Defne Yaz Kılıç - 22102167
Boran Kılıç - 22103444

EEE485 Statistical Learning and Data Analytics
Final 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, and for the final report Neural Network algorithm is implemented.
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 the 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.

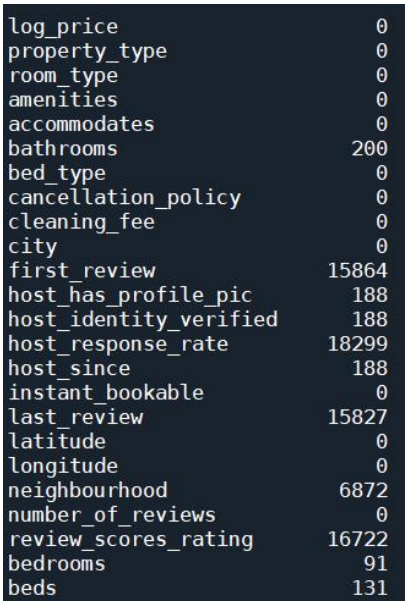

Fig. 1.1 Number of empty entries for each column

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 can be defined as replacing each category of a feature with that category's corresponding 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 deleting the whole column 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 every column is converted to a numerical one, we have deleted the outliers from the dataset using Interquartile Range (IQR) method. The Interquartile Range (IQR) is a measure of statistical dispersion, or how spread out the data points in a dataset are. It is calculated as the difference between the 75th percentile (the upper quartile, Q3) and the 25th percentile (the lower quartile, Q1) of the data. The formula is:

$$IQR = Q3 - Q1$$

The IQR is used to identify outliers in our dataset. Any data point that lies more than 1.5 times the IQR below Q1 or above Q3 is considered an outlier.

Specifically, the ranges for outliers are defined as:

$$\textit{Lower Bound: } Q1 - 1.5 \times IQR$$

$$\textit{Upper Bound: } Q3 + 1.5 \times IQR$$

For the next part of the prepossessing step we did feature selection. We created a correlation matrix with the processed dataset to observe the correlation of each feature with the price. We dropped the features which shows very small even almost zero correlation with the Airbnb price.

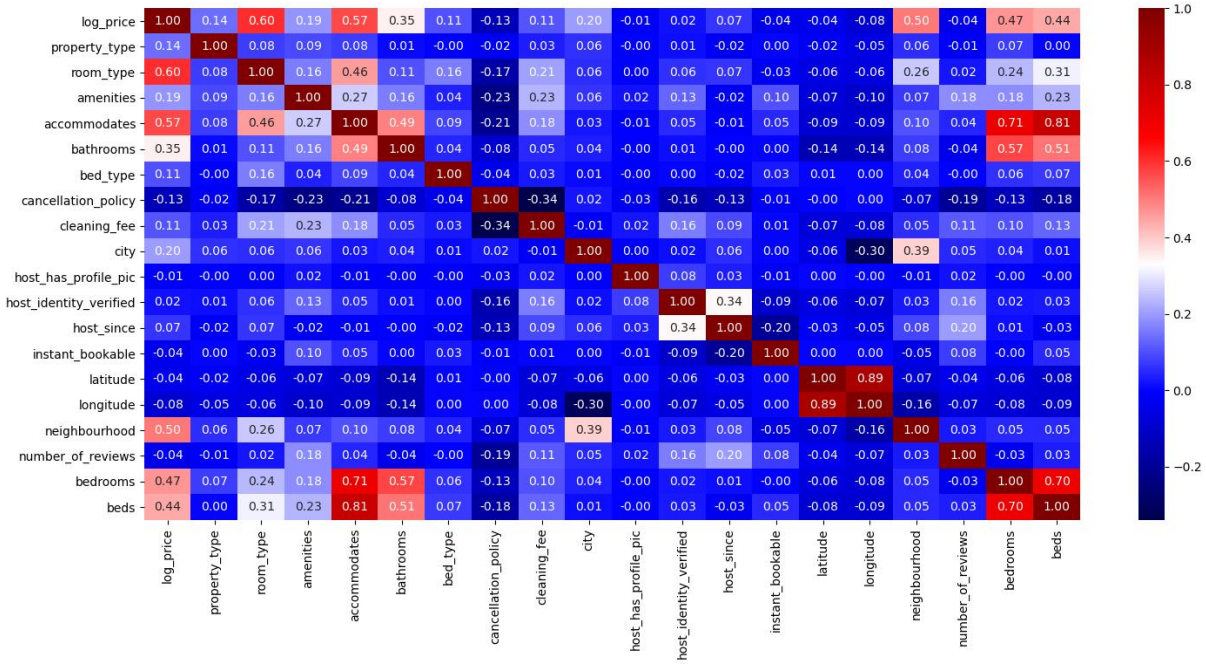

Fig. 1.2 Heatmap of the Processed Dataset

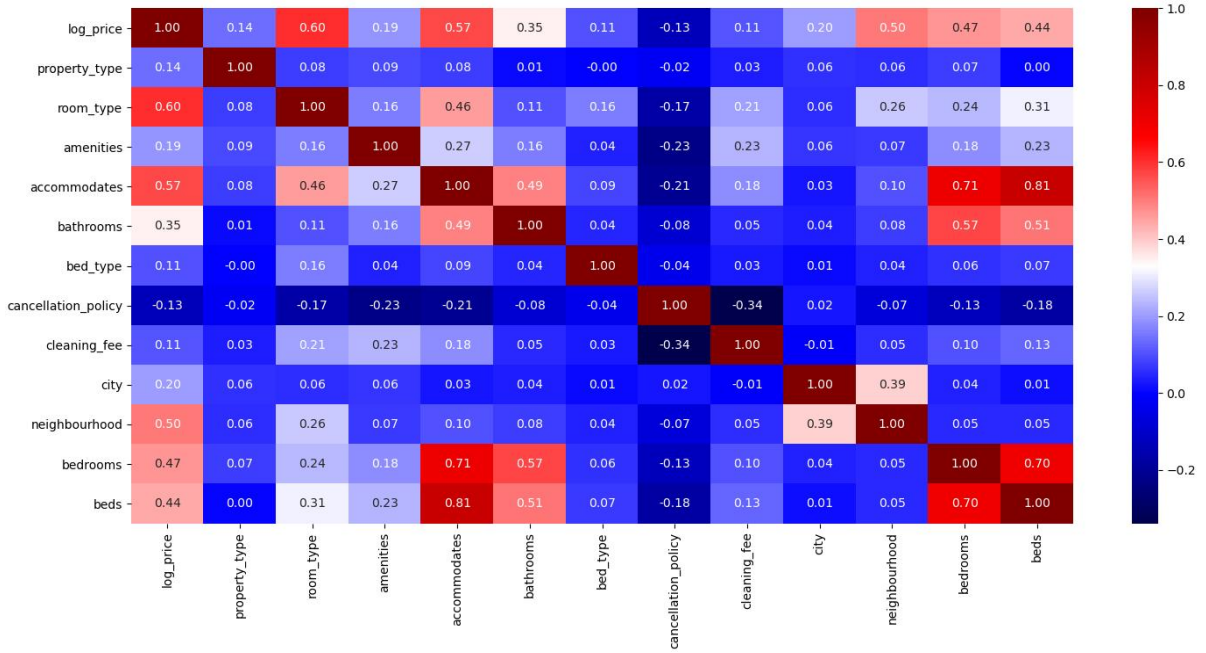

Fig. 1.3 Final Heatmap of the Processed Dataset after feature selection

Finally we standardized our dataset before implementing the machine learning algorithms. All the features are converted to zero mean, unit variance with following formula:

