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**

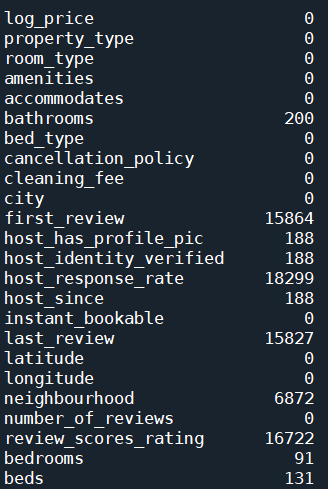
The chosen dataset contains various features that can be linked to the prices of the houses. The 'Airbnb Price Dataset' can be accessed through [here](https://www.kaggle.com/datasets/rupindersinghrana/airbnb-price-dataset)[2]. With 70111 data records and total number of 29 categorical and numerical columns 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*

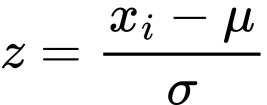
To overcome this problem, the columns containing significantly large empty entries first\_review, last\_review, host\_response\_rate and review\_scores\_rating are dropped. For the categorical columns that contain negligibly empty entries, firstly we converted all the categorical columns into numerical 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.

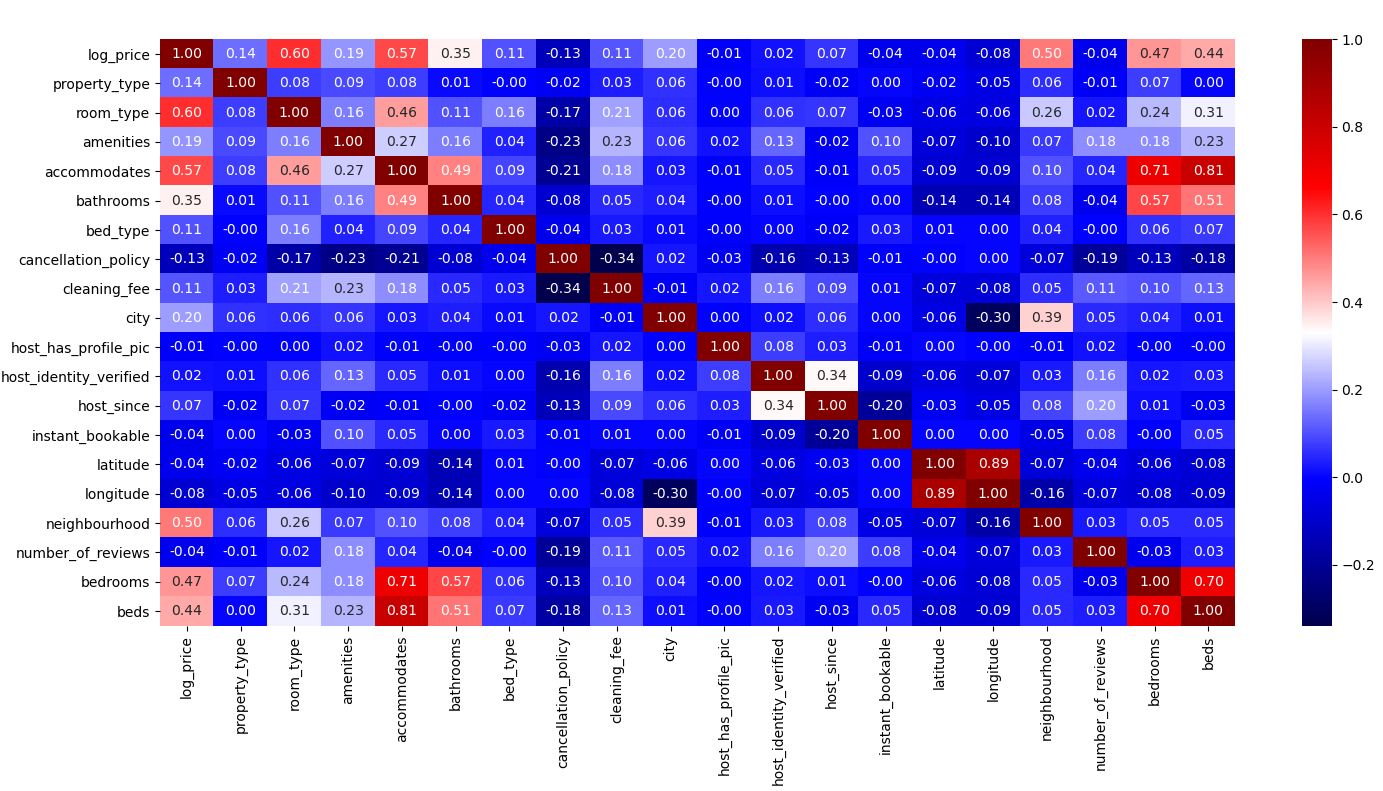
For the 'neighbourhood' column even though it has significant number of empty entries instead of deleting the whole column we deleted the corresponding rows for that empty entries. The reason for that is the 'neighbourhood' column shows significant correlation with the price of the Airbnb houses as it 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. Finally we standardized our dataset before implementing the machine learning algorithms. All the features are converted to zero mean, unit variance with following formula:



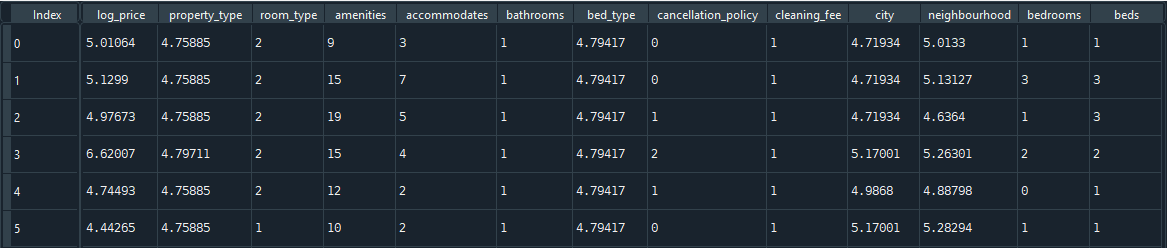
After these processes we created a correlation matrix with the processed dataset to observe the correlation of each feature with the price.



