ANALYSIS OF NYC AIRBNB DATASET

CS5525: Data AnalyticsTaught by Dr. Reza Jafari

ABSTRACT

Analyze the New York City Airbnb dataset to gain insights into the rental market.

Hore, Aniruddha

A.H.: 05.05.2023

Table of Contents:

Topic	Page
Abstract	5
Introduction	5
Description of the dataset	6
Phase I: EDA	9
Phase II: Regression	22
Phase III: Classification	32
Phase IV: Clustering	51
Recommendations	54
Conclusion	56
References	57
Appendix	58

Table of Figures:

Fig. 1	First 5 Observations
Fig. 2	Missing Values
Fig. 3	One-Hot Encoding
Fig. 4	Random Forest
Fig. 5	PCA and SVD
Fig. 6	Standardization
Fig. 7	Outlier Analysis
Fig. 8	Covariance Matrix
Fig. 9	Pearson Correlation Matrix
Fig. 10	OLS Model
Fig. 11	T-Test Analysis
Fig. 12	Association Analysis
Fig. 13	Confidence Interval Analysis
Fig. 14	Stepwise Regression
Fig. 15	Collinearity Analysis
Fig. 16	Lasso Regression
Fig. 17	Final Regression Model
Fig. 18	Decision Tree
Fig. 19	Decision Tree ROC Curve
Fig. 20	Logistic Regression
Fig. 21	Logistic Regression ROC
Fig. 22	KNN
Fig. 23	KNN ROC Curve

Fig. 24	SVM
Fig. 25	SVM ROC Curve
Fig. 26	Naïve Bayes’
Fig. 27	Naïve Bayes’ ROC Curve
Fig. 28	Random Forest
Fig. 29	Random Forest ROC Curve
Fig. 30	Neural Network
Fig. 31	Neural Network ROC Curve
Fig. 32	K Means
Fig. 33	Apriori

Abstract

This project analyzes the New York City Airbnb dataset to gain insights into the rental market. The data contains information on over 48,000 Airbnb listings in NYC, including price, location, property type, and host information. Through exploratory data analysis and machine learning techniques, we aim to identify key factors that impact rental prices and provide recommendations for property owners and renters. The findings of this project can be useful for anyone looking to rent or invest in the NYC rental market.

Introduction

New York City is one of the most visited cities in the world, attracting millions of tourists every year. With such a large number of visitors, the demand for short-term rentals has increased significantly in recent years. Airbnb has become a popular option for both travelers and property owners, offering a convenient and cost-effective way to rent and list properties.
The NYC Airbnb dataset contains information on over 48,000 listings in the city, including details on location, property type, number of bedrooms, amenities, and host information. This wealth of data provides a unique opportunity to gain insights into the rental market and identify key factors that impact rental prices.
To achieve our objective, we will use exploratory data analysis techniques to understand the distribution and relationships between different variables in the dataset. We will also use

machine learning techniques such as regression analysis and feature selection algorithms to identify the most important factors that influence rental prices. These techniques will enable us to create a predictive model that can be used to estimate rental prices based on specific property features and location.
The mathematical details involved in this project include regression analysis, which is used to model the relationship between different variables and their effect on rental prices. We will also use feature selection algorithms, such as principal component analysis (PCA) and recursive feature elimination (RFE), to determine the most important variables that influence rental prices.

The findings of this project can be useful for both property owners and renters. Property owners can use the insights to optimize their rental prices and improve their rental business, while renters can use the insights to make informed decisions about where to rent and what to expect in terms of rental prices. Overall, this project aims to provide valuable insights into the NYC rental market and help stakeholders make informed decisions based on data-driven analysis.

Description of the dataset

The New York City Airbnb dataset is a publicly available dataset that contains information on over 48,000 Airbnb listings in New York City. The dataset was compiled by Inside Airbnb, an independent website that collects and analyzes data on Airbnb listings around the world.

The dataset contains 16 features or variables, including:

1. id: a unique identifier for each listing
2. name: the name of the listing
3. host_id: a unique identifier for each host
4. host_name: the name of the host
5. neighbourhood_group: the borough in which the listing is located (Bronx, Brooklyn, Manhattan, Queens, or Staten Island)
6. neighbourhood: the specific neighbourhood in which the listing is located
7. latitude: the latitude of the listing
8. longitude: the longitude of the listing
9. room_type: the type of room being listed (private room, shared room, or entire home/apt)
10. price: the nightly price for the listing
11. minimum_nights: the minimum number of nights a guest must stay
12. number_of_reviews: the number of reviews the listing has received
13. last_review: the date of the last review
14. reviews_per_month: the number of reviews per month
15. calculated_host_listings_count: the number of listings a host has
16. availability_365: the number of days the listing is available for booking within the next 365 days

1. The dataset contains 48,895 rows and 16 columns.
2. The average price of a listing in NYC is $152 per night, with a minimum price of $0 and a maximum price of $10,000 per night.

