Apartment for Rent - Machine Learning Project Report

Atharva Sardar

12/08/2023

—

Machine Learning (CS 5805)

—

Dr. Reza Jafari

Table of Contents

Table of Figures

Table of Tables

Abstract:

Today in the world where we generate a very large amount of data, it provides us with a great opportunity to solve our day-to-day problems by applying our knowledge of machine learning. One of the biggest challenges people face when moving to a new place face is to find an apartment to rent. After finding the apartment, there is no way to figure out whether the price we are paying is in line with the market price of the property. Also we tend to have some needs like number of bedroom, based on the number of people living, location needs based on our daily routine, pets allowed or not and many more. So based on our need will we get a affordable apartment or not, is an interesting problem to solve. We tend to use different machine learning techniques like regression and classification to provide optimum information about the apartments we want to rent.

Introduction:

This project works with a "Apartment for rent classified" dataset which provides us with 100000 data entries. It provides us with a total of 22 features of which we choose 11 features them being 'id','category', 'fee', 'has_photo', 'pets_allowed', 'price_type', 'bathrooms', 'bedrooms', 'square_feet', 'latitude', 'longitude', 'price'. After this we explore the data. Initially we use different techniques to tackle with the missing data like dropping some observations while replacing some other observations with a proper keyword. Also we check for any duplicate rows and remove them. Then we encode the categorical features using either one-hot encoding or label encoding. Then we normalize and standardize the numerical features of the data. To reduce the number of features, we apply different dimensionality reduction techniques such as random forest, pca, svd and VIF for collinearlity check. We find that random forest model reduces the dimensions of the data to 6 and this is the most effective in dimentionality reduction. We check for any outliers using the concept of quartiles. Then to predict the price of a apartment based on the feature we apply multiple linear regressor on the data. After doing this we use the price feature to create the price_category feauture which provides you whether you will get a cheap, affordable or costly apartment based on your needs i.e. based on your features. We apply Decision tree, logistic regression, KNN, SVM, Naïve bayes, Random forest with bagging , stacking and adaboost method. Finally we also apply random forest and neural networks to our data. Then using silhouette analysis, elbow method to find optimum k, we apply K-means ++ algorithm. Along with that we apply DBscan and Apriori algorithm to find any clusters in the data.

Description of the Dataset:

To work on this project, I work on the dataset named "Apartment for rent classified" dataset which is available at UR Irvine Machine learning repository. It contains a total of 100000 data entries. So as we can see the basic requirement that the dataset should have a minimum of 50000 entries is satisfied. Along with that the dataset has total of 22 features out of which there are atleast 2 of each categorical and numerical features. After analyzing the data more closely I choose the following features:

USEFULL CATEGORICAL:
Category, fee, has_photo, pets_allowed, price type,

USEFUL NUMERICAL:
Bathrooms, bedrooms, squarefeet, latitude, longitude

I did this based on the number of unique values like some of the categorical features had unique value for each unique data, which would overfit the data. While a couple of features were recurring so removed them. Along with that a feature had same value for all entries, so basically it did not contain any data, so I removed them.
The main aim of the project is to provide the best apartment possible at the lowest cost possible. So, considering our main motive, I choose price as the target variable. To make it relevant, I spoke with some of the students about their experience for the house hunt. The main thing that came out of that conversation was that they think that they are overpaying of the property and also they think that it is not worth it for the facilities and location they are getting. So this problem can be solved by providing the estimated price based on the features they have and can also categorize it into cheap, affordable or costly. So if they want a apartment in cheap category, they can cut on some of the features and know which feature they need to compromise on.

Phase I: Feature Engineering & EDA:

a. Missing Values:

First of all we check the missing value in the data:
Output:
Missing Observations: id 0
category 0
fee 0
has_photo 0
pets_allowed 60424
price_type 0
bathrooms 63
bedrooms 124
square_feet 0
latitude 25
longitude 25
price 1
dtype: int64

We see that the most of the missing value is associated with pets allowed.

Replacing the Null of pets_allowed with NotAllowed:
Missing Observations: id 0
category 0
fee 0
has_photo 0
pets_allowed 0
price_type 0
bathrooms 63
bedrooms 124
square_feet 0
latitude 25
longitude 25
price 1

dtype: int64

Dropping the other null values:
Dropping the Null values
Missing Observations: id 0
category 0
fee 0
has_photo 0
pets_allowed 0
price_type 0
bathrooms 0
bedrooms 0
square_feet 0
latitude 0
longitude 0
price 0
dtype: int64

b. Duplicates Removal:

Now we check for duplicates and remove them.
Output:
Duplicate Rows:

```
             id              category fee  ... latitude longitude   price
41958  5508806580  housing/rent/apartment  No  ...  36.1536 -115.1965  1335.0
41959  5508806428  housing/rent/apartment  No  ...  39.8999 -104.9442  1331.0
41960  5508806391  housing/rent/apartment  No  ...  38.8516  -76.8871  1320.0
41961  5508806299  housing/rent/apartment  No  ...  38.8516  -76.8871  1349.0
41962  5508806233  housing/rent/apartment  No  ...  39.8999 -104.9442  1260.0
...           ...                     ... ..  ...      ...       ...     ...
83243  5197839500  housing/rent/apartment  No  ...  38.2203  -78.3844   725.0
83244  5197836604  housing/rent/apartment  No  ...  35.3201  -80.7409  1237.0
83245  5197834189  housing/rent/apartment  No  ...  33.3924 -111.9265  1476.0
83246  5197828852  housing/rent/apartment  No  ...  33.3924 -111.9265  1877.0
83247  5197828778  housing/rent/apartment  No  ...  33.3924 -111.9265  1862.0
[84 rows x 12 columns]
             id              category fee  ... latitude longitude   price
0      5668640009  housing/rent/apartment  No  ...  33.8520 -118.3759  2195.0
1      5668639818  housing/rent/apartment  No  ...  37.0867  -76.4941  1250.0
2      5668639686  housing/rent/apartment  No  ...  35.8230  -78.6438  1395.0
3      5668639659  housing/rent/apartment  No  ...  38.3622 -121.9712  1600.0
4      5668639374  housing/rent/apartment  No  ...  35.1038 -106.6110   975.0
...           ...                     ... ..  ...      ...       ...     ...
99487  5121219946  housing/rent/apartment  No  ...  29.6151  -95.1998   780.0
99488  5121219696  housing/rent/apartment  No  ...  30.2254  -81.7579   813.0
99489  5121219420  housing/rent/apartment  No  ...  32.7379 -117.0914  1325.0
99490  5121218935  housing/rent/apartment  No  ...  35.4158  -80.8451   931.0
99491  5121218844  housing/rent/apartment  No  ...  32.7379 -117.0914  1595.0
[99196 rows x 12 columns]
```

c. Discretization & Binarization and variable Transformation:

I apply one hot encoding for the fee feature because it has only 2 values yes and no. So performing one hot encoding does not increase the number of features as we just need one feature to encode fee.
Label encode 'category', 'has_photo', 'pets_allowed', 'price_type' because they are multiclass categorical features.
We perform Standardisation and normalisation on the numerical features of the dataset i.e. features being 'bathrooms', 'bedrooms', 'square_feet', 'latitude', 'longitude'

Output:
Label-Encoded Data:

```
           id  category  fee  ...  latitude  longitude   price
0  5668640009         1  0.0  ...   33.8520  -118.3759  2195.0
1  5668639818         1  0.0  ...   37.0867   -76.4941  1250.0
2  5668639686         1  0.0  ...   35.8230   -78.6438  1395.0
3  5668639659         1  0.0  ...   38.3622  -121.9712  1600.0
4  5668639374         1  0.0  ...   35.1038  -106.6110   975.0
[5 rows x 12 columns]
```

d. Dimentionality Reduction:

Output:
Selected Features from Random Forest Analysis:
['id', 'bathrooms', 'bedrooms', 'square_feet', 'latitude', 'longitude']
Selected Features based on PCA:

```
Index(['id', 'category', 'fee', 'has_photo', 'pets_allowed', 'price_type',
       'bathrooms', 'bedrooms', 'square_feet', 'latitude', 'longitude'],
      dtype='object')
```

Selected Features and Their Contributions in the SVD Analysis:

```
         Feature  Contribution
0             id  1.000000e+00
4   pets_allowed  5.220997e-10
3      has_photo  2.735250e-10
1       category  1.865268e-10
9       latitude  1.604513e-12
6      bathrooms -4.668629e-13
2            fee  3.724988e-13
8    square_feet -2.098196e-13
7       bedrooms -9.166490e-14
5     price_type  4.743785e-15
10     longitude  4.123105e-15
```

Selected Features from VIF:
['id', 'category', 'fee', 'has_photo', 'pets_allowed', 'price_type', 'bathrooms', 'bedrooms', 'square_feet', 'latitude', 'longitude']

Random forest gives the best of significant features. Vif removes the correlated features from the data which is also a good option. PCA takes into consideration condition number.
We choose the random forest method because random forest reduces the features to the lowest value with the selected features being the most important ones.

Output:
Feature Reduced Data:
<bound method DataFrame.to_string of id bathrooms bedrooms square_feet latitude longitude price

```
0  5668640009  -0.814713 -0.972724    -1.133082 -0.672887  -1.696449  2195.0
1  5668639818   0.099198  1.697882     1.488360  0.030280   0.952493  1250.0
2  5668639686   1.013110  1.697882     1.898815 -0.244427   0.816529  1395.0
3  5668639659  -0.814713  0.362579    -0.372371  0.307551  -1.923845  1600.0
4  5668639374  -0.814713 -0.972724    -0.908700 -0.400768  -0.952342   975.0>
```

e. Outlier Detection

We use the 1.5 times of the interquartile range as the lower and upper bound of the data to remove any outliers outside it.
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
Output:
Original Dataset Shape: (99196, 7)
Dataset Shape After Outlier Removal: (94505, 7)

Observation:
Based on our method for outlier detection we remove around 5000 data entries which lie outside the bound. This is based on the fact that most of the important data lies in the 1.5 times the IQR from the Q1 and Q3. So we consider other points as outlier and remove them.

f. **Sample Covariance Matrix:**

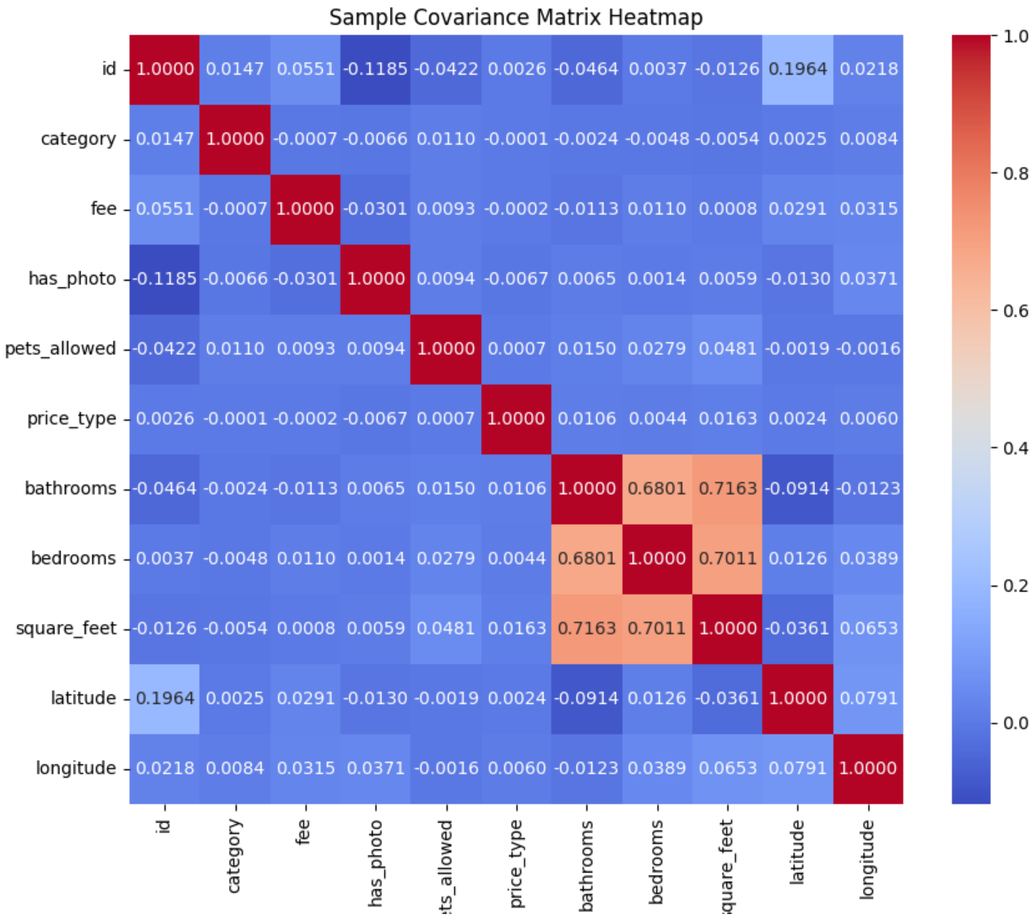

Figure 1 Sample Covariance Matrix

We see that the bathrooms, bedrooms and square_feet are correlated to each other. It is consistent with the real life, as with increase in the number of rooms increases the square_feet. Other features are not highly correlated to each other.