$$z = \frac{x_i - \mu}{\sigma}$$

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

log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy	cleaning_fee	city	neighbourhood	bedrooms	beds
0.387582	-0.24189	0.858704	-1.35842	-0.0203083	-0.394326	0.171965	-1.02122	0.604245	-0.438914	0.62563	-0.28391	-0.568957
0.571398	-0.24189	0.858704	-0.347874	1.99258	-0.394326	0.171965	-1.02122	0.604245	-0.438914	0.96022	2.24359	1.18116
0.335331	-0.24189	0.858704	0.325824	0.986135	-0.394326	0.171965	0.154699	0.604245	-0.438914	-0.443272	-0.28391	1.18116
2.86815	0.180104	0.858704	-0.347874	0.482913	-0.394326	0.171965	1.33062	0.604245	2.64837	1.33383	0.979841	0.306103
-0.0219363	-0.24189	0.858704	-0.853148	-0.52353	-0.394326	0.171965	0.154699	0.604245	1.39327	0.270238	-1.54766	-0.568957
-0.487831	-0.24189	-0.96661	-1.19	-0.52353	-0.394326	0.171965	-1.02122	0.604245	2.64837	1.39037	-0.28391	-0.568957
0.0436592	2.71034	0.858704	1.5048	-0.52353	-0.394326	0.171965	0.154699	0.604245	-0.430993	0.962631	-0.28391	-0.568957
0.0436592	0.180104	-0.96661	0.662674	-0.52353	-0.394326	0.171965	0.154699	0.604245	2.64837	1.03587	-0.28391	-0.568957
-0.237346	-0.24189	-0.96661	-0.347874	-0.52353	-0.394326	0.171965	-1.02122	0.604245	-0.438914	0.406548	-0.28391	-0.568957
0.387582	0.180104	0.858704	0.831098	0.482913	0.58312	0.171965	-1.02122	0.604245	-0.430993	1.63846	0.979841	0.306103

Fig. 1.4 A portion of the processed Dataset

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:

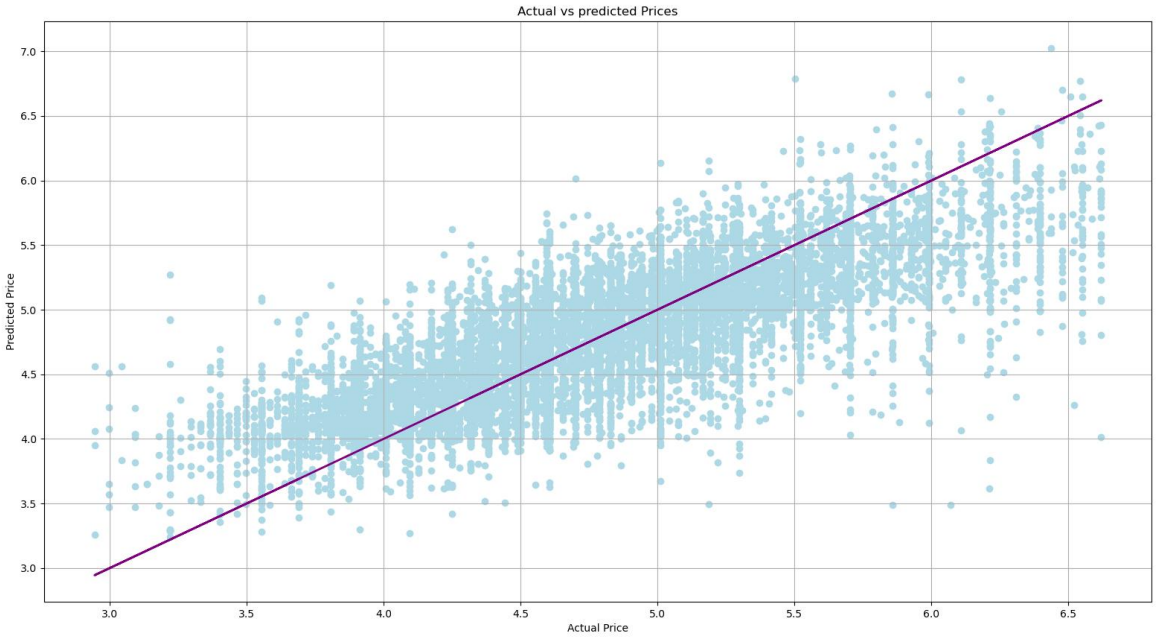

Fig. 2.1 Scatter Plot of Linear Regression Predictions

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

```
Training time of linear regression: 0.0019998550415039062 seconds
Mean Absolute Error of Linear Regression   :  0.29777141819102815
Mean Squarred Error of Linear Regression     :  0.1542213305520119
Root Mean Squarred Error of Linear Regression:  0.39271023739140276
R2 Score of Linear Regression                :  0.6345494399387908
```

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.

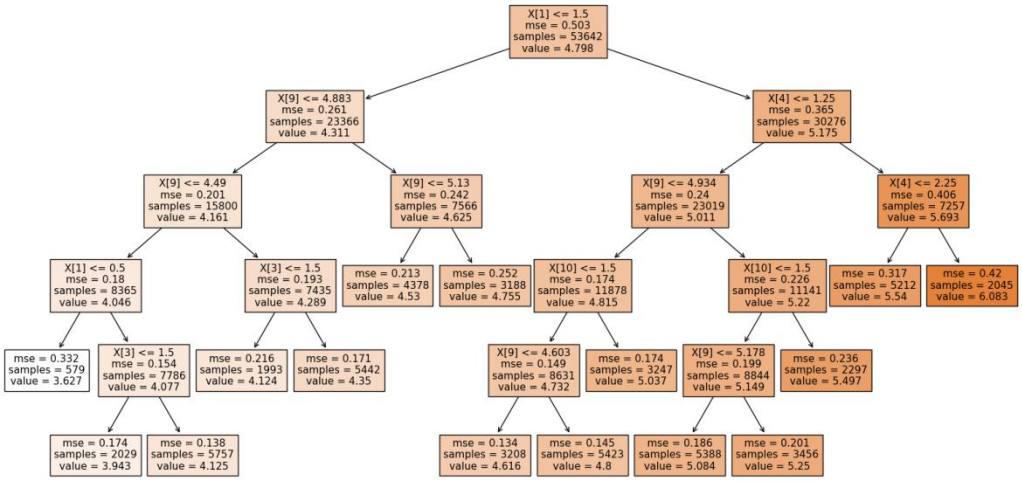

Fig. 3.1 Sample Decision Tree

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 split 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 hyper-parameters 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 hyper-parameter enables to restrict the size of the tree. The other hyper-parameter is the minimum 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 validate the hyper-parameters we did cross validation. We run an algorithm which assigns different values to each hyper-parameter, trains the tree, makes predictions with the validation set which is the 10% of the randomly shuffled dataset and calculates the mean squared error for each hyper-parameter duo. After this algorithm is completed running, we plotted the MSE with respect to max depth and min samples split and decided on the hyper-parameters.

