_test.mean())**2
    return 1-(nominator/denominator)
```

```
def calc_RSS (y_test, y_predict):
    RSS = 0
    for i in range(len(y_test)):
        RSS = RSS + (y_test[i]-y_predict[i])**2
    return RSS

def mean_squared_error (y_test, y_predict):
    RSS = calc_RSS (y_test, y_predict)
    MSE = RSS/len(y_test)
    return MSE

def root_mean_squared_error(y_test,y_predict):
    MSE = mean_squared_error (y_test, y_predict)
    return np.sqrt(MSE)

def mean_absolute_error (y_test, y_predict):
    total = 0
    for i in range(len(y_test)):
        total = total + abs(y_test[i]-y_predict[i])
    MSE = total/len(y_test)
    return MSE

def train_test_split(x,y, seed, test_size):
    np.random.seed(seed)
    test_size = int(len(x) * test_size)
    x = x.to_numpy()

    indices = np.arange(len(x))
    np.random.shuffle(indices)

    x_shffl = x[indices]
    y_shffl = y[indices]

    x_train = x_shffl[test_size:]
    x_test = x_shffl[:test_size]
    y_train = y_shffl[test_size:]
    y_test = y_shffl[:test_size]
    return x_train, x_test, y_train, y_test

################################DECISION TREE#######################################

def fit_tree(x_train, y_train, min_samples, max_depth, depth=0):
    num_samples, num_features = x_train.shape
    if num_samples < min_samples or depth >= max_depth:
        return np.mean(y_train)

    best_ft, best_thr = best_split(x_train, y_train, num_features)
    if best_ft is None:
        return np.mean(y_train)

    left_idxs = x_train[:, best_ft] <= best_thr
    right_idxs = x_train[:, best_ft] > best_thr
    left_child = fit_tree(x_train[left_idxs], y_train[left_idxs], min_samples, max_depth, depth + 1)
```

```
    right_child = fit_tree(x_train[right_idxs], y_train[right_idxs], min_samples, max_depth, depth +
1)

    return best_ft, best_thr, left_child, right_child

def DT_RSS(child):
    RSS = 0
    mean = np.mean(child)
    RSS = np.sum((child - mean) ** 2)
    return RSS

def best_split(x_train, y_train, num_features):
    min_error = float('inf')
    best_ft = None
    best_thr = None
    for fidx in range(num_features):
        possible_thrs = np.unique(x_train[:, fidx])
        for th in possible_thrs:
            left = y_train[x_train[:, fidx] <= th]
            right = y_train[x_train[:, fidx] > th]
            error = DT_RSS(left) + DT_RSS(right)
            if error < min_error:
                min_error = error
                best_ft = fidx
                best_thr = th
    return best_ft, best_thr

def predict(tree, row):
    if type(tree) is not tuple:
        return tree
    else:
        feature, threshold, left_child, right_child = tree
        if row[feature] <= threshold:
            return predict(left_child, row)
        else:
            return predict(right_child, row)

def predict_tree(tree, x_test):
    if len(x_test.shape) > 1:
        return [predict(tree, row) for row in x_test]
    else:
        return predict(tree, x_test)
```

Linear Regression code:

linearregression.py

```
import pandas as pd
import numpy as np
from functions import *
import time

data = pd.read_csv('proccessed_airbnb_data.csv')

x = data.drop(["log_price"], axis=1)
y = data['log_price'].astype(float).values
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y,seed = 42, test_size = 0.2)


X_train = np.column_stack((np.ones(len(x_train)), x_train))
X_test =    np.column_stack((np.ones(len(x_test)), x_test))