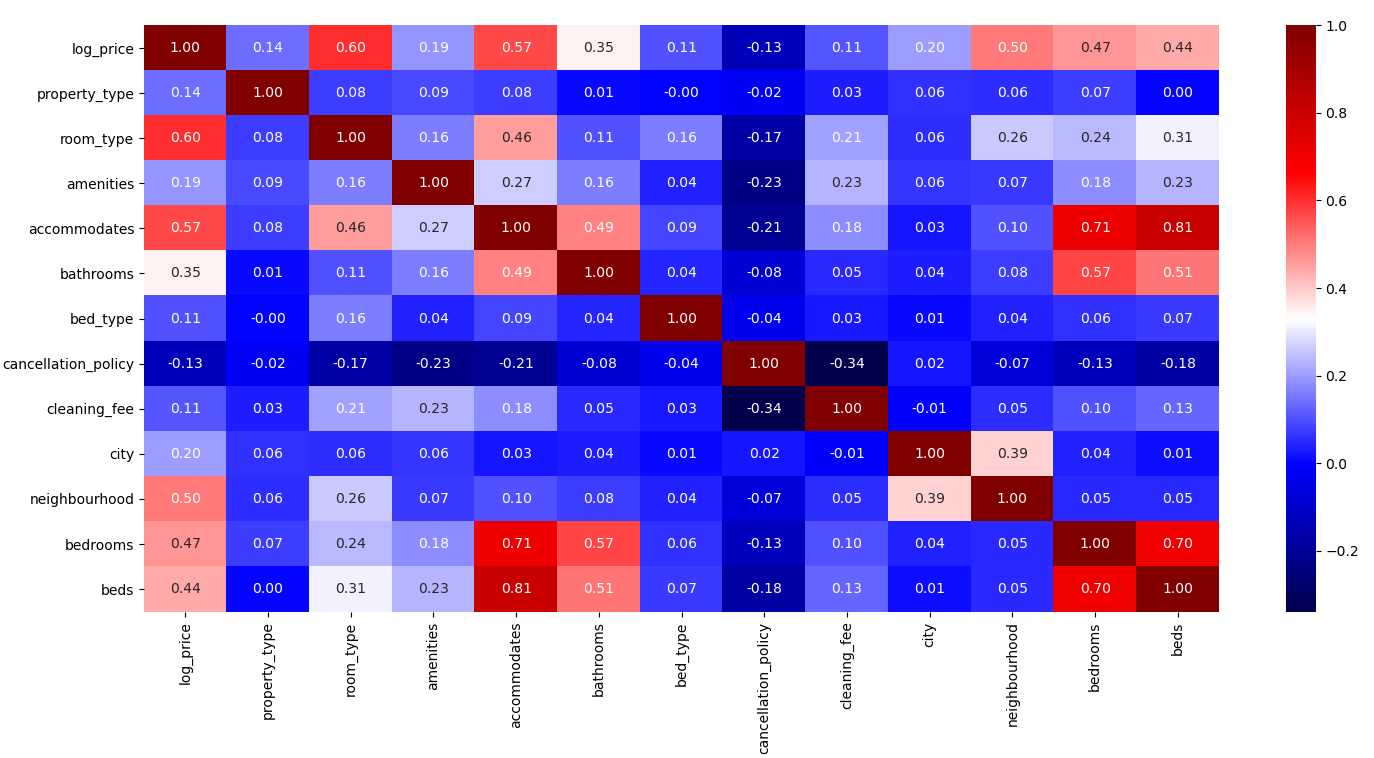
*Fig. 1.2 Heatmap of the Processed Dataset*

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.



*Fig. 1.3 First five entries of the processed Dataset*



*Fig. 1.4 Final Heatmap 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:

 (1)

If  and then eq.1 can be expressed as:

 (2)

Where the  vector can be found by the least squares method:

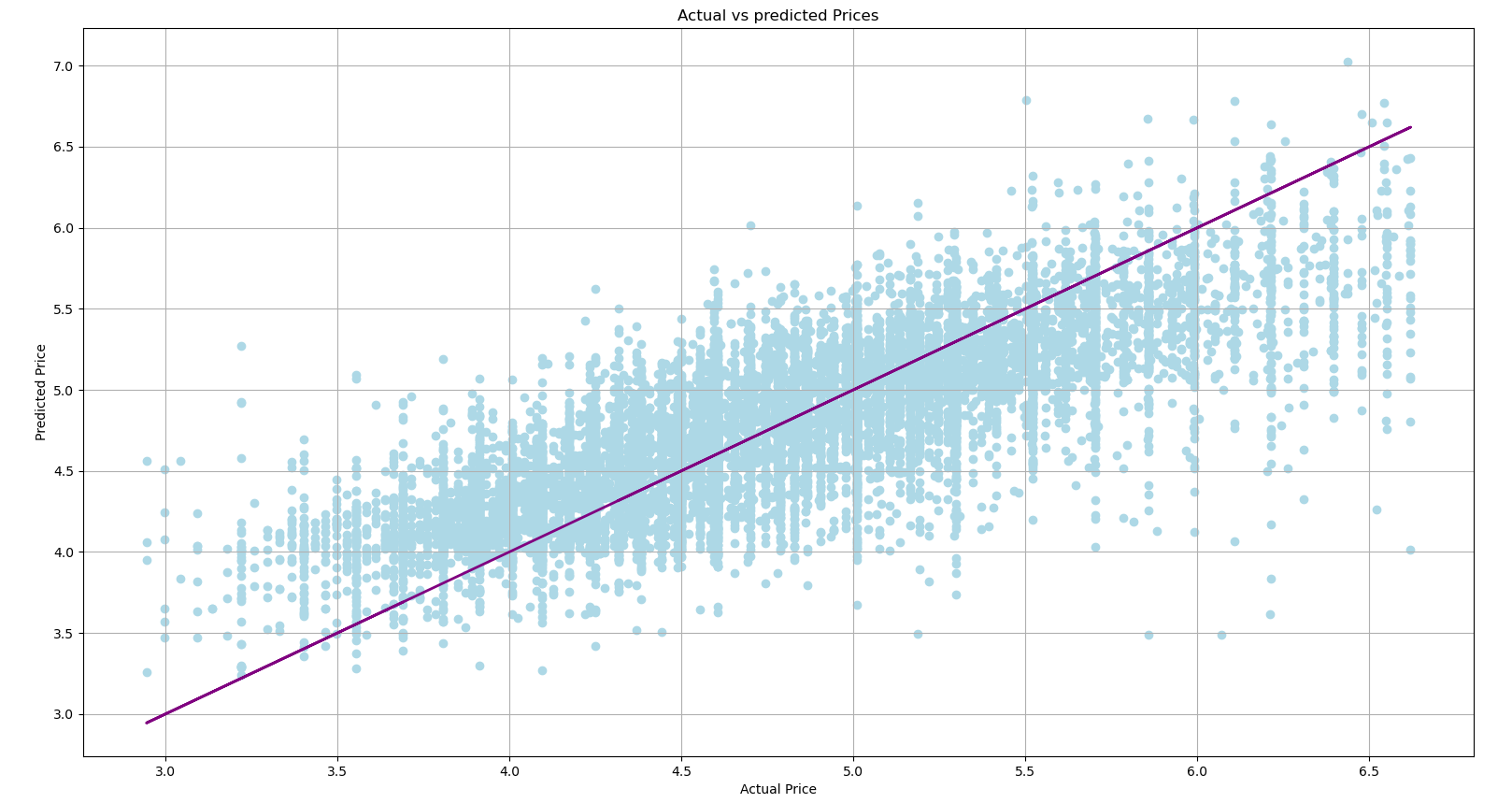
 (3)

Assuming X has full column rank:

 (4)

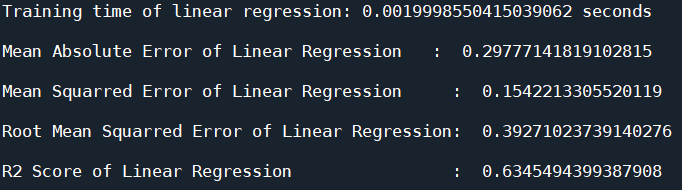
 (5)

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.

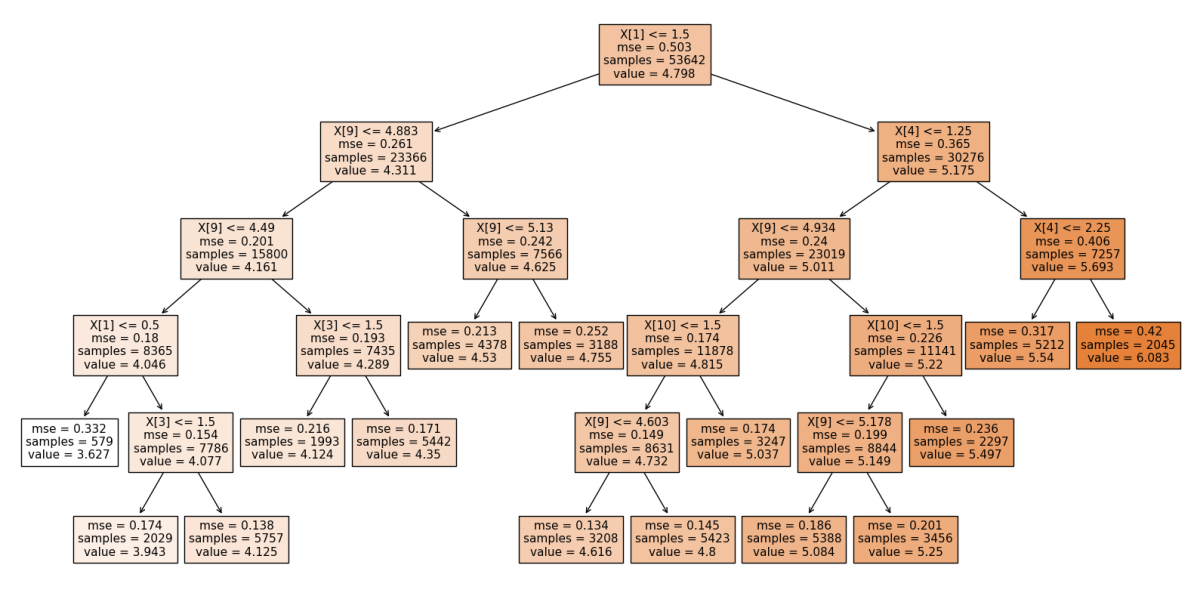


*Fig. 2.2 Performance metrics of Linear Regression*

The result are discussed in the results and conclusion part.