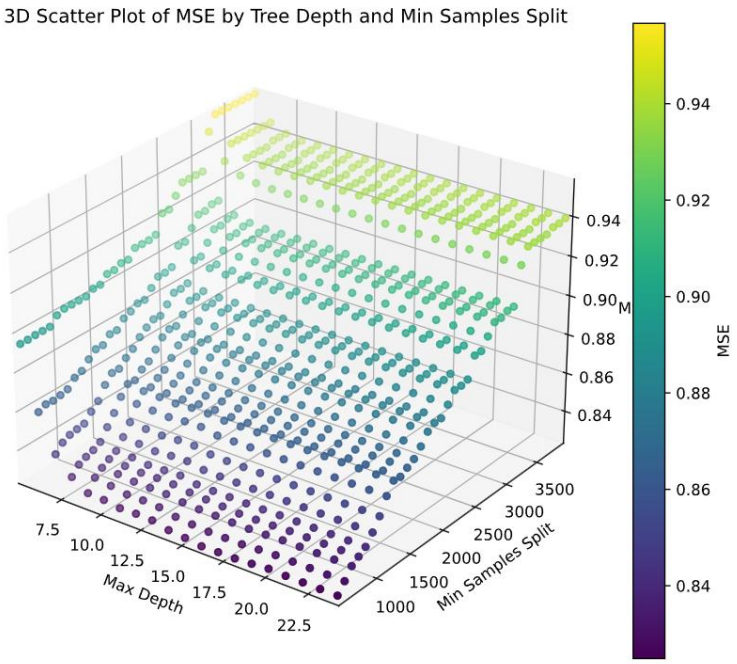

Fig. 3.2 3D Scatter Plot of MSE by Tree Depth and Min Samples Split

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 70% 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.

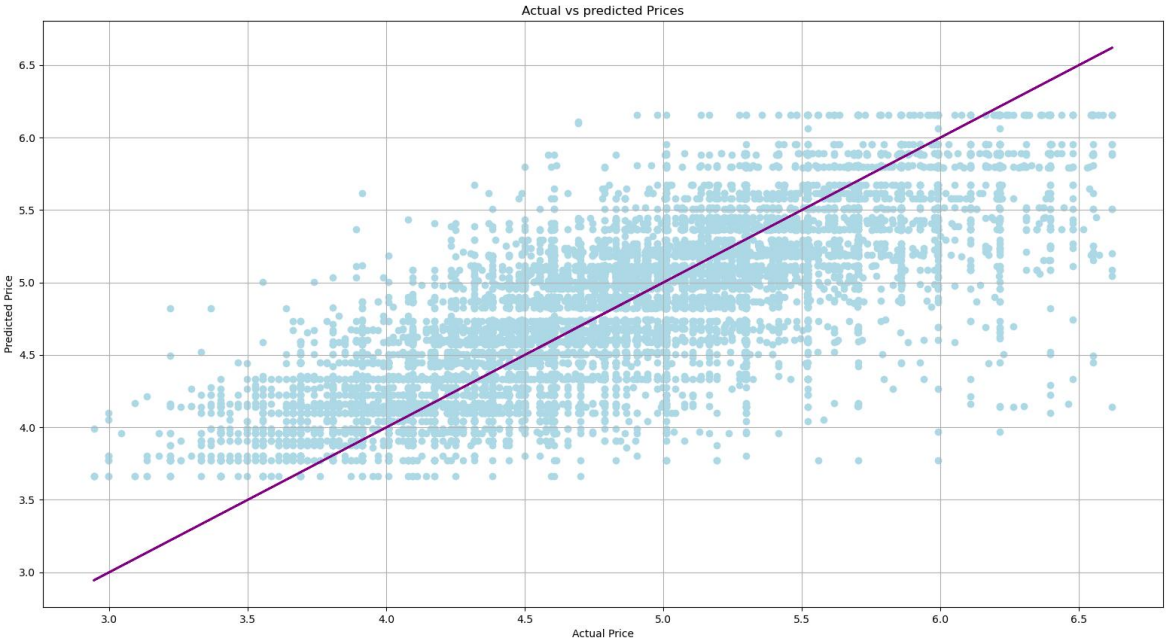

Fig. 3.3 Scatter Plot of Decision Tree Predictions

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

```
Training time of the decision tree: 2.1538193225860596 seconds

Mean Absolute Error of Decision Tree:  0.2914699138856216

Mean Squarred Error of Decision Tree:  0.15223791356288155

Root Mean Squarred Error of Decision Tree:  0.3901767721980404

R2 Score of Decision Tree:  0.6314183783138881
```

Fig. 3.4 Performance metrics of Decision Tree

The result are discussed in the results and conclusion part.

4. Neural Network

Neural Network is chosen as the last method due to its flexibility and versatility. A neural network similar to the brain structure, consist of perceptrons which work similar to neurons in brain as shown in Fig.4.1.

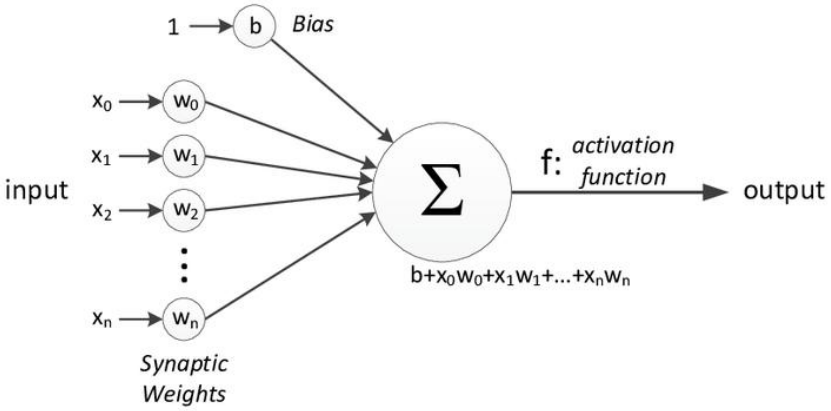

Fig. 4.1 A Perceptron [6]

Here the x_i's symbolizes each data feature with an additional constant bias term. These features are then multiplied with specific weights (w_i) and a sum is being done. The step function or activation function is the last step before obtaining the output, it acts as a switch which enables a neuron to be fired or not. The activation function also adds a non-linear behaviour to the system, distinguishes neural networks from ordinary linear regression models and adds to the capacity to perform more complex tasks [7]. In our project we used ReLU as the activation function due to its prevalence and unproblematic approach to so called the vanishing gradient problem. The combination of multiple perceptrons creates the neural network as shown in Fig.4.2.