g. **Sample Pearson Correlation coefficients Matrix :**

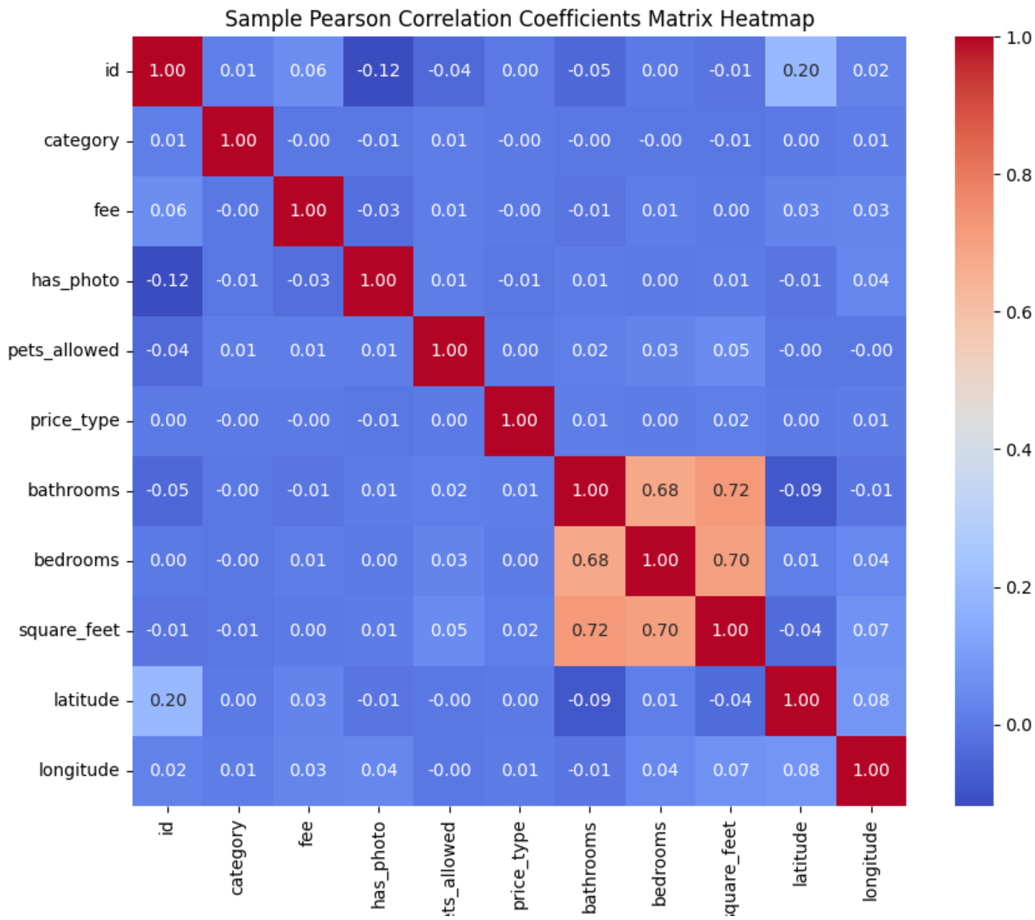

Figure 2 Sample correlation matrix

We see that the bathrooms, bedrooms and square_feet are correlated to each other. It is consistent with the real life, as with increase in the number of rooms increases the square_feet. Other features are not highly correlated to each other.

h. **We create the classification feature for the classification analysis so we create it accordingly that the dataset is balanced**

Phase II: Regression Analysis:

We apply linear regression model on the data with the target variable being the price.

a. T-test Analysis

Output:
T-test analysis:

```
                    Test for Constraints
==============================================================================
            coef    std err        t      P>|t|     [0.025      0.975]
------------------------------------------------------------------------------
c0       962.8027    52.820    18.228     0.000    859.276    1066.329
c1      8.474e-08  9.86e-09     8.597     0.000   6.54e-08    1.04e-07
c2        77.5865     2.812    27.590     0.000     72.075      83.098
c3       -59.5271     2.756   -21.595     0.000    -64.930     -54.124
c4       184.0290     3.156    58.311     0.000    177.843     190.215
c5        82.4263     1.830    45.031     0.000     78.839      86.014
c6       -51.6908     1.845   -28.023     0.000    -55.306     -48.075
==============================================================================
```

The T-test analysis reveals significant coefficients for variables c0, c1, c2, c3, c4, c5, and c6, indicating their substantial impact on the dependent variable.
The low p-values (0.000) suggest strong evidence against the null hypothesis, reinforcing the reliability of these coefficients in explaining the observed constraints.

b. F-test Analysis:

Output:
F-test analysis:
<F test: F=2100.920964189656, p=0.0, df_denom=7.56e+04, df_num=6>

The F-test statistic of 2100.92 with a p-value of 0.0 indicates a highly significant result, suggesting that the overall regression model, involving six variables, is statistically significant.
The large F-value and extremely low p-value imply that the inclusion of the variables in the model collectively contributes significantly to explaining the variance in the dependent variable, reinforcing the model's overall goodness of fit.

c. Confidence Interval Analysis:

Output:
Confidence Intervals for Coefficients:

```
                   0                    1
```

```
const          8.592761e+02     1.066329e+03
id             6.541700e-08     1.040550e-07
bathrooms      7.207482e+01     8.309813e+01
bedrooms      -6.492976e+01    -5.412435e+01
square_feet    1.778433e+02     1.902147e+02
latitude       7.883862e+01     8.601394e+01
longitude     -5.530617e+01    -4.807534e+01
```

Higher positive value for feature has positive effect on target variable and vice versa.

d. Regression Analysis Output:

```
Output:
Train Set Metrics:
R-squared:             0.1416730225363183
Adjusted R-squared:    0.1416048986443017
AIC:                   1150882.5633701482
BIC:                   1150947.1962214438
MSE:                   239054.45848905912
Test Set Metrics:
R-squared:             0.1416730225363183
Adjusted R-squared:    0.1416048986443017
AIC:                   1150882.5633701482
BIC:                   1150947.1962214438
MSE:                   237810.14930851903
```

```
OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.100
Model:                            OLS   Adj. R-squared:                  0.100
Method:                 Least Squares   F-statistic:                 1.055e+04
Date:                Thu, 07 Dec 2023   Prob (F-statistic):               0.00
Time:                        19:56:28   Log-Likelihood:            -7.2142e+05
No. Observations:               94505   AIC:                         1.443e+06
Df Residuals:                   94503   BIC:                         1.443e+06
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1412.6117      1.629    866.902      0.000    1409.418    1415.806
square_feet  187.9646      1.830    102.701      0.000     184.377     191.552
==============================================================================
Omnibus:                     8411.823   Durbin-Watson:                   1.837
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            10988.925
```

Skew: 0.776 Prob(JB): 0.00
Kurtosis: 3.617 Cond. No. 1.14
==
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

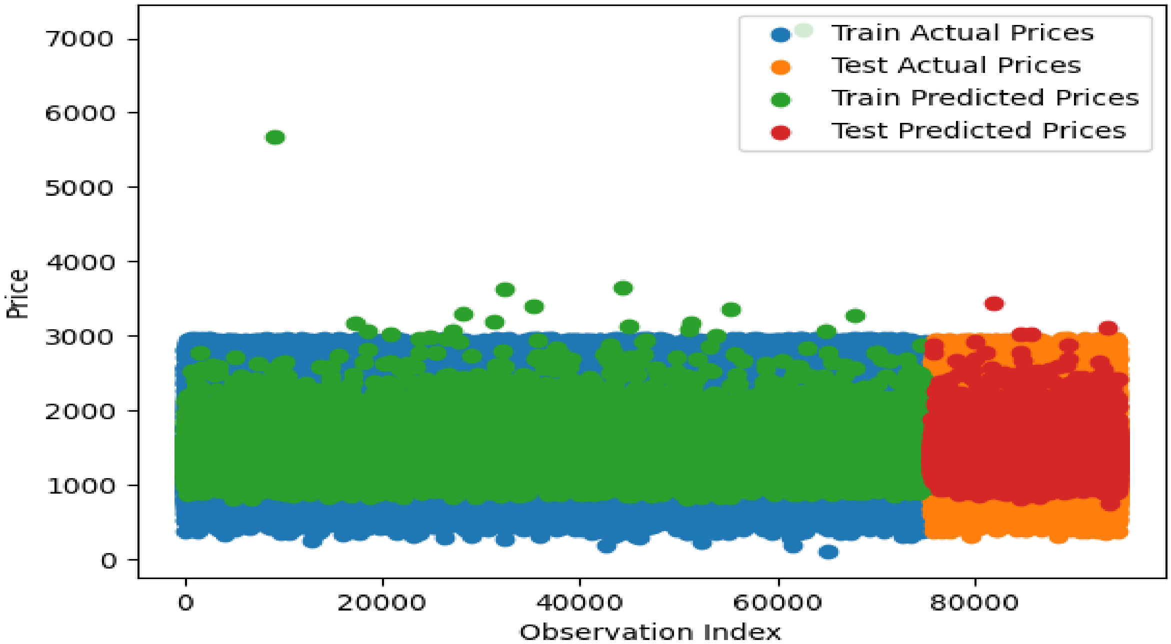

Figure 3 Linear Regression Train-test Data

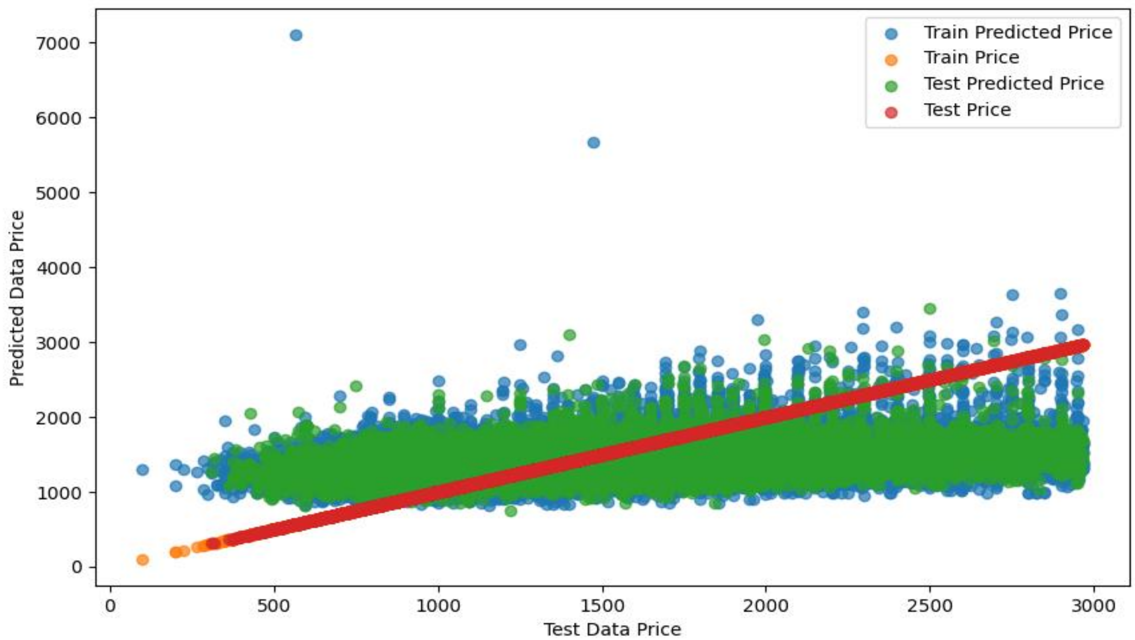

Figure 4 Linear Regression Test-pred data

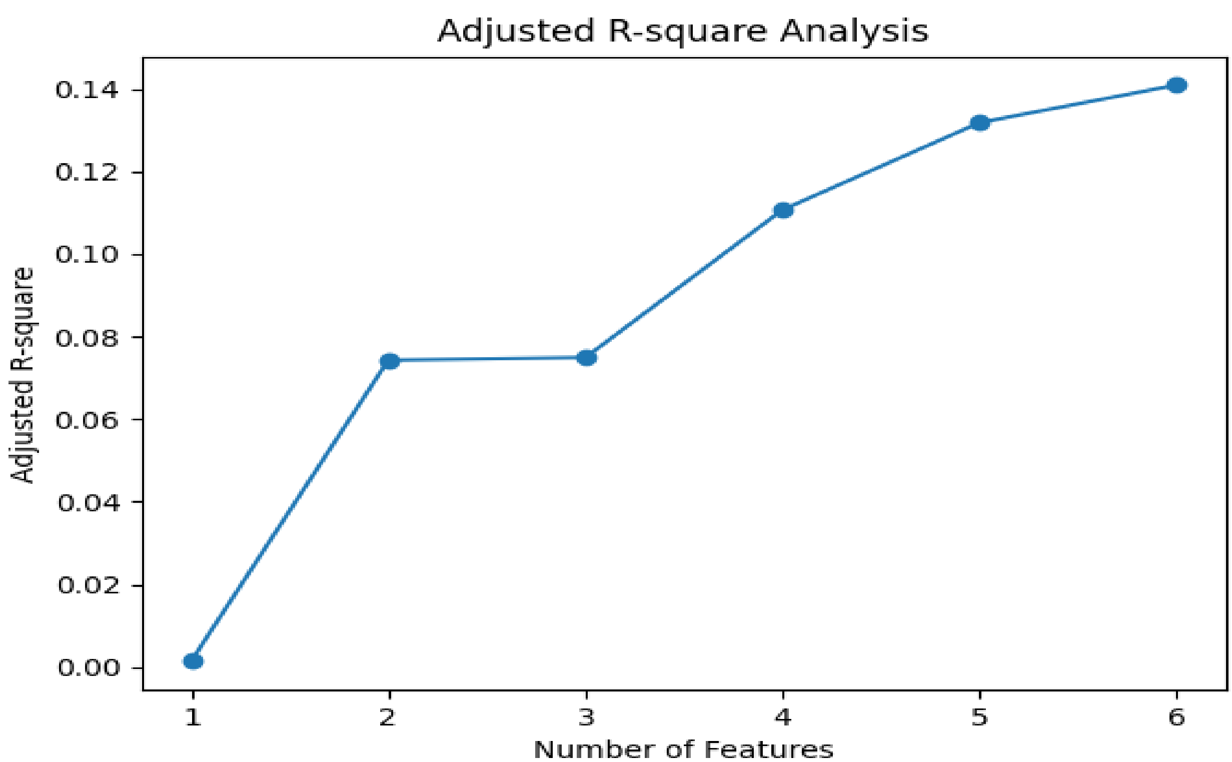

Figure 5 Adj. r-squared analysis

Linear Regression Equation:

Price = 962.80 + 0.00 * id + 77.59 * bathrooms + -59.53 * bedrooms + 184.03 * square_feet + 82.43 * latitude + -51.69 * longitude

Phase III: Classification Analysis:

a. Creating the categorical dataset and encoding it.