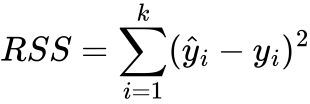
**3. Decision Tree**

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.



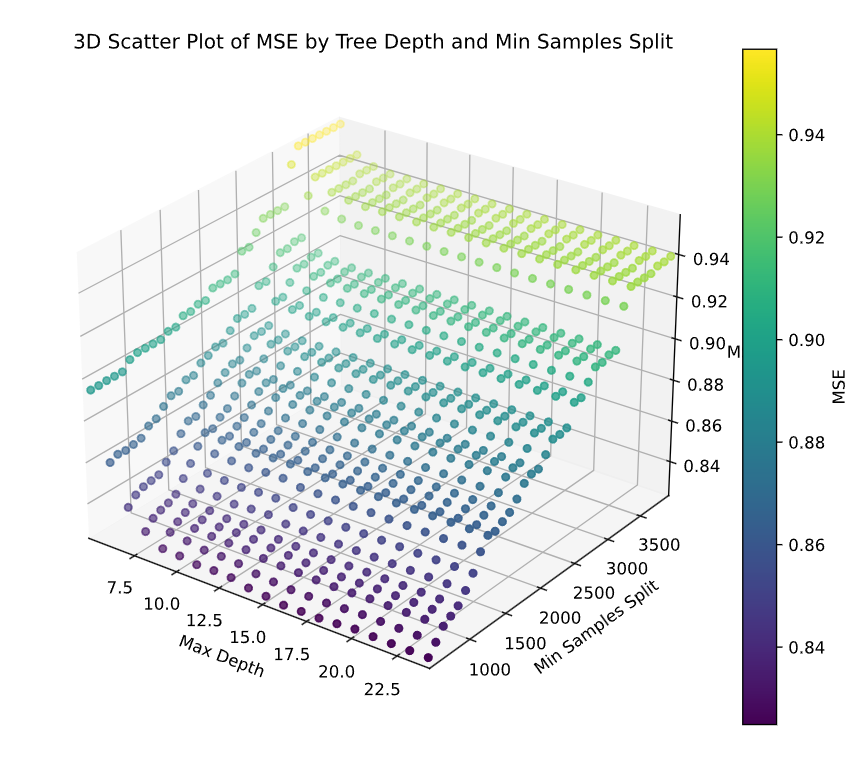
*Fig. 3.1 Sample Decision Tree*

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 yi denotes the mean value of the corresponding node:

 (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 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.

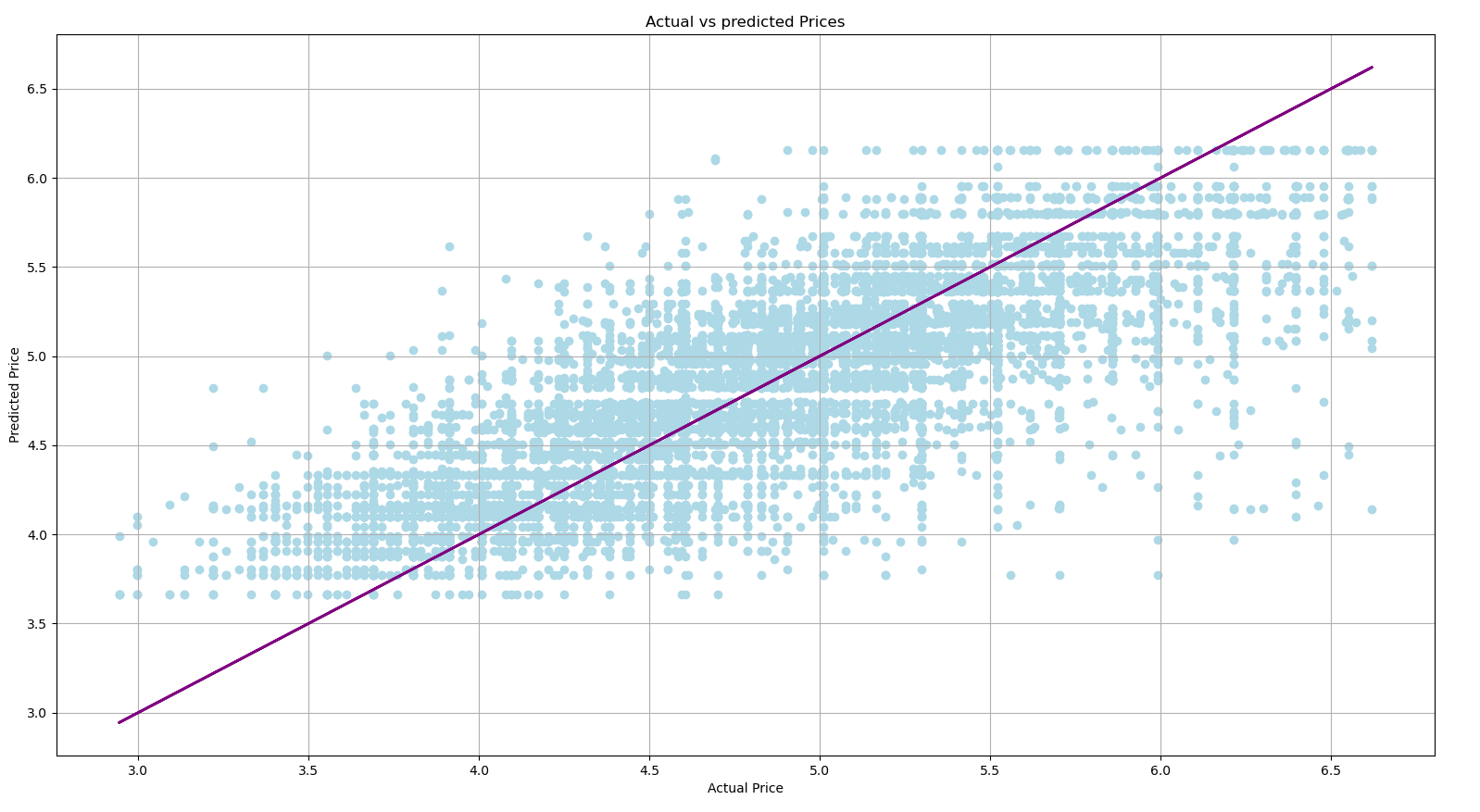
In order to validate the hyperparameters we run an algorithm which assigns different values to each hyperparameter, trains the tree, makes predictions with the validation set which is the 10% of the randomly shuffled dataset and calculates the mean squared error for each hyperparameter duo. After this 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*