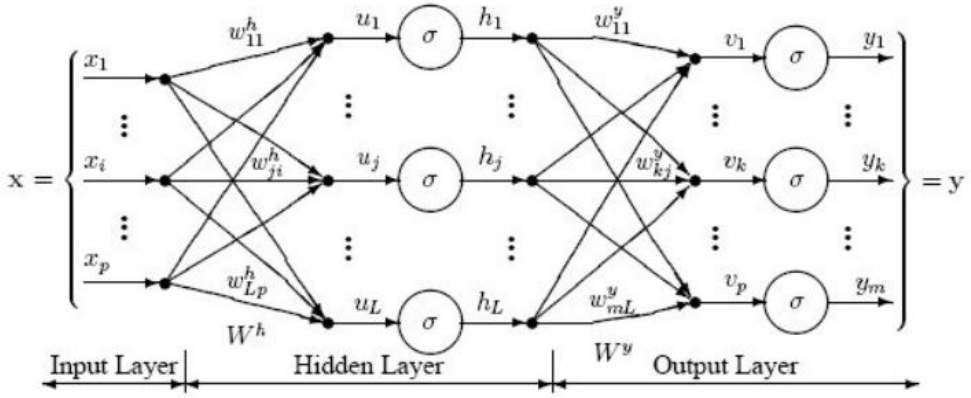

Fig. 4.2 A Neural Network [8]

For the Airbnb price prediction, we worked with 12 features corresponding to 12 inputs, one hidden layer and one output neuron, "price". As the initialization step, the weights were assigned randomly from standard random Gaussian distribution and zero bias. The convergence of weights and bias terms for both layers was obtained using the back-propagation algorithm. Starting from 1989, the back-propagation algorithm is highly popular among the training algorithms of neural networks. It is based on gradient descent-derived methods and the chain rule. The derivative of a function measures the sensitivity to change of the function value (output value) with respect to change in its argument (input value) [9]. Therefore, starting from the output, the parameters can be adjusted by looking at what changes they have on the output. Mathematically the loss function of the neural network can be minimized by decomposing its derivative with respect to its internal parameters using the chain rule. This way the influence of each parameter on the output can be observed and be tuned for minimum loss.

Given a neural network with layers indexed by ***l***, where each layer ***l*** computes an activation $a^{(l)}$ using an activation function σ based on the input $z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$, the back-propagation algorithm computes the gradient of the loss function ***L*** with respect to the weights ***w*** and biases ***b***. The loss function measures the difference between the predicted output $\hat{y}$ and the actual output y. Here are the steps followed for back-propagation:

Forward Pass, computing the output for each neuron from the input layer with initial parameter definitions.

$$z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$$
$$a^{(l)} = \sigma(z^{(l)}) \quad (7)$$

Backward Pass, the gradients of the loss ***L*** with respect to the weights $w^{(l)}$ and biases $b^{(l)}$ are calculated using the chain rule:

$$\frac{\partial L}{\partial w^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}}$$
$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial b^{(l)}} \quad (8)$$

Where,

- $\frac{\partial L}{\partial a^{(l)}}$ is the gradient of the loss function with respect to the output of layer ***l***.
- $\frac{\partial a^{(l)}}{\partial z^{(l)}}$ is the derivative of the activation function σ at layer ***l***.
- $\frac{\partial z^{(l)}}{\partial w^{(l)}}$ is simply $a^{(l-1)}$.
- $\frac{\partial z^{(l)}}{\partial b^{(l)}}$ is 1.

Update Step, using the computed gradients, the weights and biases are updated via the gradient descent method:

$$w^{(l)} := w^{(l)} - \eta\frac{\partial L}{\partial w^{(l)}}$$
$$b^{(l)} := b^{(l)} - \eta\frac{\partial L}{\partial b^{(l)}} \quad (9)$$

Where η is the learning rate that we determined using cross validation.

Before the training process there were 3 hyper-parameters to determine, number of epochs, number of neurons in the hidden layer and the learning rate. The number of epochs in a neural network algorithm refers to the number of times the entire training dataset is passed forward and backward through the neural network.
In order to choose these hyper-parameters we did cross validation with the validation set which is 10% of the randomly shuffled dataset. The results of the cross validations is as follows:

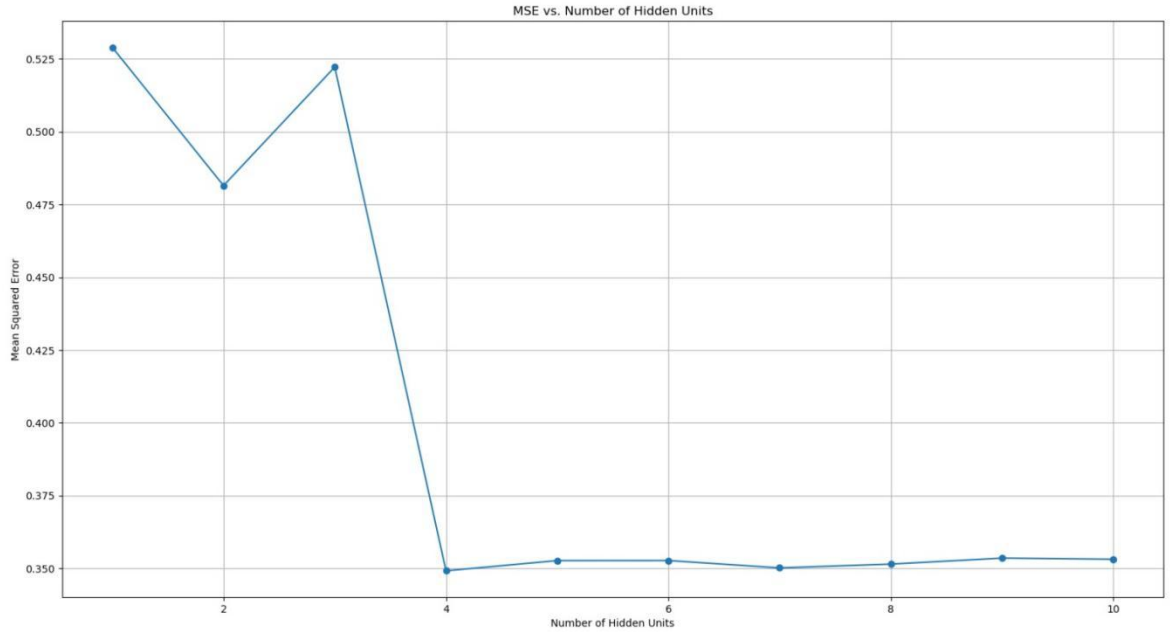

Fig. 4.3 Plot of MSE versus Number of Hidden Units

As it can be seen from Fig. 4.3, after 4 neurons per hidden layer, the mean square error stabilizes.Therefore considering both the computation time and the MSE, the number of neurons per hidden layer is chosen as 4.