Output:
Length of each category:
Cheap 31543
Costly 31494
Affordable 31468
Name: price_category, dtype: int64

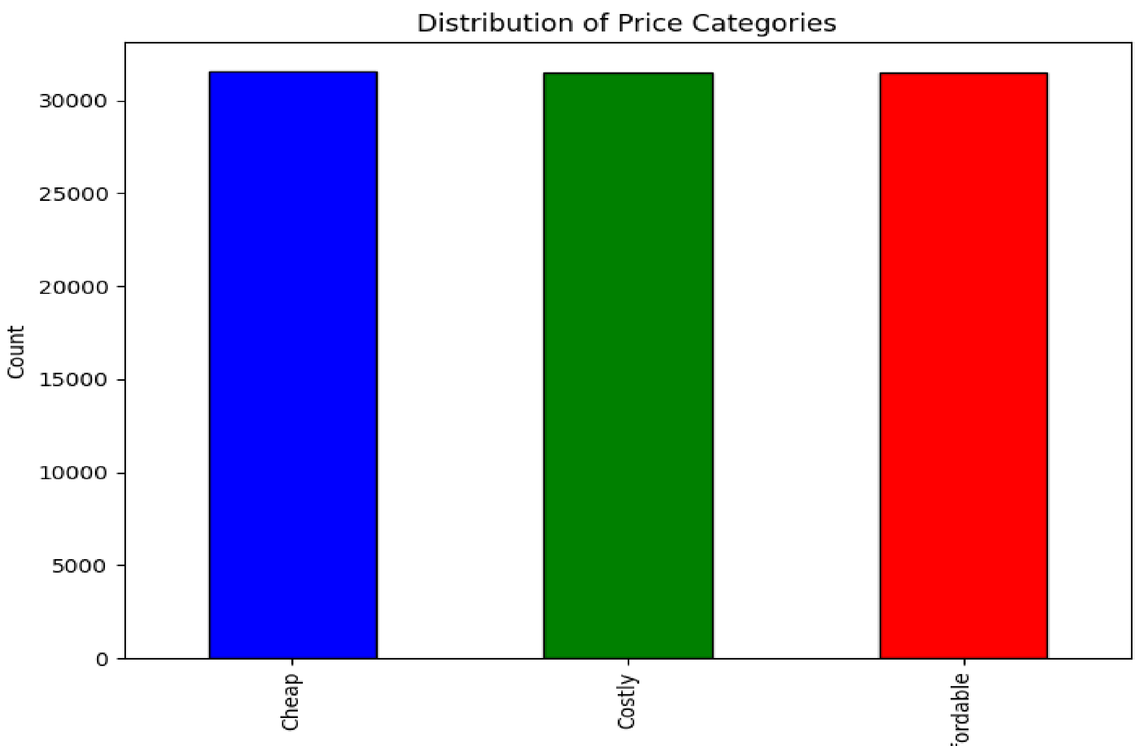

Figure 6 Distribution of price_categories

b. Decision Tree classification

1. Initial Decision Tree

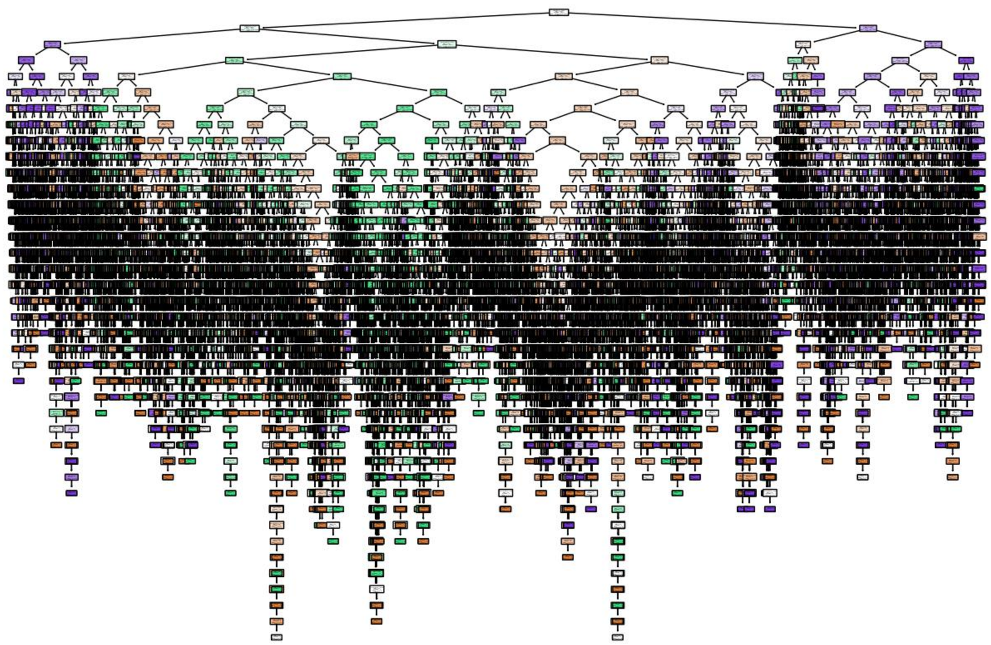

Figure 7 Initial Decision Tree

Roc Curve:

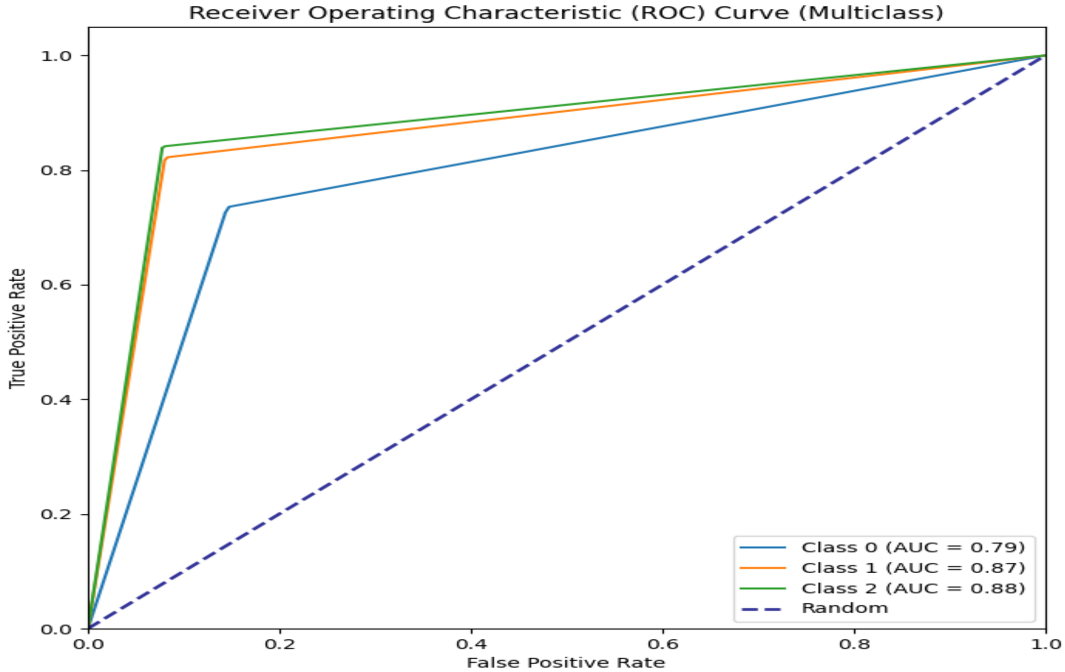

Figure 8 roc curve decision tree

2. Prepruned Decision Tree:

Output:
Best Hyperparameters: {'ccp_alpha': 0.0, 'criterion': 'entropy', 'max_depth': 40, 'max_features': None, 'min_samples_split': 2, 'splitter': 'best'}
Prepruning Decision Tree:
Accuracy: 0.8028146658906936
Classification Report:

	precision	recall	f1-score	support
0	0.72	0.74	0.73	6363
1	0.84	0.82	0.83	6375
2	0.85	0.84	0.84	6163
accuracy			0.80	18901
macro avg	0.80	0.80	0.80	18901
weighted avg	0.80	0.80	0.80	18901

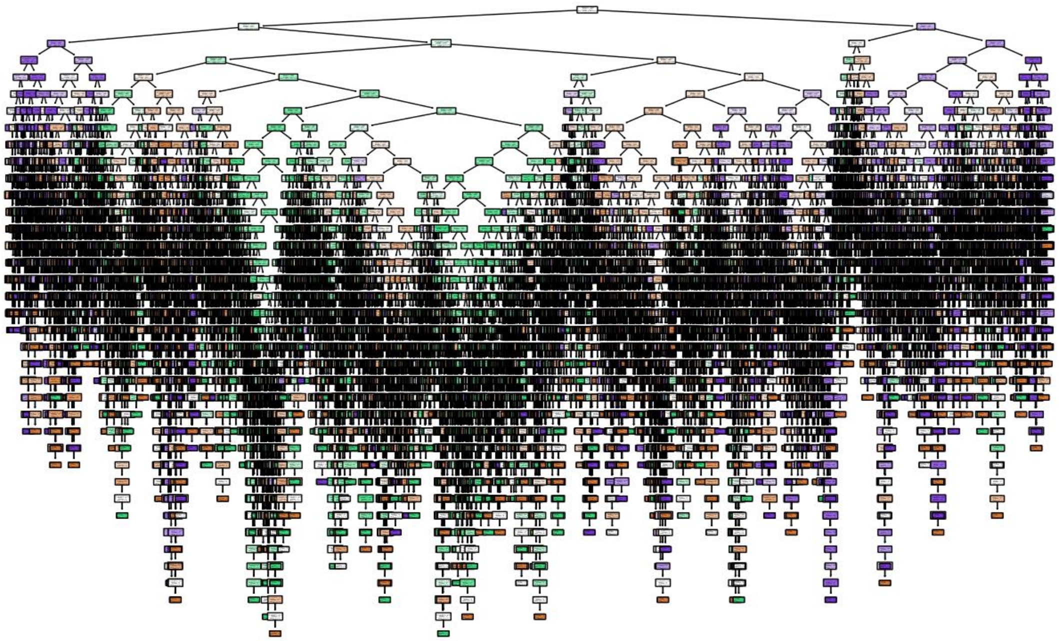

Figure 9 Pre-pruned decision tree

Roc Curve:

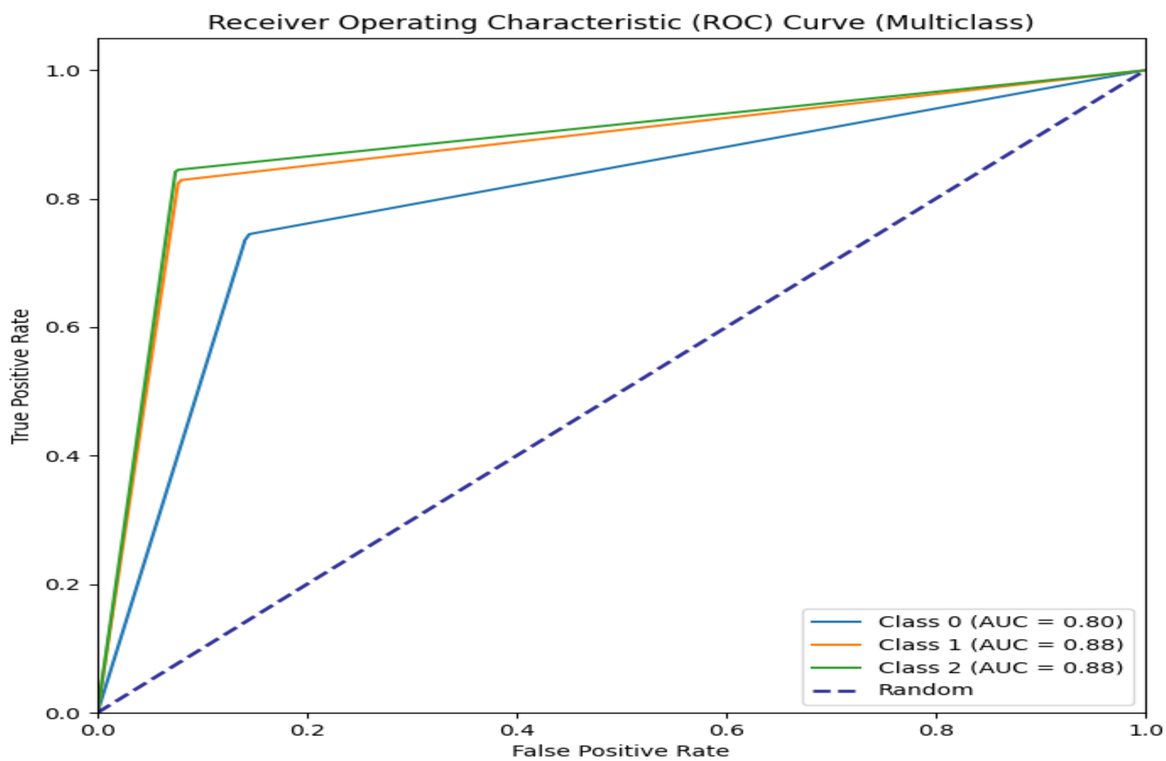

Figure 10 roc curve pre-pruned decision tree

3. Post pruning decision tree

For best alpha:

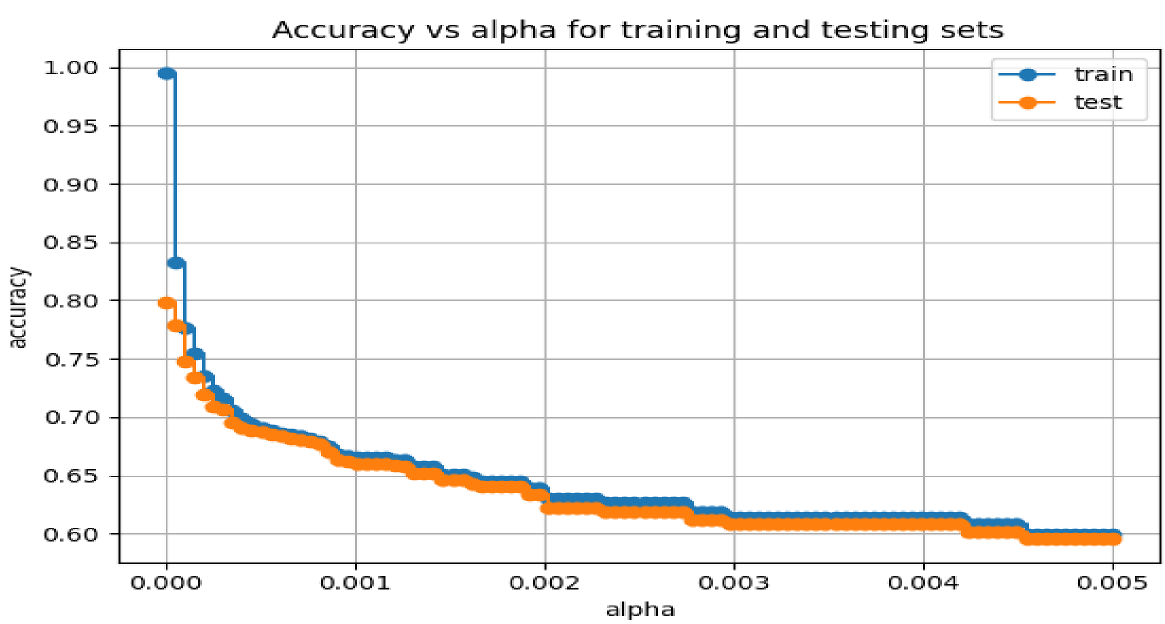

Figure 11 accuracy vs alphs

We consider best alpha to be 0.0001

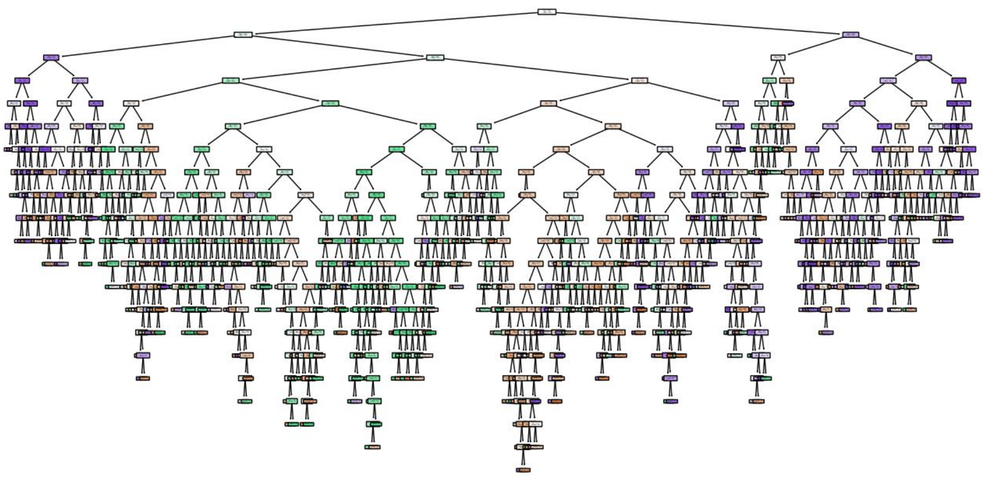

Figure 12 post-pruned decision tree

Post pruning Decision Tree
Post pruning Decision Tree
Train accuracy 0.78
Test accuracy 0.75

Roc Curve:

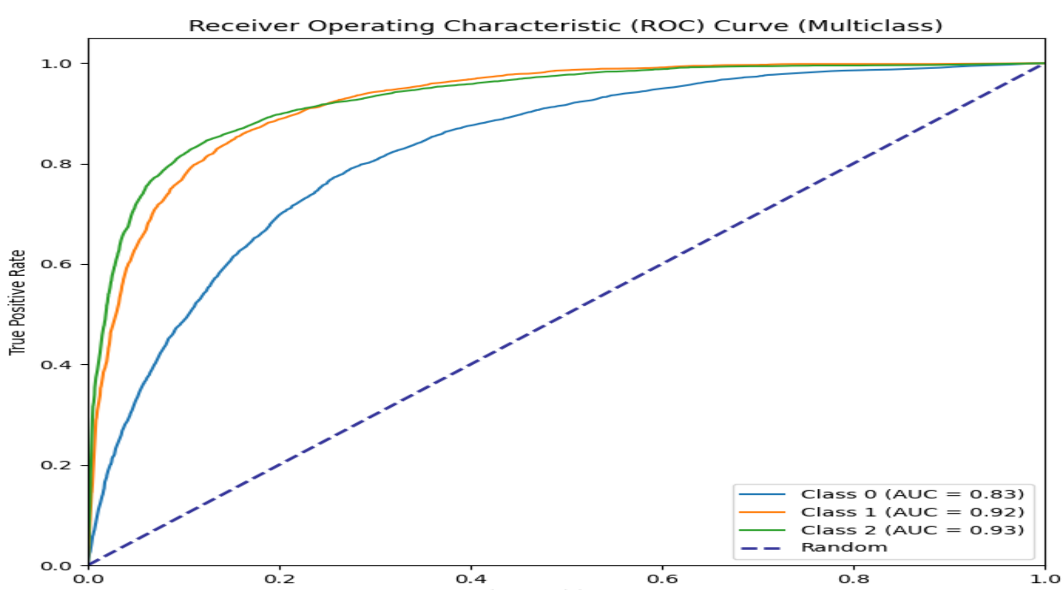

Figure 13 roc curve post-pruned decision tree

c. Logistic Regression

As it works only for binary classification, we apply OVR strategy. In the OvR strategy, a separate binary logistic regression model is trained for each class, treating it as the positive class and the rest as the negative class.

```
[[   0    0 6363]
 [   0    0 6375]
 [   0    0 6163]]
```

Logistic Regression Macro-Averaged Precision score using sklearn library : 0.11
Logistic Regression Micro-Averaged Precision score using sklearn library : 0.33
Logistic Regression Macro-averaged recall score using sklearn : 0.33
Logistic Regression Micro-averaged recall score using sklearn : 0.33
Logistic Regression Macro-Averaged F1 score using sklearn library : 0.16
Logistic Regression Micro-Averaged F1 score using sklearn library : 0.33
Roc Curve:

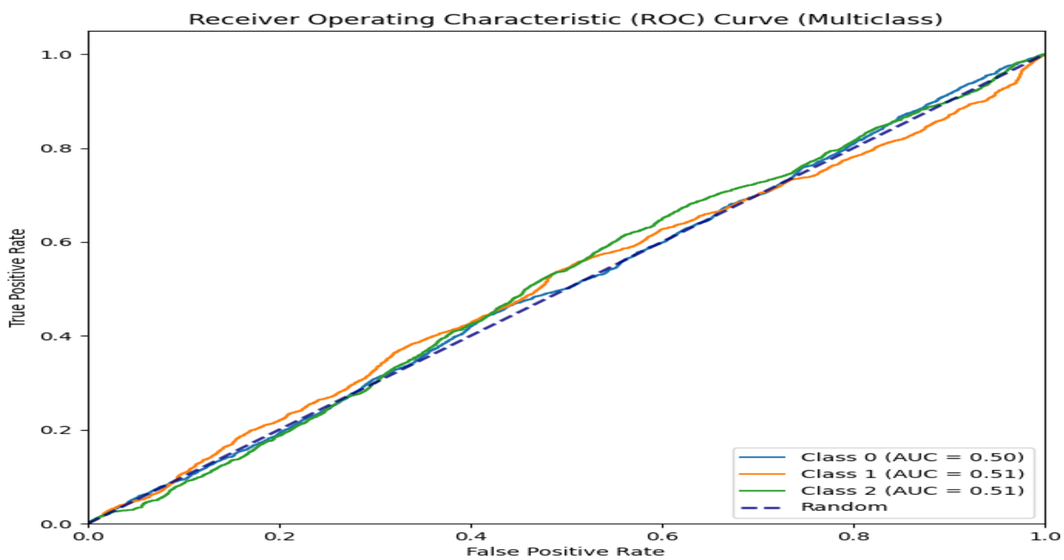

Figure 14 roc curve logistic regression

d. KNN method:

Error vs k-value to find optimal k

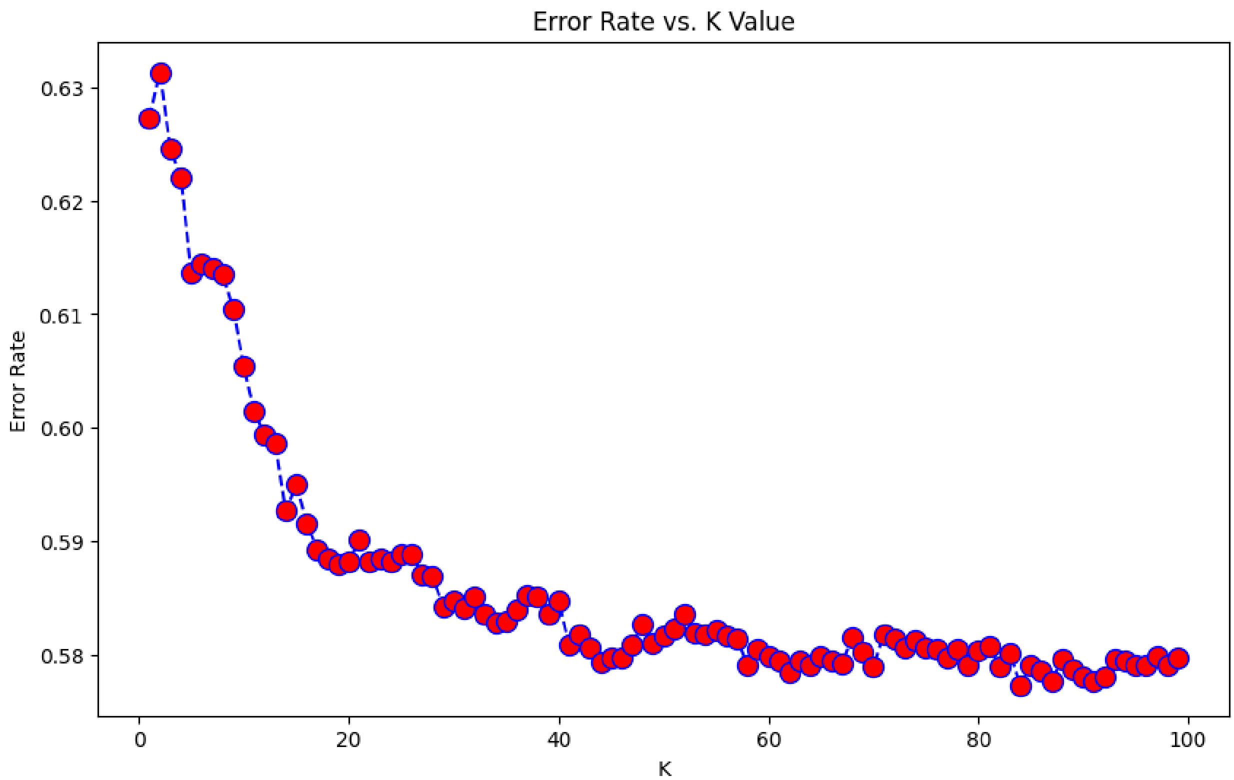

Figure 15 error vs k-value

Output:
Optimal K: 84
Accuracy with Optimal K: 0.42
Roc curve:

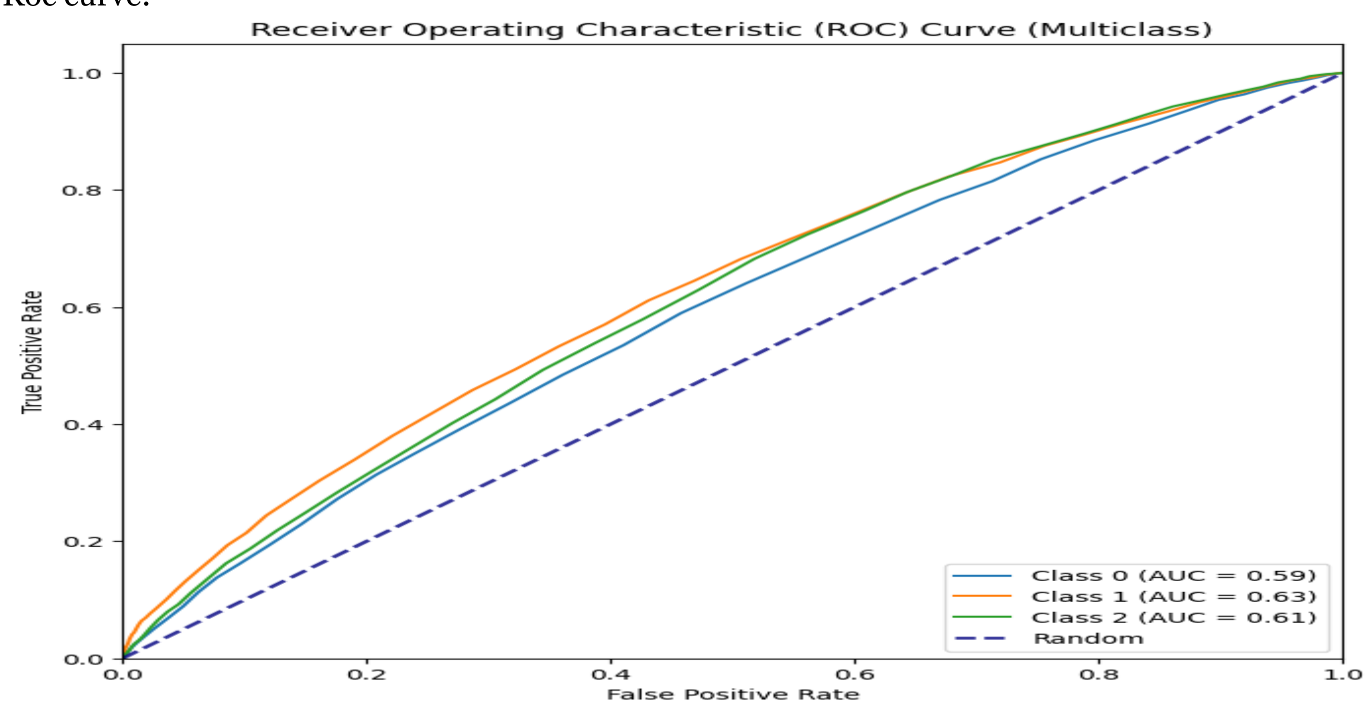

Figure 16 roc curve knn

e. SVM:

Performance of the baseline svm on the test set:
Accuracy of the model: 0.35
Confusion Matrix:
[[0 3141 3222]
[4 3263 3108]
[1 2849 3313]]
Precision Score: 0.23
Recall Score: 0.35
Specificity Score: 0.35
F1 Score: 0.28

Roc Curve:

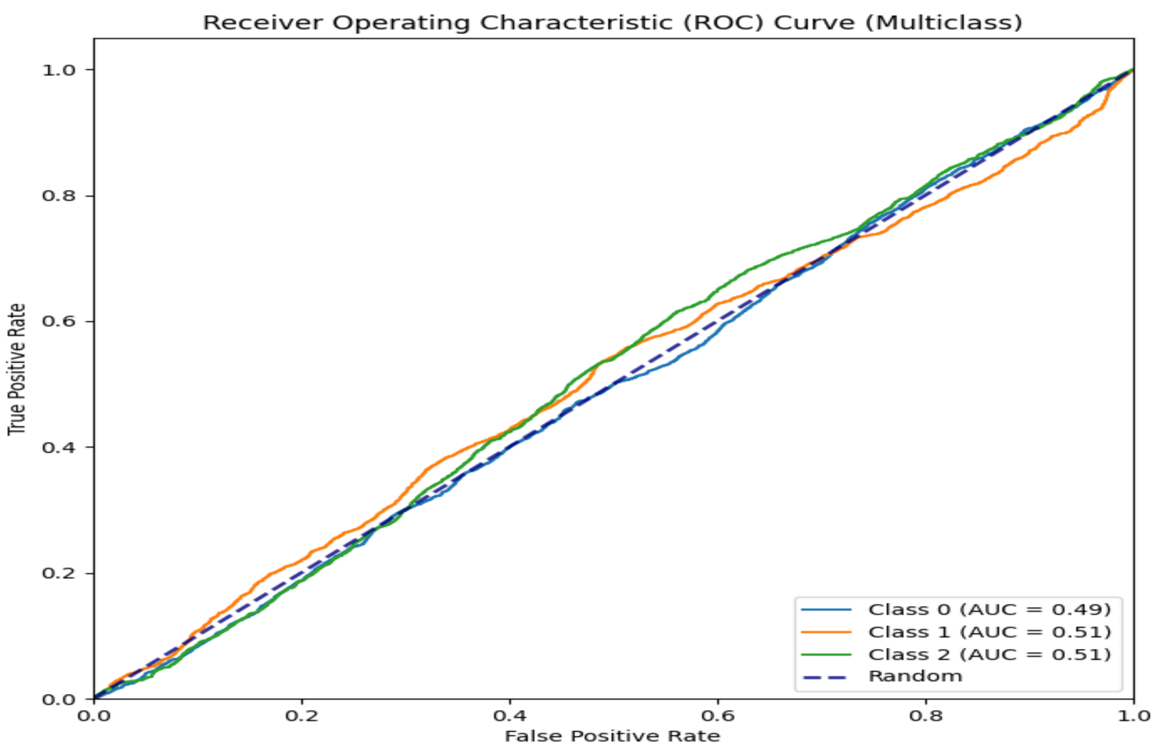

Figure 17 roc curve svm

Best Estimator:
SVC(C=1, gamma=0.1, probability=True, random_state=5805)
Best Score:
0.3376276204402651
Best Paramters:
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}

f. Naïve bayes

Roc Curve:

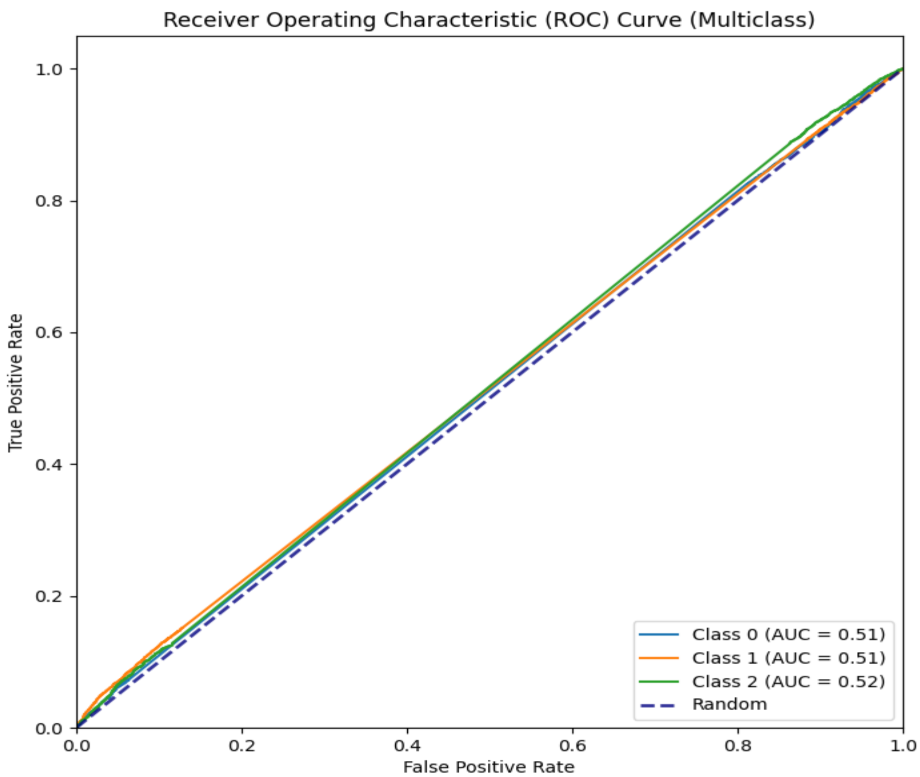

Figure 18 roc curve Naive bayes

g. Random forest (Adaboost method)

AdaBoost Classifier:
Accuracy: 0.64

	precision	recall	f1-score	support
0	0.52	0.51	0.51	6363
1	0.70	0.72	0.71	6375
2	0.71	0.70	0.70	6163
accuracy			0.64	18901
macro avg	0.64	0.64	0.64	18901
weighted avg	0.64	0.64	0.64	18901

```
[[3242 1667 1454]
 [1475 4580  320]
 [1537  319 4307]]
```

Roc Curve:

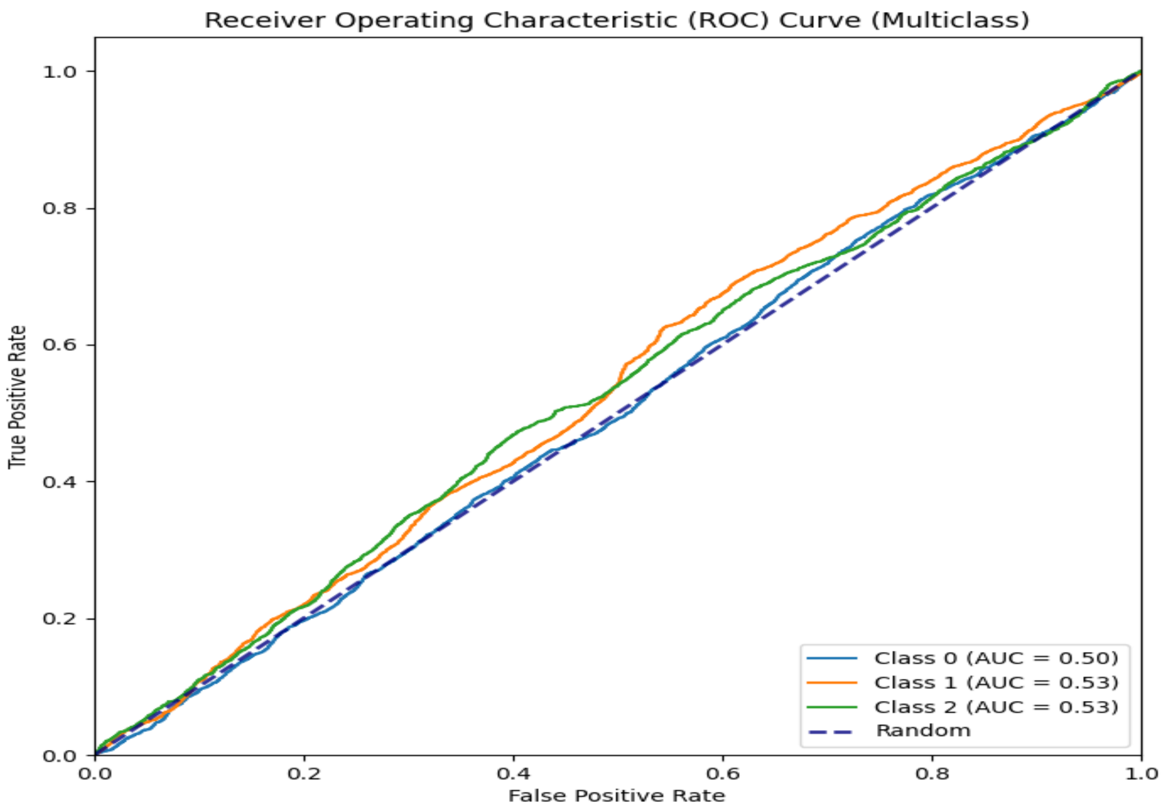

Figure 19 roc curve random forest

h. Stacking Random Forest

Stacking Classifier:
Accuracy: 0.83

	precision	recall	f1-score	support
0	0.76	0.77	0.77	6363
1	0.87	0.86	0.87	6375
2	0.87	0.87	0.87	6163
accuracy			0.83	18901
macro avg	0.83	0.83	0.83	18901
weighted avg	0.83	0.83	0.83	18901

```
[[4884  745  734]
 [ 784 5503   88]
 [ 735   93 5335]]
```

Roc Curve:

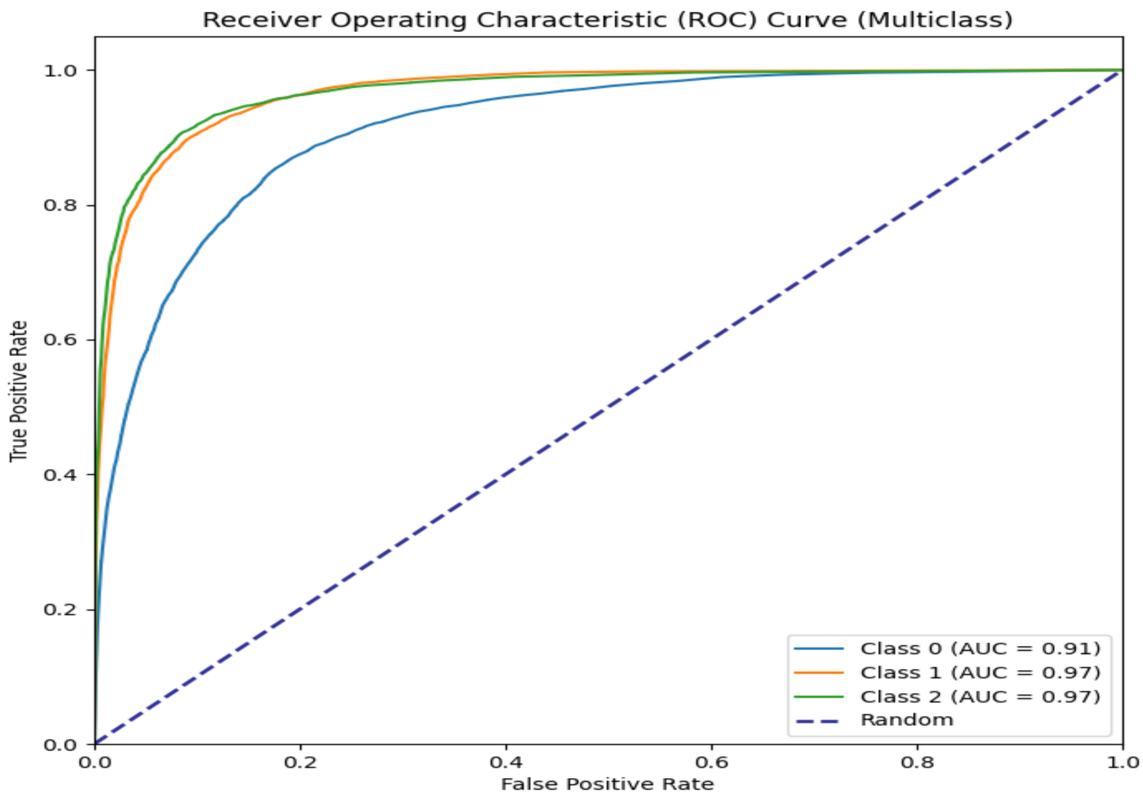

Figure 20 roc curve stacking random forest

i. Neural Network

Multi-layered Perceptron (MLP) Classifier:
Accuracy: 0.70

	precision	recall	f1-score	support
0	0.60	0.58	0.59	6363
1	0.74	0.78	0.76	6375
2	0.78	0.75	0.76	6163
accuracy			0.70	18901
macro avg	0.70	0.70	0.70	18901
weighted avg	0.70	0.70	0.70	18901