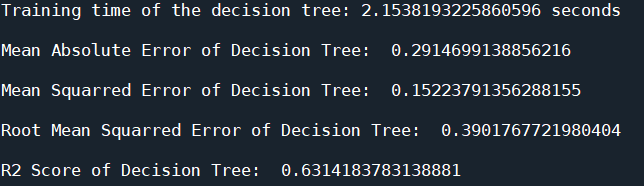
From Fig. 3.2 it can be seen that MSE does not decrease much after max depth = 10 for constant min samples split therefore max depth has chosen as 10 in order to minimize the error while also minimizing computation time. On the other hand, it can be seen that, as minimum sample size per split decreases 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 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.

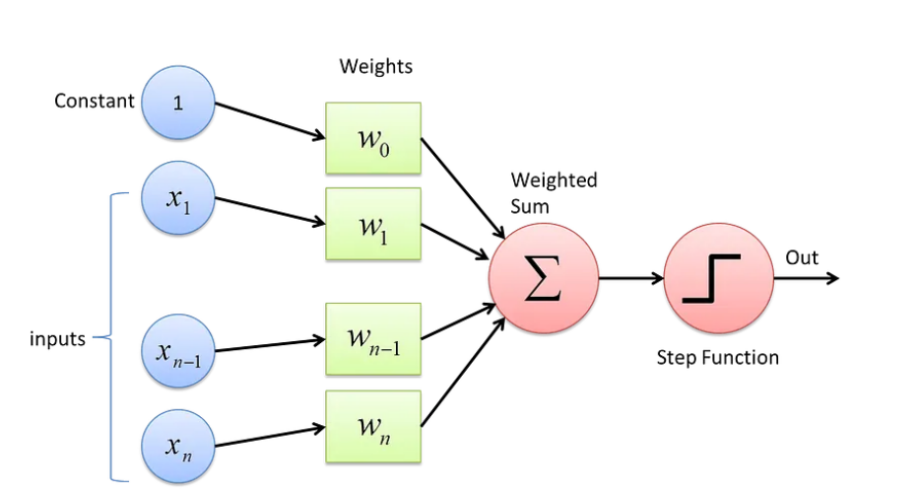


*Fig. 3.4 Performance metrics of Decision Tree*

The result are discussed in the results and conclusion part.

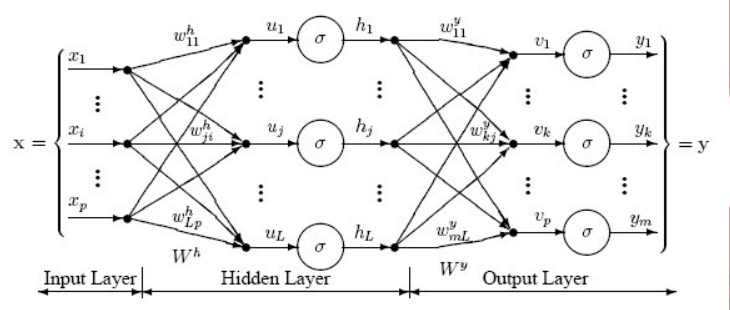
1. **Neural Network**

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 as shown in Fig.4.