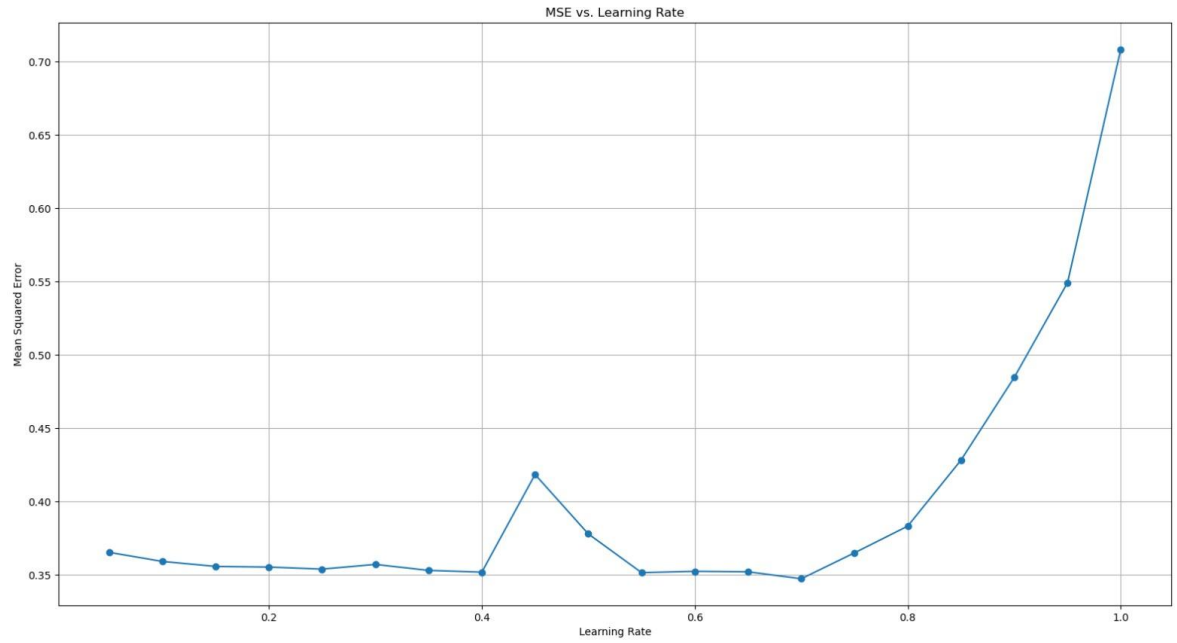

Fig. 4.4 Plot of MSE versus Number of Learning Rate

The second hyper-parameter, learning rate represents how fast the network converges in other words, “learns”. It changes between 1 and 0. While higher learning rate results in faster learning, it decreases tha capacity of generalization of the network. Considering its importance for obtaining consistent results, tuning the learning rate is the most crucial step of hyper-parameter selection. For its optimization , a similar approach used in number of hidden layers was followed. As can be

understood from Fig4.4, the optimum value for the learning rate is approximately 0.70 since it enables the fastest convergence while maintaining low MSE.

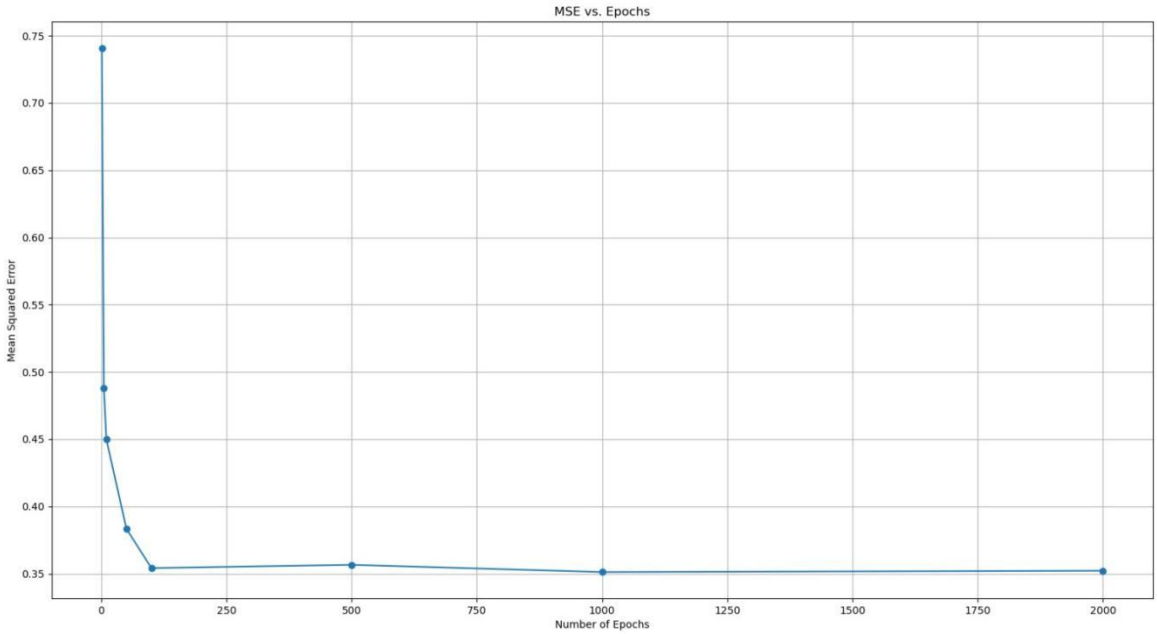

Fig. 4.5 Plot of MSE versus Number of Epochs

As for the number of epochs, the MSE reaches to its minimum value after 100 epochs as shown in Fig.4.5. However we decided to work with 1000 epochs considering that the increase in training time could be afforded and that the R2 score was significantly higher compared to 100 Epochs.

```
R2 Score of Neural Network epochs=100:  [0.59030845]
R2 Score of Neural Network epochs=1000:  [0.63638395]
```

Fig. 4.6 R2 score comparison epochs 100 vs 1000

After the training process, we obtained the predicted prices via forward propagation.

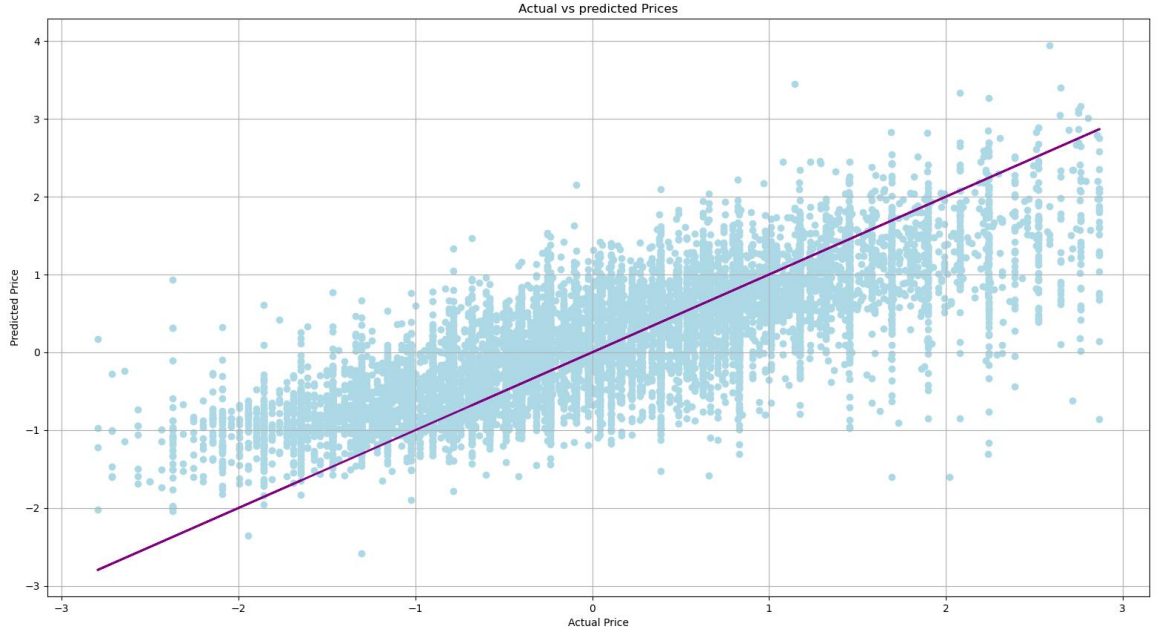

Fig. 4.7 Scatter Plot of Neural Network Predictions

In Fig4.7 it can be seen that the predicted prices were representing the nature of the actual prices close enough.

```
Training time of the neural network: 6.28859543800354 seconds

Mean Absolute Error of Neural Network:  [0.45269777]

Mean Squarred Error of Neural Network:  [0.35589551]

Root Mean Squarred Error of Neural Network:  [0.59656979]

R2 Score of Neural Network:  [0.64497962]
```

Fig. 4.8 Performance metrics of Decision Tree

The result are discussed in the results and conclusion part.