[[3686 1531 1146]
[1192 4996 187]
[1312 253 4598]]

Roc Curve:

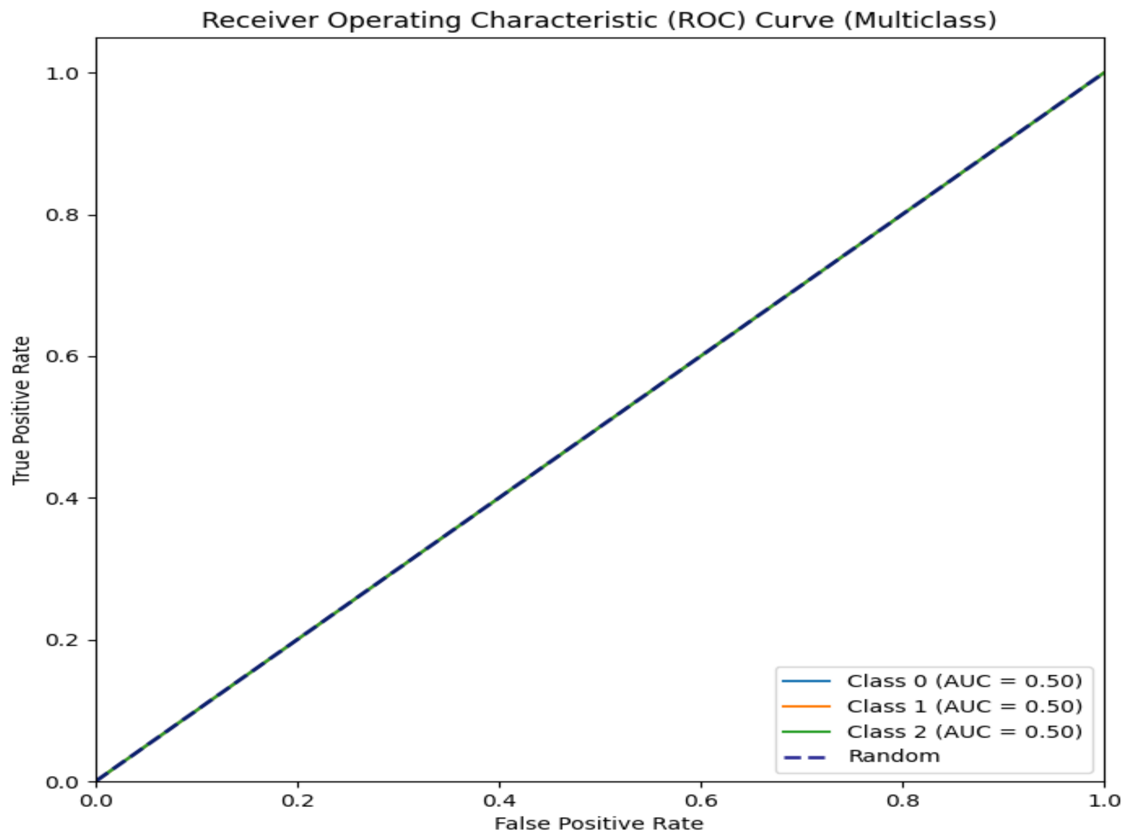

Figure 21 roc curve Neural Network

Classification Report:

Table 1 Classification Report

	Model	Accuracy	Precision	Recall	F1 Score	Specificity	Cross Validation Mean Score
0	Decision Tree	0.80	0.80	0.80	0.80	0.80	0.79
0	Prepruned Decision Tree	0.80	0.80	0.80	0.80	0.80	0.79
0	Postpruning Decision Tree	0.75	0.75	0.75	0.75	0.75	0.75
0	Logistic Regression	0.33	0.11	0.33	0.16	0.33	0.34
0	KNN	0.42	0.42	0.42	0.42	0.42	0.42
0	SVC	0.35	0.23	0.35	0.28	0.35	0.36
0	Best SVC	0.34	0.36	0.34	0.22	0.34	0.34
0	Naive Bayes	0.35	0.23	0.35	0.28	0.35	0.36
0	Random Forest	0.35	0.23	0.35	0.28	0.35	0.83
0	Stacking Random Forest	0.83	0.83	0.83	0.83	0.83	NaN
0	Adaboost Model	0.64	0.64	0.64	0.64	0.64	0.65
0	MLP	0.70	0.70	0.70	0.70	0.70	0.33

The best classifier for my data was stacking random forest with accuracy of 83%. It also had high precision and recall. The second best being the decision tree with 80% accuracy. We can improve the performance of classification by increasing the data and based on that, we can tweak the classes used for classification. Also in the case of neural networks, if we could have increased the depth or to be precise the number of layers, we could have got the better result. Also for other methods not working, the dataset might not work good with that classifiers.

Phase IV: Clustering and Association

a. Silhoutte Analysis for K-selection

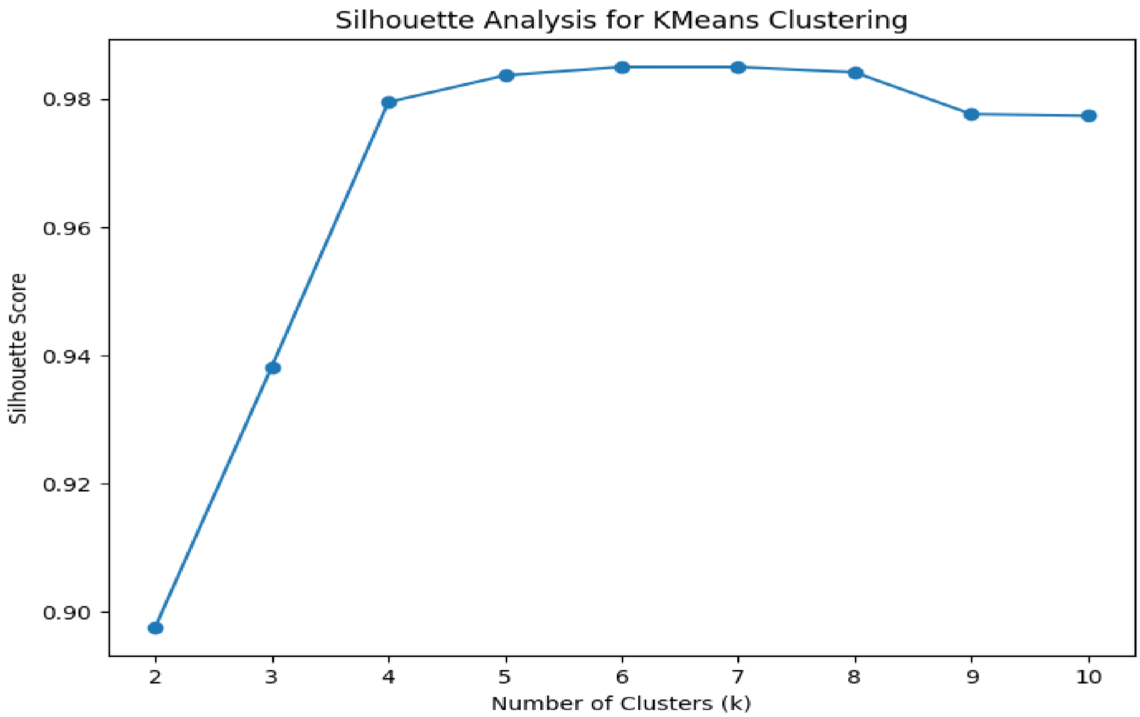

Figure 22 Silhoutte method for k

b. Elbow method for K-selection

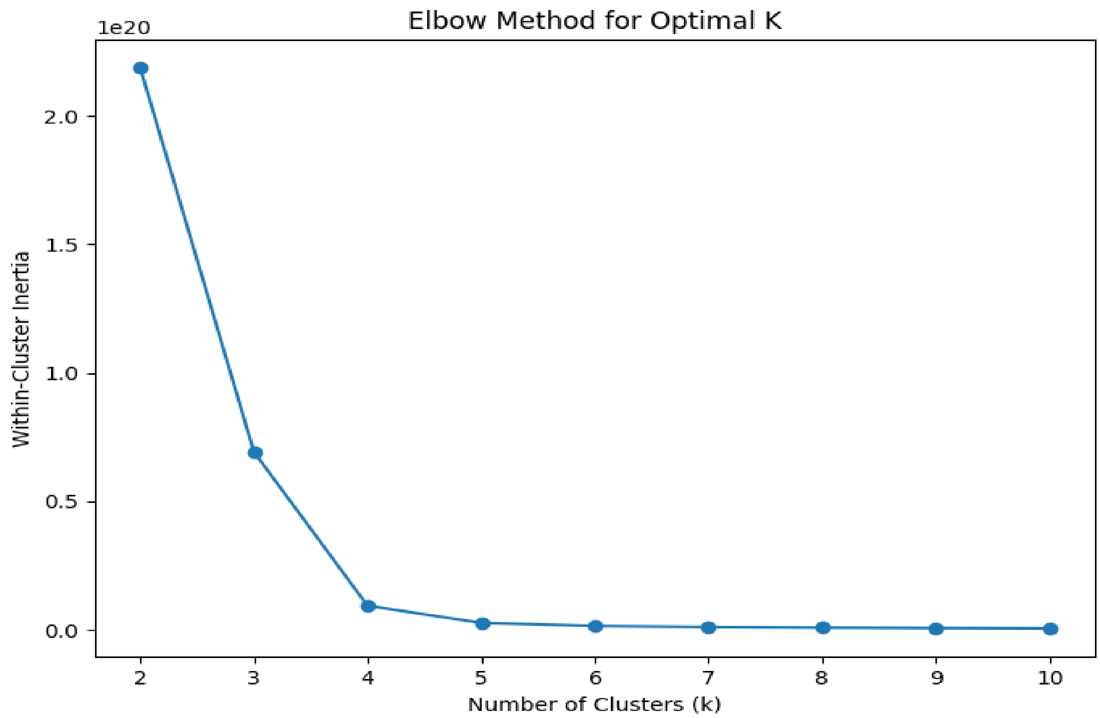

Figure 23 elbow method for k

So based on both the value of k =6

c. K-Means Output

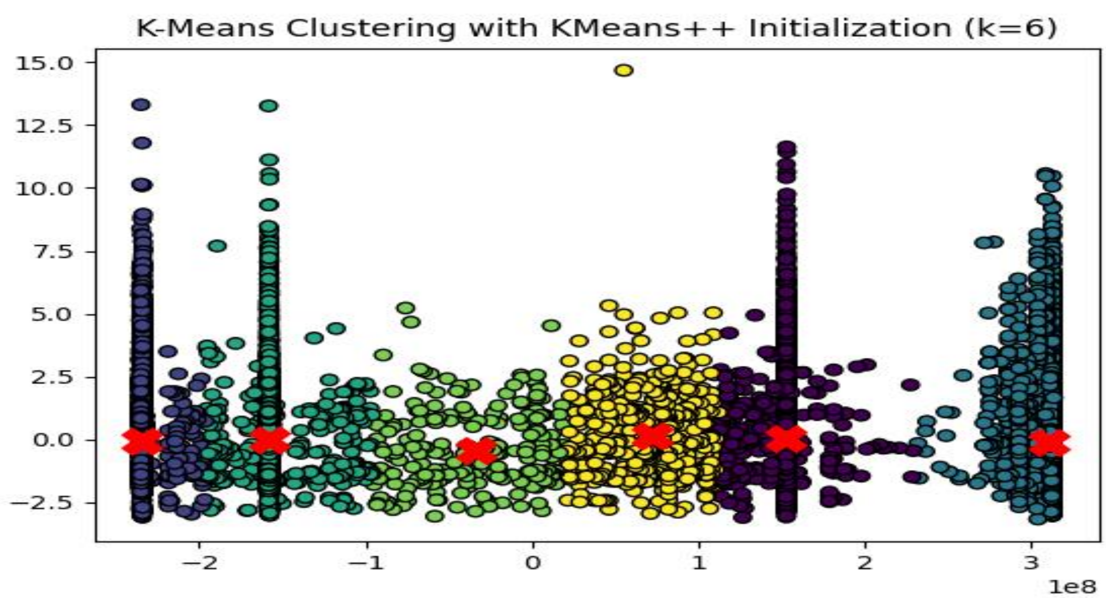

Figure 24 K-means ++ clustering

d. Apriori Algorithm

Output:
Frequent Itemsets:

	support	itemsets
0	0.75	(0.0)
1	0.75	(1.0)
2	1.00	(2.0)
3	0.75	(0.0, 1.0)
4	0.75	(0.0, 2.0)
5	0.75	(1.0, 2.0)
6	0.75	(0.0, 1.0, 2.0)

Association Rules:

	antecedents	consequents	antecedent support	consequent support	support \
0	(0.0)	(1.0)	0.75	0.75	0.75
1	(1.0)	(0.0)	0.75	0.75	0.75
2	(0.0)	(2.0)	0.75	1.00	0.75
3	(2.0)	(0.0)	1.00	0.75	0.75
4	(1.0)	(2.0)	0.75	1.00	0.75
5	(2.0)	(1.0)	1.00	0.75	0.75
6	(0.0, 1.0)	(2.0)	0.75	1.00	0.75
7	(0.0, 2.0)	(1.0)	0.75	0.75	0.75
8	(1.0, 2.0)	(0.0)	0.75	0.75	0.75
9	(0.0)	(1.0, 2.0)	0.75	0.75	0.75
10	(1.0)	(0.0, 2.0)	0.75	0.75	0.75
11	(2.0)	(0.0, 1.0)	1.00	0.75	0.75

	confidence	lift	leverage	conviction	zhangs_metric
0	1.00	1.333333	0.1875	inf	1.0
1	1.00	1.333333	0.1875	inf	1.0
2	1.00	1.000000	0.0000	inf	0.0
3	0.75	1.000000	0.0000	1.0	0.0
4	1.00	1.000000	0.0000	inf	0.0
5	0.75	1.000000	0.0000	1.0	0.0
6	1.00	1.000000	0.0000	inf	0.0
7	1.00	1.333333	0.1875	inf	1.0
8	1.00	1.333333	0.1875	inf	1.0
9	1.00	1.333333	0.1875	inf	1.0
10	1.00	1.333333	0.1875	inf	1.0
11	0.75	1.000000	0.0000	1.0	0.0

Based on the Apriori algorithm, we interpret that when a certain number of bedrooms (represented by '0.0', '1.0', '2.0') is present, it is associated with a price_category.