X_train_transpose = np.transpose(X_train)
start_time = time.time()
beta = np.linalg.inv(X_train_transpose.dot(X_train)).dot(X_train_transpose).dot(y_train)
end_time = time.time()

training_time = end_time - start_time
print(f"Training time of linear regression: {training_time} seconds")

y_predict = X_test.dot(beta)

mse_lr = mean_squared_error(y_test, y_predict)
mae_lr = mean_absolute_error(y_test, y_predict)
rmse_lr = root_mean_squared_error(y_test, y_predict)
r2_lr = r2_score(y_test, y_predict)

print('\nMean Absolute Error of Linear Regression      : ', mae_lr)
print('\nMean Squarred Error of Linear Regression          : ', mse_lr)
print('\nRoot Mean Squarred Error of Linear Regression: ', rmse_lr)
print('\nR2 Score of Linear Regression                          : ', r2_lr)
```

Decision Tree code:

decisiontree.py

```
import pandas as pd
import numpy as np
import time
from functions import *

data = pd.read_csv('proccessed_airbnb_data.csv')

x = data.drop(["log_price"], axis=1)
y = data['log_price'].astype(float).values

x_train, x_test, y_train, y_test = train_test_split(x,y,seed = 42, test_size = 0.2)

start_time = time.time()
tree_model = fit_tree(x_train, y_train, 500, 10)
end_time = time.time()
training_time = end_time - start_time
print(f"Training time of the decision tree: {training_time} seconds")

y_predict = np.array(predict_tree(tree_model, x_test))
mae = mean_absolute_error(y_test, y_predict)
mse = mean_squared_error(y_test, y_predict)
rmse = np.sqrt(mean_squared_error(y_test, y_predict))
r2 = r2_score(y_test, y_predict)
```

```
print('\nMean Absolute Error of Decision Tree: ', mae)
print('\nMean Squarred Error of Decision Tree: ', mse)
print('\nRoot Mean Squarred Error of Decision Tree: ', rmse)
print('\nR2 Score of Decision Tree: ', r2)
```

Decision Tree hyperparameter selection code:

dthyperparameterselection.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from functions import *


data = pd.read_csv('proccessed_airbnb_data.csv')

x = data.drop(["log_price"], axis=1)
y = data['log_price'].astype(float).values

x_train, x_test, y_train, y_test = train_test_split(x,y,seed = 42, test_size = 0.2)

depths = []
samples_splits = []
mse_values = []


for depth in range(5, 25, 2):
    for sample_size in range(500, 4000, 100):
        tree_model = fit_tree(x_train, y_train, sample_size, depth)
        y_predict = predict_tree(tree_model, x_test)
        mse = mean_squared_error(y_test, y_predict)
        depths.append(depth)
        samples_splits.append(sample_size)
        mse_values.append(mse)

min_mse = np.min(mse_values)
min_index = np.argmin(mse_values)

print("Minimum MSE:", min_mse)
print("Achieved with max_depth =", depths[min_index], "and min_samples_split =",
samples_splits[min_index])

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')


depths = np.array(depths)
samples_splits = np.array(samples_splits)
mse_values = np.array(mse_values)


scat = ax.scatter(depths, samples_splits, mse_values, c=mse_values, cmap='viridis')

ax.set_xlabel('Max Depth')
ax.set_ylabel('Min Samples Split')
ax.set_zlabel('MSE')
```

```
cbar = fig.colorbar(scat, ax=ax, extend='neither', orientation='vertical')
cbar.set_label('MSE')

plt.title('3D Scatter Plot of MSE by Tree Depth and Min Samples Split')
plt.show()
```

Plotting the distributions code:
distributions.py

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('proccessed_airbnb_data.csv')

plt.figure(figsize=(6, 4))
sns.histplot(data['log_price'],bins = len(data['log_price'].unique()), kde=False)
plt.title("Distribution of log_price")
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(15, 10))
axes = axes.flatten()
for i, col in enumerate(data.columns[1:]):
    sns.histplot(data[col],bins = len(data[col].unique()), kde=False, ax=axes[i])
    axes[i].set_title(col)
    axes[i].set_xlabel('Value')
    axes[i].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```