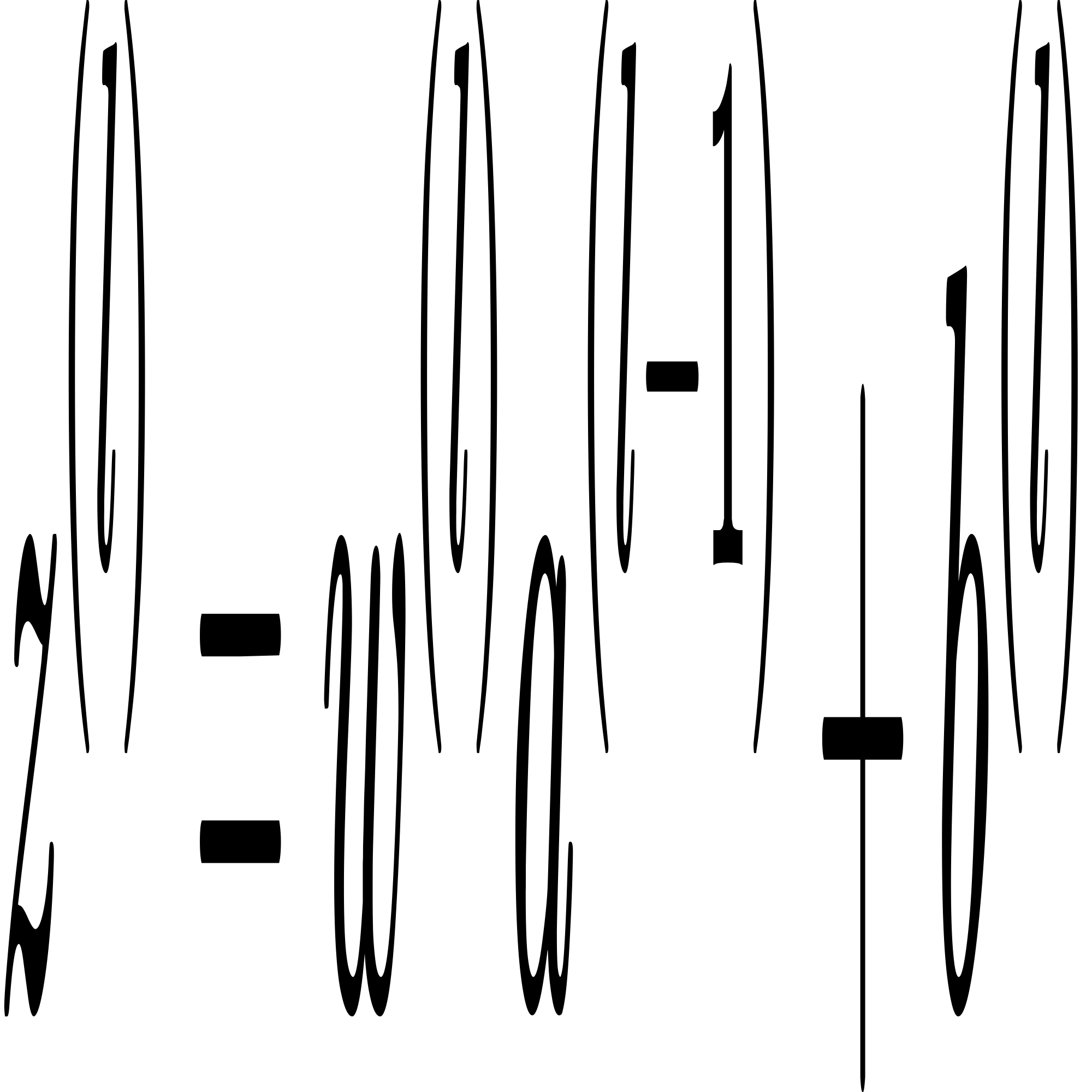
*Fig. 4.1 A Perceptron [A]*

Here the xi’s symbolizes each data feature with an additional constant bias term. These features are then multiplied with specific weights (wi) 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 [B]. 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.5.

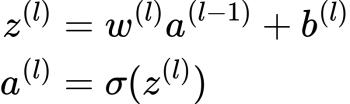


*Fig. 4.2 A Neural Network [C]*

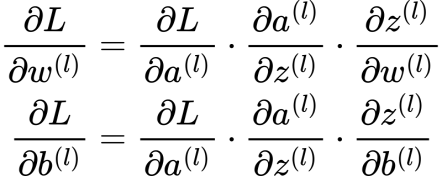
For the Airbnb price prediction, we worked with 12 features corresponding to 12 inputs, one hidden layer and one output, “price”. As the initialization step, the weights were assigned according to standard random gaussian distribution and zero bias. The convergence of weights and bias terms for both 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) [D]. 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 and 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 **wps** using an activation function ***σ*** based on the input , 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 wps and the actual output wps. Here are the steps followed for back-propagation:

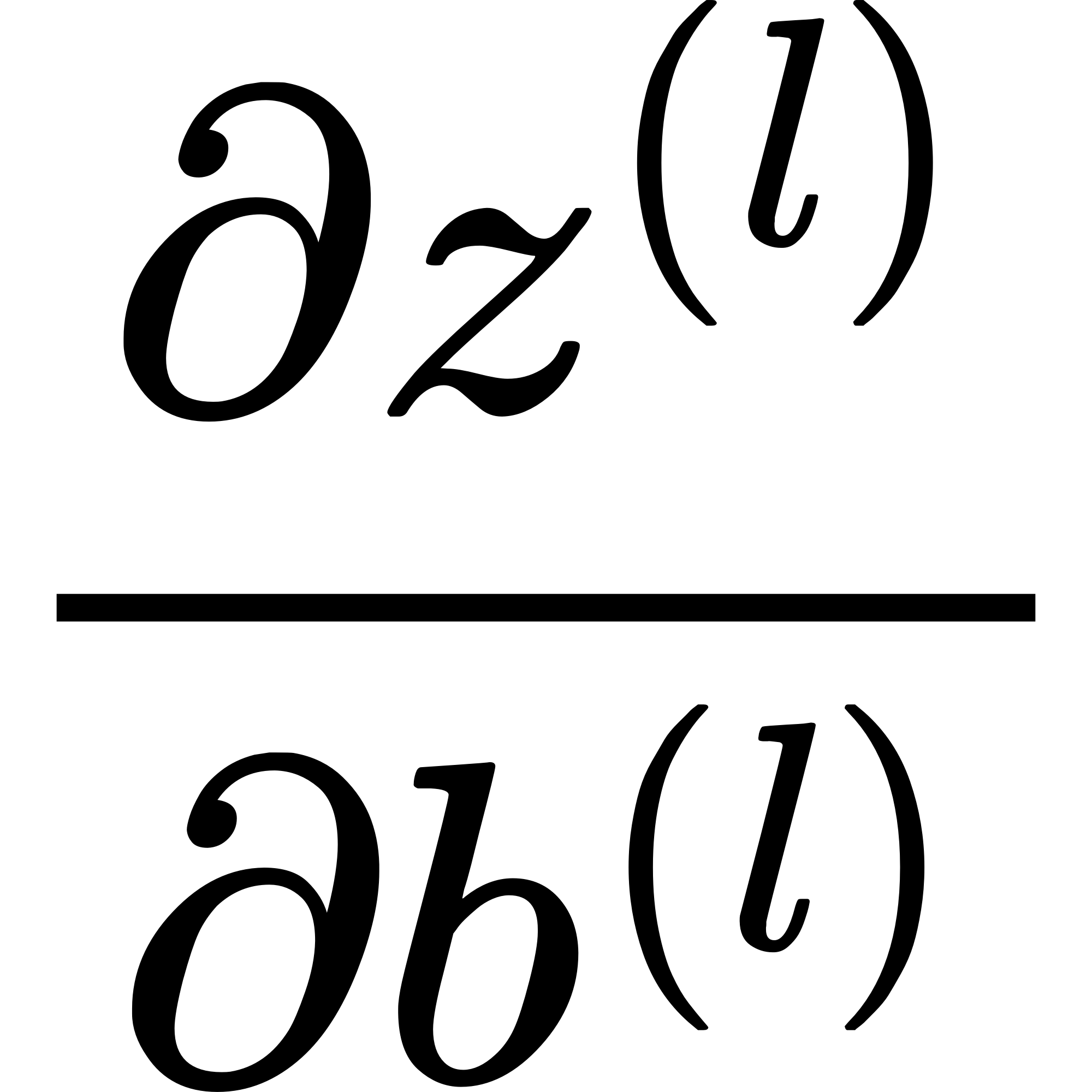
Forward Pass, computing the output for each neuron from the input layer with initial parameter definitions.

 (7)

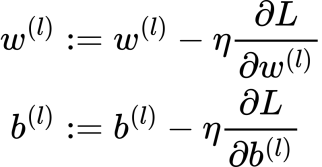
Backward Pass, the gradients of the loss ***L*** with respect to the weights wpsand biases wpsare calculated using the chain rule:

 (8)

Where:

* wpsis the gradient of the loss function with respect to the output of layer ***l****.*
* is the derivative of the activation function ***σ*** at layer ***l***.
* wpsis simply wps.
* is 1.

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

 (9)

After the training process, we obtained the predicted prices via forward propagation. Hyperparametreleri açıklayacağız ne işe yaradıklarını anlatacağız validation processimizi yazacağız ve sonuçları koyacağız

**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. R2 score of Linear Regression was 0.6406 whereas for Decision Tree it was 0.6466. Higher R2 score corresponds to better predictions which means the model with the highest R2 score can be considered a more accurate model. In our case, R2 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 choice 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].

[A] https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

1. <https://www.geeksforgeeks.org/activation-functions-neural-networks/>
2. https://www.dtreg.com/solution/multilayer-perceptron-neural-networks
3. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