Recommendations:

It was my first time working on such a large dataset. I learned how to handle large data along with it, how different models behave with the I learned the different machine learning concepts like regression and classification. In this cluttered market of renting, I came across how location, number of bedrooms, etc. can significantly affect the cost of renting the apartment.

The best classifier for my data was stacking random forest with accuracy of 83%. It also had high precision and recall. The second best being the decision tree with 80% accuracy. We can improve the performance of classification by increasing the data and based on that, we can tweak the classes used for classification. MLP could have given better results if we had increased the depth of the MLP classifier.

Based on the Apriori algorithm, we interpret that when a certain number of bedrooms (represented by '0.0', '1.0', '2.0') is present, it is associated with a price_category.

Based on Silhouette and Elbow method, for k-means there are 6 clusters in the dataset.

Code Appendix and References

Code:

```
from sklearn import tree
import pandas as pd
import warnings
import numpy as np
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error, confusion_matrix, precision_score,
recall_score, f1_score, accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, MinMaxScaler,
StandardScaler
import statsmodels.api as sm
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier,
AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.tree import plot_tree, DecisionTreeClassifier
from statsmodels.stats.outliers_influence import variance_inflation_factor
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from mlxtend.classifier import StackingClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from mlxtend.frequent_patterns import apriori, association_rules
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
```

```
warnings.filterwarnings("ignore")




def plot_multiclass_roc_curve(classifier, X_test, y_test):


    y_scores = classifier.predict_proba(X_test)

    plt.figure(figsize=(8, 8))

    for i in range(len(classifier.classes_)):
        fpr, tpr, _ = roc_curve(y_test == classifier.classes_[i], y_scores[:, i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'Class {classifier.classes_[i]} (AUC = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], color='navy', linestyle='--', linewidth=2, label='Random',
alpha=0.8)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve (Multiclass)')
    plt.legend(loc='lower right')
    plt.show()

"""Loading the Data"""

paths = "apartments_for_rent_classified_100K.csv"
df_initial_loaded = pd.read_csv(paths, encoding='ISO-8859-1', sep=';',low_memory=False)
# Selected features
selected_features = ['id','category', 'fee', 'has_photo', 'pets_allowed', 'price_type',
'bathrooms', 'bedrooms', 'square_feet', 'latitude', 'longitude', 'price']

# DataFrame with selected features
df = df_initial_loaded[selected_features]
#.iloc[np.random.randint(1, len(df_initial_loaded), 1000)]
print(df.head().to_string())
print("Length of DataFrame:",len(df))
df2=df

"""PHASE 1"""


#####
#   #
# # #
#
#      1

###################################################
######### Checking the missing values   ########
###################################################
```

```
print("Missing Observations:",df.isnull().sum())

# When we check the missing values we find that most of none values are with
pets_allowed, so replaced it with "NotAllowed"
print("Replacing the Null of pets_allowed with NotAllowed:")
df['pets_allowed'].fillna('NotAllowed', inplace=True)
print("Missing Observations:",df.isnull().sum())
# Drop the null values
print("Dropping the Null values")
df.dropna(inplace=True)
print("Missing Observations:",df.isnull().sum())
###################################################
## Checking for the duplicates and removing them  ##
###################################################

duplicates = df.duplicated()
print("Duplicate Rows:")
print(df[duplicates])
df = df.drop_duplicates()
print(df)

# No Aggregation and the down sampling

###################################################
#########  Discretization & Binarization ########
###################################################

# One hot encoding for the fee feature:
onehot_encoder = OneHotEncoder(sparse=False, drop='first')
onehot_encoded_fee = onehot_encoder.fit_transform(df[['fee']])
df['fee'] = onehot_encoded_fee
print(df.head())
# Label encode 'category', 'has_photo', 'pets_allowed', 'price_type'
label_encoder = LabelEncoder()
df['category'] = label_encoder.fit_transform(df['category'])
df['has_photo'] = label_encoder.fit_transform(df['has_photo'])
df['pets_allowed'] = label_encoder.fit_transform(df['pets_allowed'])
df['price_type'] = label_encoder.fit_transform(df['price_type'])
print("\nLabel-Encoded Data:")
print(df.head())

###################################################
######### Variable Transformation  ########
###################################################

numerical_features = ['bathrooms', 'bedrooms', 'square_feet', 'latitude', 'longitude']
scaler = MinMaxScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# We will not do differencing because this is not a time series data, in which case it is
generally useful

###################################################
## Dimensionality reduction/feature selection #####
###################################################

X = df.drop('price', axis=1)
```

```python
y = df['price']

# Random Forest
rf_model = RandomForestRegressor()
rf_model.fit(X, y)
feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
selected_features_rf = feature_importances[feature_importances > 0.01].index.tolist()
print("Selected Features from Random Forest Analysis:")
print(selected_features_rf)

# PCA and Condition Number
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
# Check the condition number before and after PCA
condition_number_before_pca = np.linalg.cond(X)
condition_number_after_pca = np.linalg.cond(X_pca)
feature_importances_pca = np.abs(pca.components_)
explained_variance_ratio = pca.explained_variance_ratio_
threshold_variance_ratio = 0.01  # Adjust the threshold
selected_features_pca = X.columns[explained_variance_ratio > threshold_variance_ratio]
print("Selected Features based on PCA:")
print(selected_features_pca)

# # Singular Value Decomposition (SVD) Analysis
feature_names = X.columns.tolist()
X_train, X_test = train_test_split(X, test_size=0.2, random_state=5805)
n_components = 5  # Number of components to retain
svd = TruncatedSVD(n_components=n_components)
X_train_svd = svd.fit_transform(X_train)
X_test_svd = svd.transform(X_test)
first_component = svd.components_[0]
feature_contributions = pd.DataFrame({
    'Feature': feature_names,
    'Contribution': first_component
})
feature_contributions =
feature_contributions.reindex(feature_contributions['Contribution'].abs().sort_values(asc
ending=False).index)
print("Selected Features and Their Contributions in the SVD Analysis:")
print(feature_contributions)



# VIF for Collinearity Check
X_scaled_with_intercept = sm.add_constant(X_scaled)
vif_data = pd.DataFrame()
vif_data["Variable"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X_scaled_with_intercept, i) for i in
range(1, X_scaled_with_intercept.shape[1])]
# Remove highly correlated features based on VIF
selected_features_vif = vif_data[vif_data["VIF"] < 5]["Variable"].tolist()
print("\nSelected Features from VIF:")
print(selected_features_vif)


####################################################
# We will use the features selected by Random Forest and use those features moving
```

```
forward
###################################################
df = df[selected_features_rf + ['price']]
print("Feature Reduced Data:\n",df.head().to_string)


###################################################
########## Outlier detection  ########
###################################################

Q1 = df['price'].quantile(0.25)
Q3 = df['price'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_no_outliers = df[(df['price'] >= lower_bound) & (df['price'] <= upper_bound)]
print("Original Dataset Shape:", df.shape)
print("Dataset Shape After Outlier Removal:", df_no_outliers.shape)
df = df_no_outliers


###################################################
######### covariance matrix  ########
###################################################

covariance_matrix = np.cov(X_scaled, rowvar=False)
plt.figure(figsize=(10, 8))
sns.heatmap(covariance_matrix, annot=True, fmt=".4f", cmap="coolwarm",
xticklabels=X.columns, yticklabels=X.columns)
plt.title('Sample Covariance Matrix Heatmap')
plt.show()


###################################################
# sample Pearson correlation coefficients matrix  #
###################################################

correlation_matrix = X.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
xticklabels=correlation_matrix.columns, yticklabels=correlation_matrix.columns)
plt.title('Sample Pearson Correlation Coefficients Matrix Heatmap')
plt.show()


# Checking whether the dataset is balanced
sns.histplot(df['price'], bins=3, kde=True)
plt.title('Distribution of Prices')
plt.xlabel('Price')
plt.show()
# It looks like the most of the data is in the middle range, but this is common on
renting industry and hence will not balance the data.

"""PHASE 2"""

#####
#   #
# # #
#
```

```
#       2

X = df.drop('price', axis=1)
y = df['price']


X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y, test_size=0.2,
random_state=5805)


X_train_reg = sm.add_constant(X_train_reg)
X_test_reg = sm.add_constant(X_test_reg)

final_model = sm.OLS(y_train_reg, X_train_reg).fit()

# T-test analysis for each variable
t_test_results = final_model.t_test(np.eye(len(final_model.params)))
print("T-test analysis:")
print(t_test_results)

# F-test analysis
f_test_result = final_model.f_test(np.eye(len(final_model.params)))
print("\nF-test analysis:")
print(f_test_result)

# Prediction on train and test sets
y_train_pred = final_model.predict(X_train_reg)
y_test_pred = final_model.predict(X_test_reg)

# Plotting train, test, and predicted variables
plt.scatter(range(len(y_train_reg)), y_train_reg, label='Train Actual Prices')
plt.scatter(range(len(y_train_reg), len(y_train_reg) + len(y_test_reg)), y_test_reg,
label='Test Actual Prices')
plt.scatter(range(len(y_train_reg)), y_train_pred, label='Train Predicted Prices')
plt.scatter(range(len(y_train_reg), len(y_train_reg) + len(y_test_reg)), y_test_pred,
label='Test Predicted Prices')
plt.xlabel('Observation Index')
plt.ylabel('Price')
plt.legend()
plt.show()

# R-squared, adjusted R-square, AIC, BIC, and MSE for train set
r2_train = final_model.rsquared
adj_r2_train = final_model.rsquared_adj
aic_train = final_model.aic
bic_train = final_model.bic
mse_train = mean_squared_error(y_train_reg, y_train_pred)

# R-squared, adjusted R-square, AIC, BIC, and MSE for test set
r2_test = final_model.rsquared
adj_r2_test = final_model.rsquared_adj
aic_test = final_model.aic
bic_test = final_model.bic
mse_test = mean_squared_error(y_test_reg, y_test_pred)

plt.figure(figsize=(10, 6))
plt.scatter(y_train_reg, y_train_pred, label="Train Predicted Price", alpha=0.7)
plt.scatter(y_train_reg, y_train_reg, label="Train Price", alpha=0.7)
plt.scatter(y_test_reg, y_test_pred, label="Test Predicted Price", alpha=0.7)
```

```python
plt.scatter(y_test_reg, y_test_reg, label="Test Price", alpha=0.7)
plt.xlabel("Test Data Price")
plt.ylabel("Predicted Data Price")
plt.legend()
plt.show()

print(f"\nTrain Set Metrics:")
print(f"R-squared: {r2_train}")
print(f"Adjusted R-squared: {adj_r2_train}")
print(f"AIC: {aic_train}")
print(f"BIC: {bic_train}")
print(f"MSE: {mse_train}")

print(f"\nTest Set Metrics:")
print(f"R-squared: {r2_test}")
print(f"Adjusted R-squared: {adj_r2_test}")
print(f"AIC: {aic_test}")
print(f"BIC: {bic_test}")
print(f"MSE: {mse_test}")

# Confidence interval analysis
conf_int = final_model.conf_int()
print("\nConfidence Intervals for Coefficients:")
print(conf_int)

# printing the equation
coefficients = final_model.params
linear_regression_equation = f"Price = {coefficients['const']:.2f}"

for feature, coefficient in coefficients.items():
    if feature != 'const':
        linear_regression_equation += f" + {coefficient:.2f} * {feature}"

def forward_selected(data, response):
    remaining = set(data.columns)
    remaining.remove(response)
    selected = []
    current_score, best_new_score = float('inf'), float('inf')
    while remaining and current_score == best_new_score:
        scores_with_candidates = []
        for candidate in remaining:
            formula = "{} ~ {}".format(response, ' + '.join(selected + [candidate]))
            score = sm.formula.ols(formula, data).fit().rsquared_adj
            scores_with_candidates.append((score, candidate))
        scores_with_candidates.sort()
        best_new_score, best_candidate = scores_with_candidates.pop()
        if current_score > best_new_score:
            remaining.remove(best_candidate)
            selected.append(best_candidate)
            current_score = best_new_score
    formula = "{} ~ {}".format(response, ' + '.join(selected))
    model = sm.formula.ols(formula, data).fit()
    return model




print("Linear Regression Equation:")
print(linear_regression_equation)
```

```
forward_selected_model = forward_selected(df, 'price')

print(forward_selected_model.summary())

# Adjusted R-square analysis
adj_r2_values = []
num_features = range(1, len(selected_features_rf) + 1)

for num in num_features:
    selected_features_adj_r2 = selected_features_rf[:num]
    formula_adj_r2 = "price ~ {}".format(' + '.join(selected_features_adj_r2))
    model_adj_r2 = sm.formula.ols(formula_adj_r2, df).fit()
    adj_r2_values.append(model_adj_r2.rsquared_adj)

plt.plot(num_features, adj_r2_values, marker='o')
plt.title('Adjusted R-square Analysis')
plt.xlabel('Number of Features')
plt.ylabel('Adjusted R-square')
plt.show()



"""PHASE 3"""

#####
#   #
# # #
#
#      3

###################################################
######## Creating Classification Feature   ########
###################################################



# df['price_category'], bin_edges = pd.qcut(df['price'], q=3, labels=[0,1,2],
retbins=True) #30000 in each part
df['price_category'], bin_edges = pd.qcut(df['price'], q=3,
labels=['Cheap','Affordable','Costly'], retbins=True) #30000 in each part



print(df.head())
print(len(df))
# length of each category
category_lengths = df['price_category'].value_counts()
print("\nLength of each category:")
print(category_lengths)

# Plot a bar plot for the distribution of price categories
plt.figure(figsize=(8, 6))
category_lengths.plot(kind='bar', color=['blue', 'green', 'red'], edgecolor='black')
plt.title('Distribution of Price Categories')
plt.xlabel('Price Category')
plt.ylabel('Count')
plt.show()
```

```
# Print starting and ending values of each category
for i in range(len(bin_edges) - 1):
    start_value = bin_edges[i]
    end_value = bin_edges[i + 1]
    category = df['price_category'].cat.categories[i]
    print(f"{category} category: {start_value} to {end_value}")

label_encoder = LabelEncoder()
df['price_category'] = label_encoder.fit_transform(df['price_category'])


classification_table = pd.DataFrame(columns=['Model', 'Accuracy', 'Precision', 'Recall',
'F1 Score', 'Specificity', 'Cross Validation Mean Score'])


X = df.drop(['price', 'price_category'], axis=1)
# Exclude 'price' and 'price_category' from features
y = df['price_category']

# ##################################################
# #############     Decision Tree    ###############
# ##################################################




X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)

dt_classifier = DecisionTreeClassifier(random_state=5805)

# hyperparameters
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'max_features': [None, 'sqrt', 'log2'],
    'ccp_alpha': [0.0, 0.01, 0.1, 0.2, 0.5]
}

# Grid Search with 5-fold cross-validation
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring='accuracy', n_jobs=-
1)
grid_search.fit(X_train, y_train)
dt_classifier.fit(X_train, y_train)
y_pred = dt_classifier.predict(X_test)
classification_rep = classification_report(y_test, y_pred)
plt.figure(figsize=(15, 10))
plot_tree(dt_classifier, feature_names=X.columns, class_names=np.unique(y).astype(str),
filled=True, rounded=True)
plt.show()
scores = cross_val_score(dt_classifier, X_train, y_train, cv=5, scoring='accuracy')


classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Decision
Tree',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
```

```
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(dt_classifier, X_test, y_test)

# best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

print("Prepruning Decision Tree:")

best_dt_classifier = DecisionTreeClassifier(**best_params, random_state=5805)
best_dt_classifier.fit(X_train, y_train)

y_pred = best_dt_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
plt.figure(figsize=(15, 10))
plot_tree(best_dt_classifier, feature_names=X.columns,
class_names=np.unique(y).astype(str), filled=True, rounded=True)
plt.show()

scores = cross_val_score(best_dt_classifier, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Prepruned
Decision Tree',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(best_dt_classifier, X_test, y_test)

# alpha for post pruning
print("Post pruning Decision Tree")
clf = DecisionTreeClassifier(random_state=5805)
path = clf.cost_complexity_pruning_path(X_train,y_train)
alphas = np.linspace(0, 0.005, 100)
#==============================
# Grid search for best alpha
#==============================
accuracy_train, accuracy_test = [],[]
for i in alphas:
```

```
    clf_post = DecisionTreeClassifier(ccp_alpha=i)
    clf_post.fit(X_train, y_train)
    y_train_pred = clf_post.predict(X_train)
    y_test_pred = clf_post.predict(X_test)
    accuracy_train.append(accuracy_score(y_train, y_train_pred))
    accuracy_test.append(accuracy_score(y_test, y_test_pred))
fig, ax = plt.subplots()
ax.set_xlabel('alpha')
ax.set_ylabel('accuracy')
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(alphas, accuracy_train, marker="o", label="train",
drawstyle="steps-post")
ax.plot(alphas, accuracy_test, marker="o", label="test",
drawstyle="steps-post")
ax.legend()
plt.grid()
plt.tight_layout()
plt.show()

# ========================
# So best alpha is 0.0001
#==========================
clf_post_final = DecisionTreeClassifier(random_state=5805, ccp_alpha=0.0001)
clf_post_final.fit(X_train, y_train)
y_train_pred = clf_post_final.predict(X_train)
y_test_pred = clf_post_final.predict(X_test)
print("Post pruning Decision Tree")
print(f'Train accuracy {accuracy_score(y_train, y_train_pred):.2f}')
print(f'Test accuracy {accuracy_score(y_test, y_test_pred):.2f}')
plt.figure(figsize=(16,8))
tree.plot_tree(clf_post_final, rounded=True, filled=True)
plt.show()
y_pred = clf_post_final.predict(X_test)
scores = cross_val_score(clf_post_final, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model':
'Postpruning Decision Tree',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)

plot_multiclass_roc_curve(clf_post_final, X_test, y_test)

###################################################
#############   Logistic Regression   ###############
###################################################


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)
Multiclassmodel = LogisticRegression(multi_class='ovr',random_state=5805)
Multiclassmodel.fit(X_train, y_train)
```

```
y_pred = Multiclassmodel.predict(X_test)
probs_y = Multiclassmodel.predict_proba(X_test)
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g',
xticklabels=np.unique(y_test),
yticklabels=np.unique(y_test))
plt.title('Logistic Regression Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()
plt.show()
#====================================
# Calculating micro and macro precision
#=========================================
# ==================
# Precision
#=================
macro_averaged_precision = metrics.precision_score(y_test, y_pred,
average='macro')
micro_averaged_precision = metrics.precision_score(y_test
                                                   , y_pred,
average='micro')
print(f"Logistic Regression Macro-Averaged Precision score using sklearn "
f"library : {macro_averaged_precision:.2f}")
print(f"Logistic Regression Micro-Averaged Precision score using sklearn "
f"library : {micro_averaged_precision:.2f}")
# ==================
# Recall
#=================
macro_averaged_recall = metrics.recall_score(y_test, y_pred,
average = 'macro')
micro_averaged_recall = metrics.recall_score(y_test, y_pred,
average = 'micro')
print(f"Logistic Regression Macro-averaged recall score using "
f"sklearn : {macro_averaged_recall:.2f}")
print(f"Logistic Regression Micro-averaged recall score using "
f"sklearn : {micro_averaged_recall:.2f}")
# ==================
# f1 score
#=================
macro_averaged_f1 = metrics.f1_score(y_test, y_pred, average = 'macro')
print(f"Logistic Regression Macro-Averaged F1 score using sklearn library"
f" : {macro_averaged_f1:.2f}")
micro_averaged_f1 = metrics.f1_score(y_test, y_pred, average = 'micro')
print(f"Logistic Regression Micro-Averaged F1 score using sklearn library "
f": {micro_averaged_f1:.2f}")

scores = cross_val_score(Multiclassmodel, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Logistic
Regression',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
```

```
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(Multiclassmodel, X_test, y_test)

###################################################
#############   KNN   ###############
###################################################

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)
error_rate = []
for i in range(1,100):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
plt.figure(figsize=(10,6))
plt.plot(range(1,100),error_rate,color='blue', linestyle='dashed',
marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')

# Find the optimum k
optimal_k = error_rate.index(min(error_rate)) + 1
print(f"Optimal K: {optimal_k}")

final_knn_classifier = KNeighborsClassifier(n_neighbors=optimal_k)
final_knn_classifier.fit(X_train, y_train)

y_pred = final_knn_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with Optimal K: {accuracy:.2f}")

scores = cross_val_score(final_knn_classifier, X_train, y_train, cv=5,
scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'KNN',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(final_knn_classifier, X_test, y_test)

###################################################
#############   SVM   ##############################
###################################################
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)

svc = SVC(probability=True,random_state=5805)
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)

print('Performance of the baseline svm on the test set:')
print('Accuracy of the model:', accuracy_score(y_test, y_pred).round(2))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Precision Score:', precision_score(y_test, y_pred, average='weighted').round(2))
print('Recall Score:', recall_score(y_test, y_pred, average='weighted').round(2))
print('Specificity Score:', recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2))
print('F1 Score:', f1_score(y_test, y_pred, average='weighted').round(2))



scores = cross_val_score(svc, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'SVC',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)

plot_multiclass_roc_curve(svc, X_test, y_test)

svc = SVC(probability=True,random_state=5805)

parameters = [{'kernel': ['rbf', 'poly'],
                    'gamma': [0.1, 1],
                    'C': [0.1, 1]}]

clf = GridSearchCV(svc, parameters, cv=5, scoring='accuracy', n_jobs=-1, verbose=False)
clf.fit(X_train, y_train)

print("Best Estimator: \n", clf.best_estimator_)
print("Best Score: \n", clf.best_score_)
print("Best Paramters: \n", clf.best_params_)

best_svc = clf.best_estimator_
best_svc.fit(X_train, y_train)

y_pred = best_svc.predict(X_test)




scores = cross_val_score(best_svc, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Best
```

```
SVC',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(best_svc, X_test, y_test)



###################################################
#############   Naive Bayes  ######################
###################################################



nb_model = GaussianNB()

nb_model.fit(X_train, y_train)



y_pred = nb_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

scores = cross_val_score(nb_model, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Naive
Bayes',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(nb_model, X_test, y_test)

###################################################
#############  Random Forest  #####################
###################################################


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)
```