Results and Conclusion

From the results visible in Figures 2.2, 3.4, and 4.8, it can be observed that the training time for Linear Regression was approximately 0.002 seconds, which is significantly shorter compared to the Decision Tree at about 2.15 seconds and the Neural Network at roughly 6.29 seconds. The R2 scores for these models were quite close, with Linear Regression at 0.6345, Decision Tree at 0.6314, and Neural Network achieving the highest at 0.6449. These scores suggest that while the Neural Network offers slightly better prediction accuracy, its substantially longer training time may not justify the marginal increase in performance over the other models. Linear Regression, therefore, emerges as a more efficient choice, especially in scenarios where training speed and computational efficiency are crucial, considering its comparably high accuracy and minimal training time. Additionaly, the predictive performance of the regression methods are highly dependent on the chosen dataset this might be the reason for that the accuracy rates were quite similar for all three methods.

This project enhanced significantly our understanding of the mathematical principles underlying regression algorithms and our practice of the implementation of different regression algorithms in the context of machine learning especially for price prediction scenarios.

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].

[6] R. Diandaru, "A little about perceptrons and activation functions," Medium, [Online]. Available: https://rayendito.medium.com/a-little-about-perceptrons-and-activation-functions-aed19d672656. [Accessed: May 12, 2024].

[7] GeeksforGeeks, "Activation functions in neural networks," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/activation-functions-neural-networks/. [Accessed: May 12, 2024].

[8] "Introduction to Multilayer Perceptron Neural Networks," DTREG, [Online]. Available: https://www.dtreg.com/solution/multilayer-perceptron-neural-networks. [Accessed: May 12, 2024].

[9] S. Kostadinov, "Understanding backpropagation algorithm," Medium, [Online]. Available: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd. [Accessed: May 12, 2024].

Appendices

Appendix A
Gannt Chart:

Fig. A.1: Gannt Chart

Appendix B
Distributions before standardization and deleting the outliers:

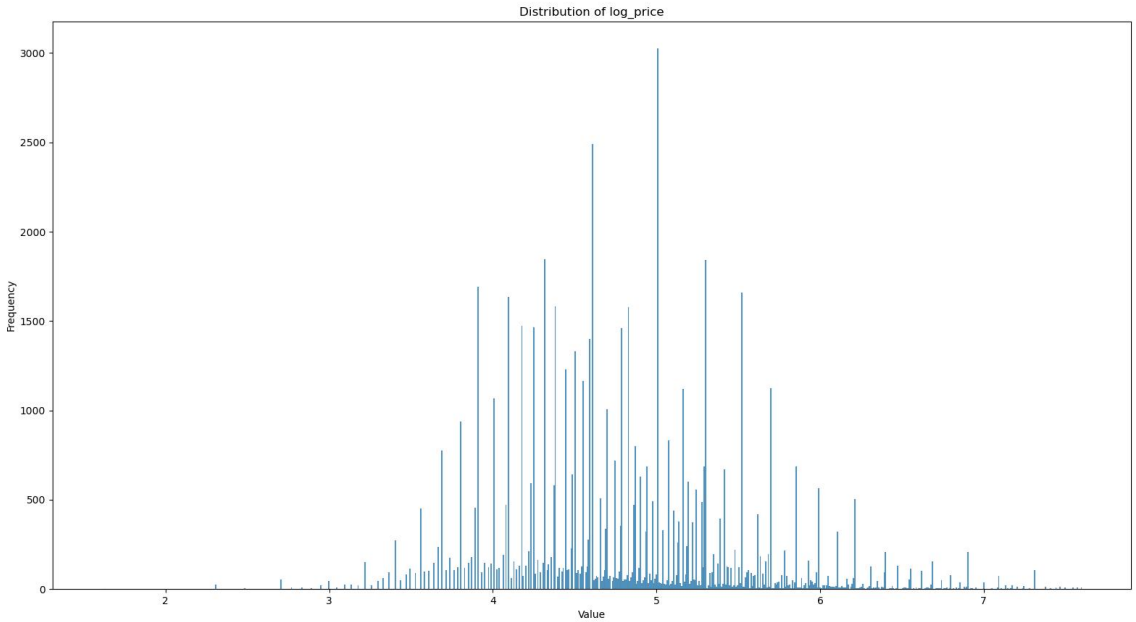

Fig. B.1: Distribution of the log price

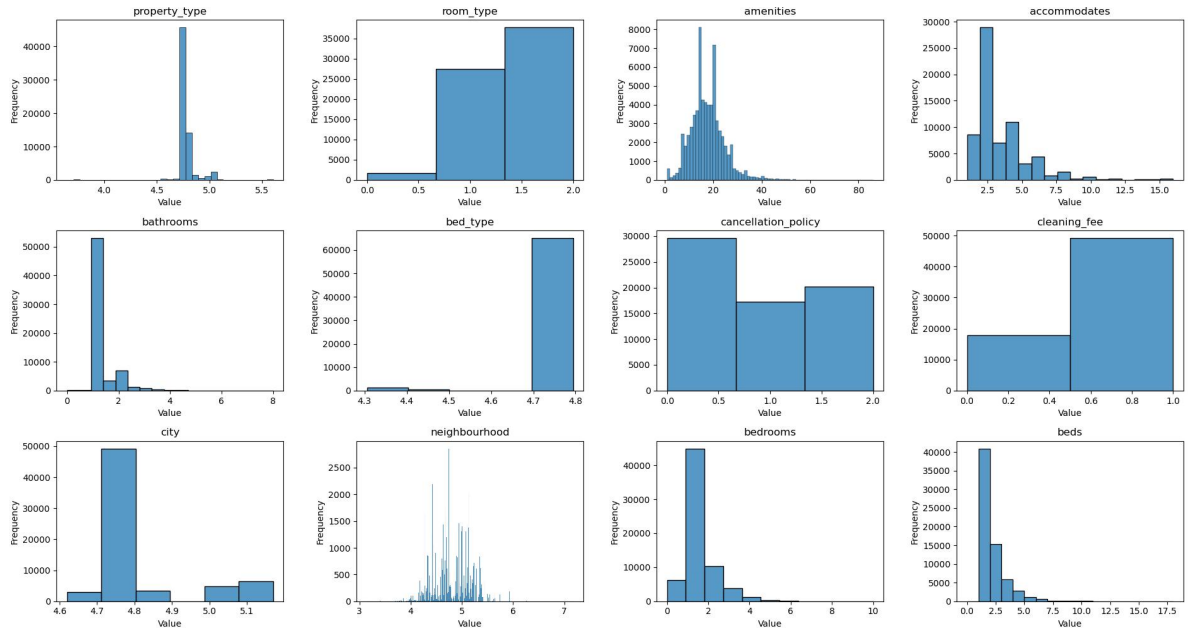

Fig. B.2: Distributions of the features

Distributions after standardization and deleting the outliers:

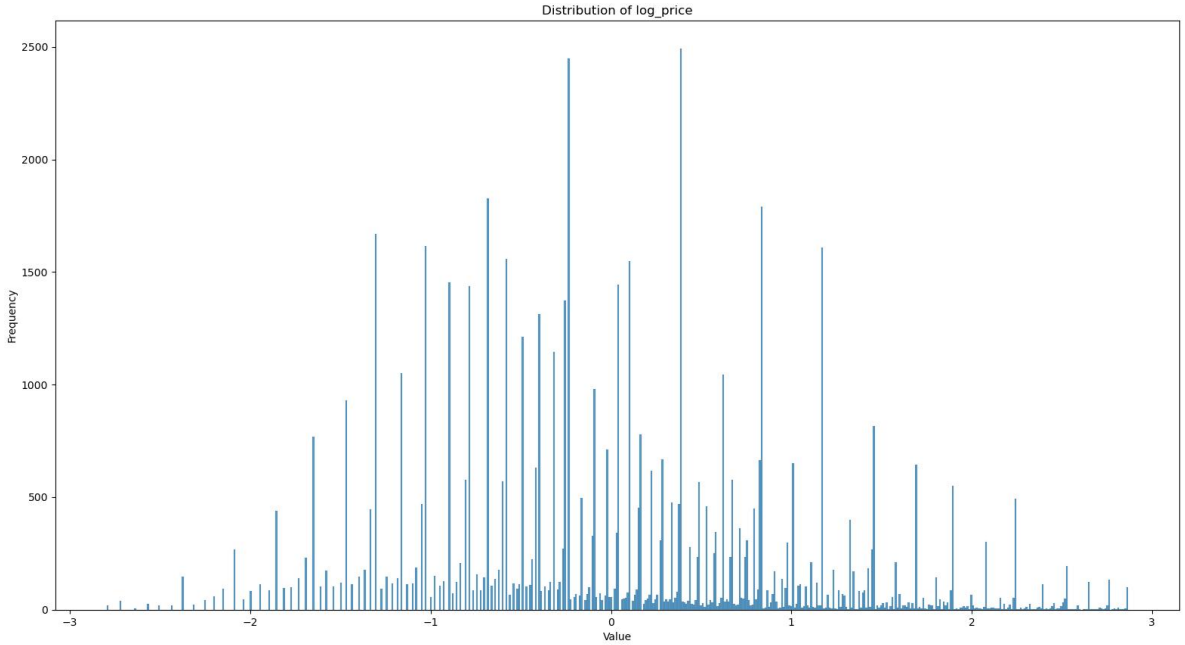

Fig. B.3: Distribution of the processed log price

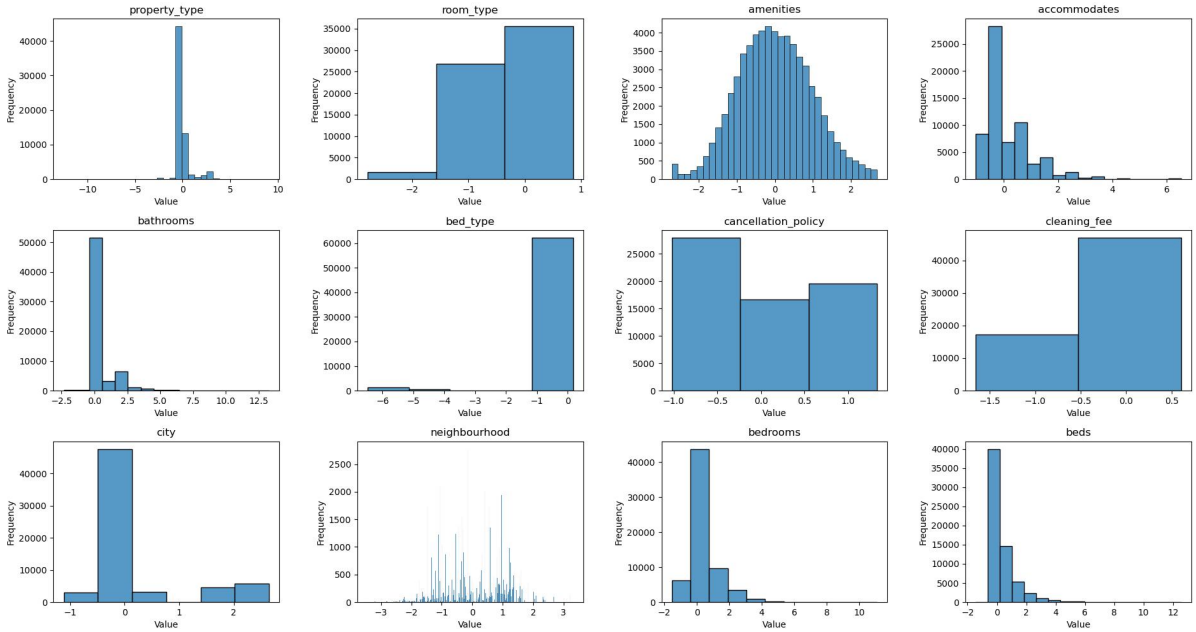

Fig. B.4: Distributions of the processed features

Appendix C

Pre-processing code:

preprocessing.py

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

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")

categorical_col = []
numerical_col = []
for column in data.columns:

    if data[column].dtypes == "float64" or data[column].dtypes == "int64":
        numerical_col.append(column)
    else:
        categorical_col.append(column)
print('\n')
print(len(categorical_col))
print(len(numerical_col))

############################## 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()
```

```
##################deleting outliers####################
columns_to_delete_outliers = ["log_price","amenities","neighbourhood"]

for column in columns_to_delete_outliers:
    Q1 = new_data[column].quantile(0.25)
    Q3 = new_data[column].quantile(0.75)
    IQR = Q3 - Q1

    condition = ~((new_data[column] < (Q1 - 1.5 * IQR)) | (new_data[column] > (Q3 + 1.5 * IQR)))

    new_data = new_data[condition]

############################ standardization #######################################

columns_to_standardize = new_data.columns.tolist()

for column in columns_to_standardize:
    col_mean = np.mean(new_data[column])
    col_std = np.std(new_data[column])
    new_data[column] = (new_data[column] - col_mean) / (col_std)

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

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()
```

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')
old_data = data

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()
```