**Appendices**

**Appendix A**

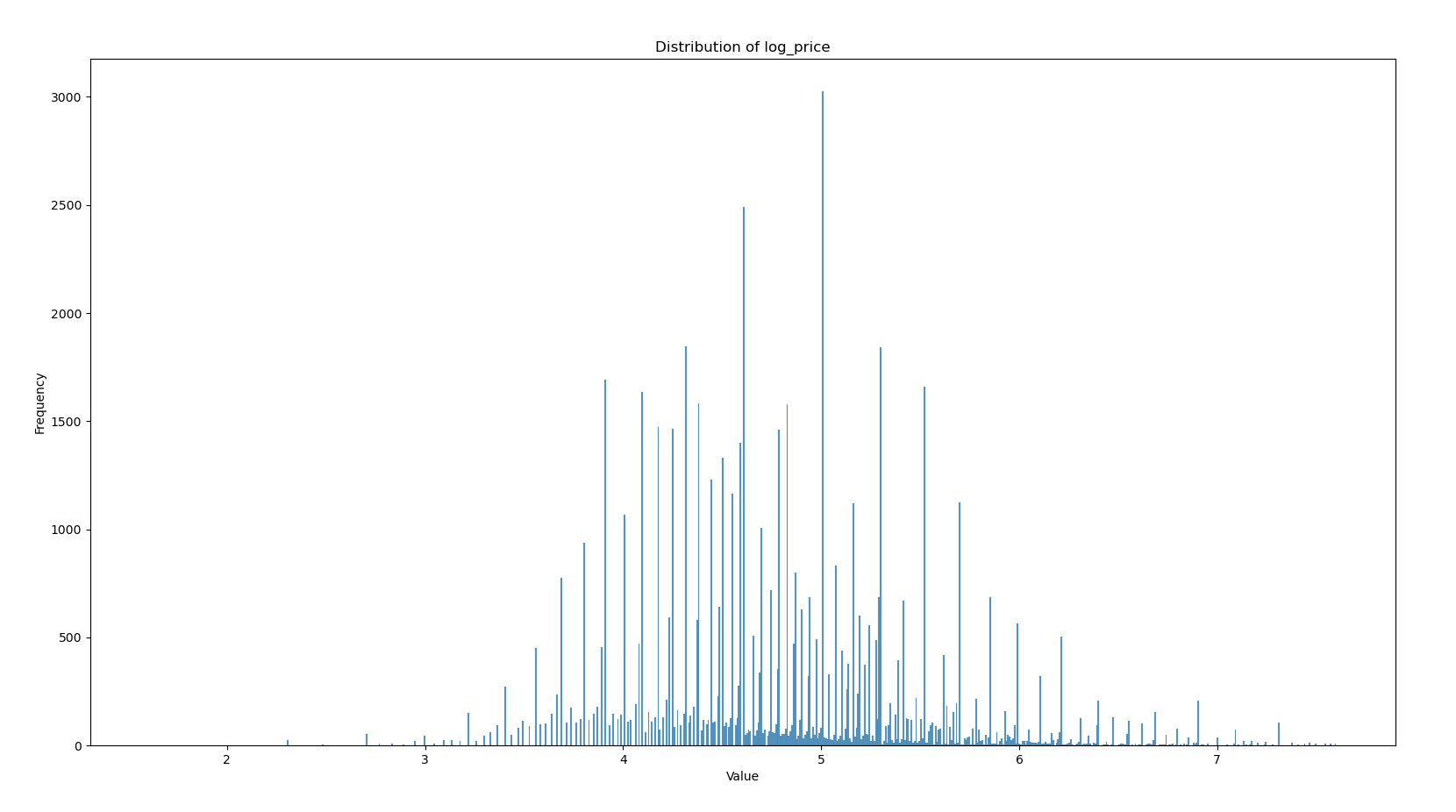
**Gannt Chart:**



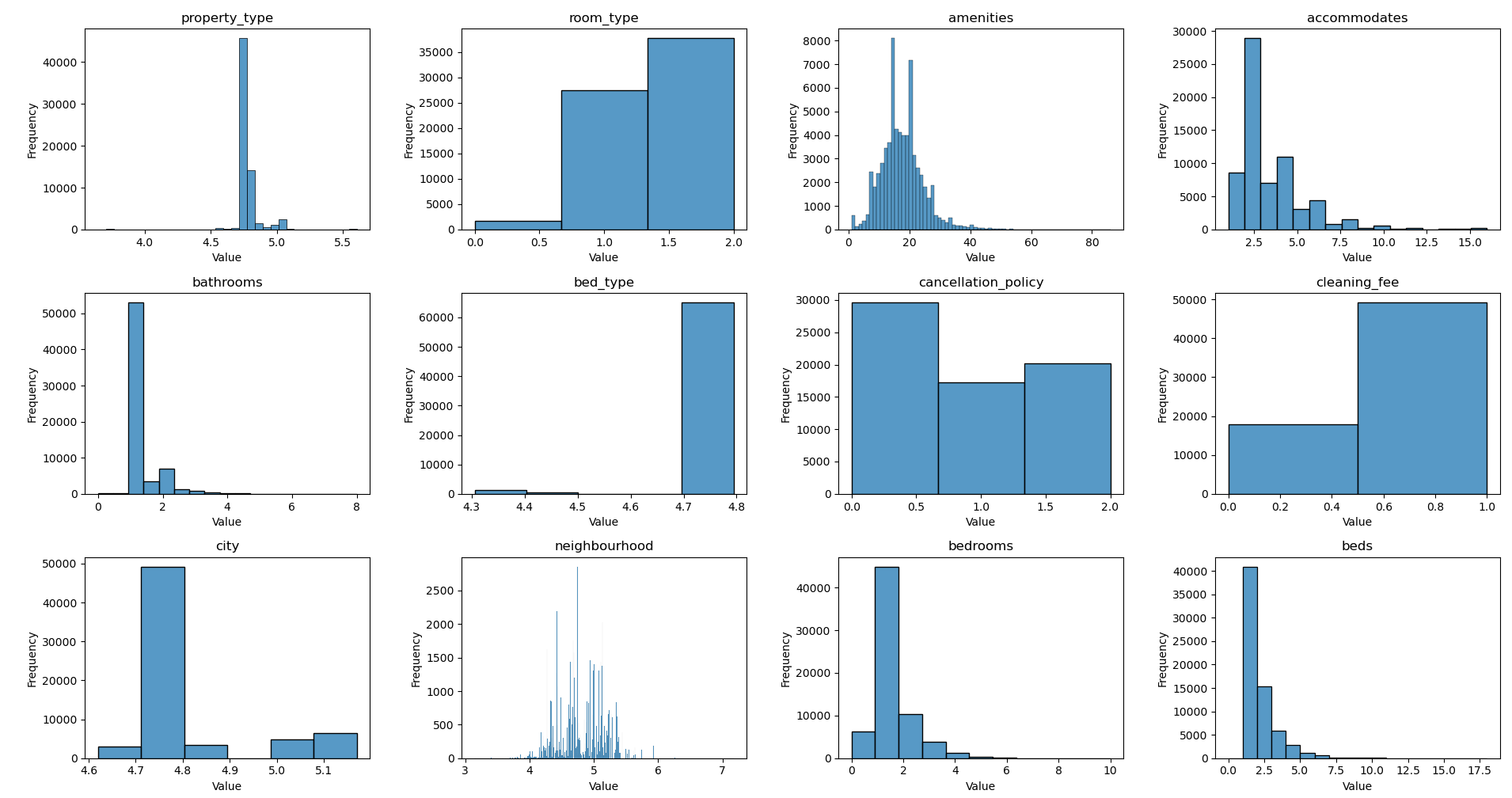
*Fig. A.1: Gannt Chart*

**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()