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=5805)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("Random Forest Classifier:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_rf):.2f}')
print(classification_report(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))
print("\n")

scores = cross_val_score(rf_model, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Random
Forest',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(rf_model, X_test, y_test)

label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

# Stacking
# Create base classifiers
classifier1 = RandomForestClassifier(n_estimators=50, random_state=5805)
classifier2 = AdaBoostClassifier(n_estimators=50, random_state=5805)

meta_classifier = LogisticRegression()

# Create a stacking classifier
stacking_model = StackingClassifier(classifiers=[classifier1, classifier2],
                                    meta_classifier=meta_classifier)

stacking_model.fit(X_train, y_train)

y_pred_stacking_encoded = stacking_model.predict(X_test)

# Decoding predicted labels back to original categorical ones
y_pred = label_encoder.inverse_transform(y_pred_stacking_encoded)

# Evaluate Stacking
print("Stacking Classifier:")
print(f'Accuracy: {accuracy_score(y_test, y_pred):.2f}')
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print("\n")

scores = cross_val_score(stacking_model, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Stacking
```

```
Random Forest',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)

# Boosting (AdaBoost)
adaboost_model = AdaBoostClassifier(n_estimators=50, random_state=5805)
adaboost_model.fit(X_train, y_train)
y_pred = adaboost_model.predict(X_test)

# Evaluate AdaBoost
print("AdaBoost Classifier:")
print(f'Accuracy: {accuracy_score(y_test, y_pred):.2f}')
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

scores = cross_val_score(adaboost_model, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'Adaboost
Model',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(adaboost_model, X_test, y_test)

###################################################
#############  Neural Network  ####################
###################################################

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


mlp_model = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500, random_state=5805)
mlp_model.fit(X_train_scaled, y_train)


y_pred = mlp_model.predict(X_test_scaled)
```

```
print("Multi-layered Perceptron (MLP) Classifier:")
print(f'Accuracy: {accuracy_score(y_test, y_pred):.2f}')
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

scores = cross_val_score(mlp_model, X_train, y_train, cv=5, scoring='accuracy')
classification_table = pd.concat([classification_table, pd.DataFrame({'Model': 'MLP',
                                'Accuracy': accuracy_score(y_test, y_pred).round(2),
                                'Precision': precision_score(y_test, y_pred,
average='weighted').round(2),
                                'Recall': recall_score(y_test, y_pred,
average='weighted').round(2),
                                'F1 Score': f1_score(y_test, y_pred,
average='weighted').round(2),
                                'Specificity': recall_score(y_test, y_pred, pos_label=0,
average='weighted').round(2),
                                'Cross Validation Mean Score': scores.mean().round(2)},
index=[0])], axis=0)
plot_multiclass_roc_curve(mlp_model, X_test, y_test)

print("Classification Table:\n",classification_table)

"""PHASE 4"""

#####
#   #
# # #
#
#      4

# Silhouette analysis
X = df.drop(['price','price_category'], axis=1)




silhouette_scores = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=5805)
    kmeans.fit(X)
    labels = kmeans.labels_
    silhouette_avg = silhouette_score(X, labels)
    silhouette_scores.append(silhouette_avg)

# Plotting Silhouette Scores
plt.figure(figsize=(8, 6))
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis for KMeans Clustering')
plt.show()



#Elbow Method
```

```
inertia_values = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=5805)
    kmeans.fit(X)
    inertia_values.append(kmeans.inertia_)

plt.figure(figsize=(8, 6))
plt.plot(range(2, 11), inertia_values, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()


# So finally k=6
k = 6

# Apply KMeans++ initialization
kmeans_model = KMeans(n_clusters=k, init='k-means++', random_state=5805)
kmeans_labels = kmeans_model.fit_predict(X)

# Use PCA for dimensionality reduction to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)


plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis', edgecolor='k')
plt.scatter(pca.transform(kmeans_model.cluster_centers_)[:, 0],
pca.transform(kmeans_model.cluster_centers_)[:, 1], s=200, marker='X', c='red')
plt.title('K-Means Clustering with KMeans++ Initialization (k=6)')
plt.show()

# Access cluster centers and labels
cluster_centers = kmeans_model.cluster_centers_
labels = kmeans_model.labels_

############################################
######### Appriori Algorithm ###############
############################################




transactions_df = df[['bedrooms', 'price_category']]

# dataFrame to a transactional format
transactions =
transactions_df.groupby('bedrooms')['price_category'].apply(list).reset_index(name='cat')

# binary dataset
binary_df = pd.get_dummies(transactions['cat'].apply(pd.Series).stack(), prefix='',
prefix_sep='').groupby(level=0).max()


min_support = 0.1
frequent_itemsets = apriori(binary_df, min_support=min_support, use_colnames=True)

rules = association_rules(frequent_itemsets, metric='confidence', min_threshold=0.7)
```

```python
print("Frequent Itemsets:")
print(frequent_itemsets)

print("\nAssociation Rules:")
print(rules)
```