Separate file of functions for linear regression and decision tree:
functions.py:

```
import numpy as np

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, validation_size=0):
    np.random.seed(seed)
    test_size = int(len(x) * test_size)
    validation_size = int(len(x) * validation_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[validation_size + test_size:]
    x_test = x_shffl[validation_size:validation_size + test_size]
    x_validation = x_shffl[:validation_size]

    y_train = y_shffl[validation_size + test_size:]
    y_test = y_shffl[validation_size:validation_size + test_size]
```

```
    y_validation = y_shffl[:validation_size]

    if validation_size == 0:
        return x_train, x_test, y_train, y_test
    else:
        return x_train, x_test, y_train, y_test, x_validation, y_validation


################################### 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)

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

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

##########################NEURAL NETWORK ################################

def relu(x):
    return np.maximum(0, x)

def relu_grad(x):
    return (x > 0).astype(float)

def initialize_weights(input_dim, output_dim):
    return np.random.randn(input_dim, output_dim) * np.sqrt(2 / input_dim)

def initialize_bias(output_dim):
    return np.zeros((1, output_dim))

def forward_pass(X, W1, b1, W2, b2):
    Z1 = np.dot(X, W1) + b1
    A1 = relu(Z1)
    Z2 = np.dot(A1, W2) + b2
    Y_hat = Z2
    return Y_hat, A1

def backward_pass(X, Y, Y_hat, A1, W1, W2):

    dZ2 = Y_hat - Y
    dW2 = np.dot(A1.T, dZ2) / len(Y)
    db2 = np.sum(dZ2, axis=0, keepdims=True) / len(Y)

    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * relu_grad(A1)
    dW1 = np.dot(X.T, dZ1) / len(Y)
    db1 = np.sum(dZ1, axis=0, keepdims=True) / len(Y)

    return dW1, db1, dW2, db2

def update_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, lr):
    W1 -= lr * dW1
    b1 -= lr * db1
    W2 -= lr * dW2
    b2 -= lr * db2
    return W1, b1, W2, b2

def neural_network(X_train, y_train, hidden_dim, output_dim, epochs, learning_rate):
    input_dim = X_train.shape[1]
    W1, b1 = initialize_weights(input_dim, hidden_dim), initialize_bias(hidden_dim)
    W2, b2 = initialize_weights(hidden_dim, output_dim), initialize_bias(output_dim)

    for epoch in range(epochs):
        Y_hat, A1 = forward_pass(X_train, W1, b1, W2, b2)
```

```
            # loss = np.mean((Y_hat - y_train)**2)
            # if epoch % 10 == 0:
            #         print(f'Epoch {epoch}, Loss: {loss}')
            dW1, db1, dW2, db2 = backward_pass(X_train, y_train, Y_hat, A1, W1, W2)
            W1, b1, W2, b2 = update_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, learning_rate)
    return W1, b1, W2, b2

def predict_neural(X, W1, b1, W2, b2):
    Y_hat, _ = forward_pass(X, W1, b1, W2, b2)
    return Y_hat
```

Linear Regression code:
linearregression.py

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

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

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

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)


plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_predict, color='lightblue')
plt.plot(y_test, y_test, color='purple', linewidth=2)
plt.title("Actual vs predicted Prices")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.grid(True)
plt.show()


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 matplotlib.pyplot as plt
import time
from functions import *


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

x = data.drop(["log_price"], axis=1)
y = data['log_price'].values.reshape(-1, 1)
x_train, x_test, y_train, y_test, x_validation, y_validation = train_test_split(x,y,seed = 42, test_size =
0.2, validation_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))

plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_predict, color='lightblue')
plt.plot(y_test, y_test, color='purple', linewidth=2)
plt.title("Actual vs predicted Prices")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.grid(True)
plt.show()

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 validation code:
dtvalidation.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'].values.reshape(-1, 1)

x_train, x_test, y_train, y_test, x_validation, y_validation = train_test_split(x,y,seed = 42, test_size =
0.2, validation_size=0.1)


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


for depth in range(5, 25, 1):
      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_validation)
            mse = mean_squared_error(y_validation, 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()
```

Neural Network code:
neuralnetwork.py

```
import numpy as np
```

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

data = pd.read_csv('proccessed_airbnb_data.csv')
X = data.drop(["log_price"], axis=1)
y = data['log_price'].values.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, seed = 42, test_size=0.2)

hidden_dim = 4
output_dim = 1
epochs = 1000
learning_rate = 0.75

start_time = time.time()
W1, b1, W2, b2 = neural_network(X_train,y_train, hidden_dim, output_dim, epochs, learning_rate)
end_time = time.time()
training_time = end_time - start_time
print('\n')
print(f"Training time of the decision tree: {training_time} seconds")

y_predict = predict_neural(X_test, W1, b1, W2, b2)

plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_predict, color='lightblue')
plt.plot(y_test, y_test, color='purple', linewidth=2)
plt.title("Actual vs predicted Prices")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.grid(True)
plt.show()


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

Neural Network validation code:
nnvalidation.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'].values.reshape(-1, 1)
```

```
x_train, x_test, y_train, y_test, x_validation, y_validation = train_test_split(x,y,seed = 42, test_size =
0.2, validation_size=0.1)

hidden_dim = 4
output_dim = 1
epochs = 1000
learning_rate = 0.65

hidden_dims = []
mse_values =[]
r2scr = []
epochs = 1000
learning_rate = 0.5
for hidden_dim in range(1,11):
    W1, b1, W2, b2 = neural_network(x_train,y_train, hidden_dim, output_dim, epochs,
learning_rate)
    y_predict = predict_neural(x_validation, W1, b1, W2, b2)
    hidden_dims.append(hidden_dim)
    mse = mean_squared_error(y_validation, y_predict)
    mse_values.append(mse)

plt.figure(figsize=(10, 5))
plt.plot(hidden_dims, mse_values, marker='o')
plt.title('MSE vs. Number of Hidden Units')
plt.xlabel('Number of Hidden Units')
plt.ylabel('Mean Squared Error')
plt.grid(True)
plt.show()

learning_rates = []
mse_values =[]
epochs = 1000
hidden_dim = 4

for learning_rate in np.arange(0.05, 1.05, 0.05):
    W1, b1, W2, b2 = neural_network(x_train,y_train, hidden_dim, output_dim, epochs,
learning_rate)
    y_predict = predict_neural(x_validation, W1, b1, W2, b2)
    learning_rates.append(learning_rate)
    mse = mean_squared_error(y_validation, y_predict)
    mse_values.append(mse)
plt.figure(figsize=(10, 5))
plt.plot(learning_rates, mse_values, marker='o')
plt.title('MSE vs. Learning Rate')
plt.xlabel('Learning Rate')
plt.ylabel('Mean Squared Error')
plt.grid(True)
plt.show()

epochslist = []
mse_values =[]

hidden_dim = 4
learning_rate = 0.65
```

```
listx = [1,5, 10,50, 100,500, 1000,2000]
for epochs in listx:
    W1, b1, W2, b2 = neural_network(x_train,y_train, hidden_dim, output_dim, epochs,
learning_rate)
    y_predict = predict_neural(x_validation, W1, b1, W2, b2)
    epochslist.append(epochs)
    mse = mean_squared_error(y_validation, y_predict)
    mse_values.append(mse)
plt.figure(figsize=(10, 5))
plt.plot(epochslist, mse_values, marker='o')
plt.title('MSE vs. Epochs')
plt.xlabel('Number of Epochs')
plt.ylabel('Mean Squared Error')
plt.grid(True)
plt.show()
```