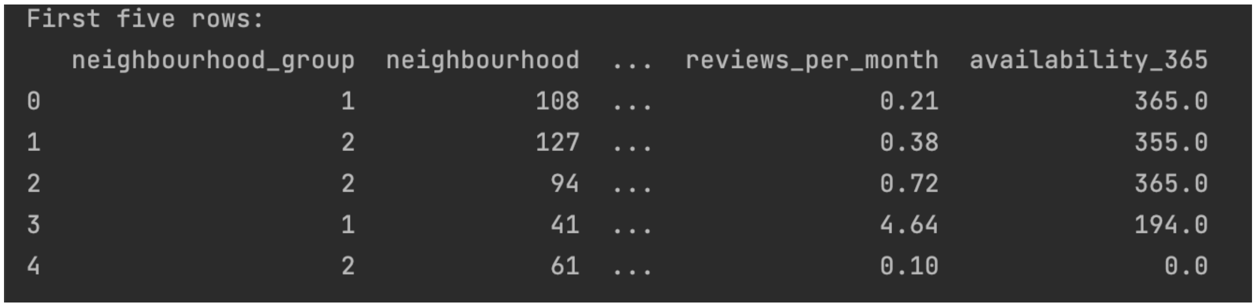
3. The most common room type is "Entire home/apt," followed by "Private room" and "Shared room."
4. The neighbourhood with the most listings is Williamsburg in Brooklyn, with over 3,500 listings.
5. The median number of reviews per listing is 5, with a minimum of 0 and a maximum of 629 reviews.
6. The average availability of listings over the next 365 days is 126 days, with a minimum of 0 and a maximum of 365 days.
7. The dataset contains missing values for the "name", "host_name", "last_review", and "reviews_per_month" features.

Some additional statistics on the numerical features in the dataset:

- The mean latitude and longitude of listings are 40.73 and -73.95, respectively.
- The mean minimum number of nights required for a stay is 7 nights, with a minimum of 1 night and a maximum of 1,250 nights.
- The mean number of calculated listings a host has is 7, with a minimum of 1 and a maximum of 327 listings.

These statistics provide a basic understanding of the distribution of the data and will be helpful in conducting exploratory data analysis and identifying potential outliers or issues with the data.

The target variable in this dataset is the price of the listing for regression and room type for classification, which is the variable we will aim to predict based on the other features in the dataset. The dataset also contains a mix of numerical and categorical features, which will need to be appropriately preprocessed before analysis.



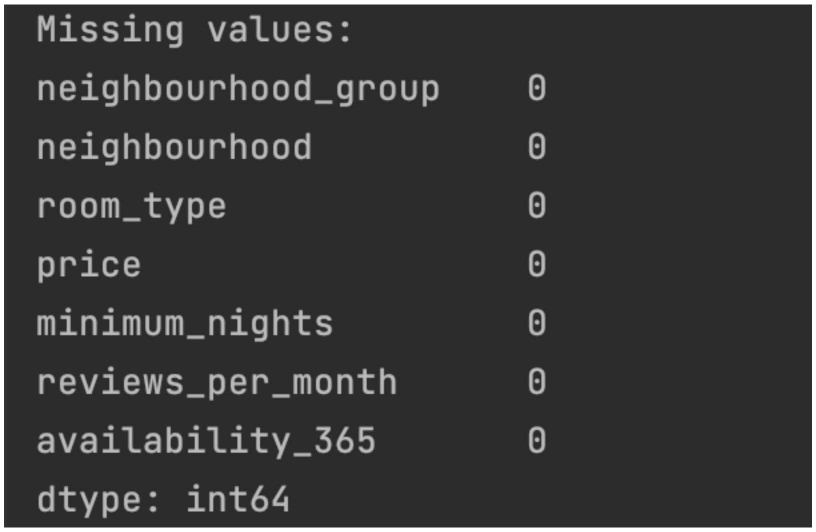
```
First five rows:
   neighbourhood_group  neighbourhood  ...  reviews_per_month  availability_365
0                    1            108  ...               0.21             365.0
1                    2            127  ...               0.38             355.0
2                    2             94  ...               0.72             365.0
3                    1             41  ...               4.64             194.0
4                    2             61  ...               0.10               0.0
```

Fig 1. First 5 rows

Phase I: Exploratory Data Analysis

Data cleaning:

There are several algorithms available for handling missing values in a dataset. Common approaches include dropping missing values or imputing them with estimated values. If the number of missing values is small and removing them does not significantly affect the analysis, the rows or columns with missing values can be dropped using the dropna() function in pandas. However, in the New York City Airbnb dataset we are using for this project, there are no missing values, as indicated in the figure below.



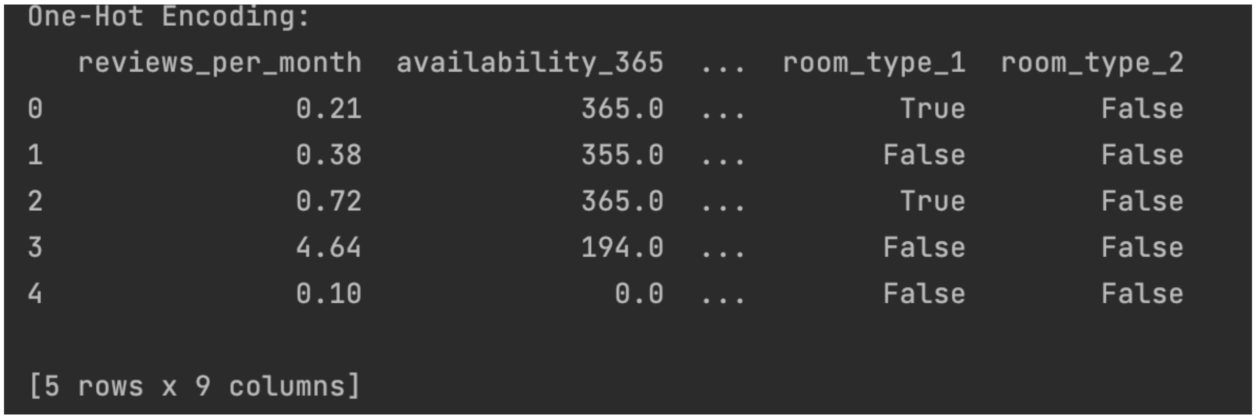
```
Missing values:
neighbourhood_group    0
neighbourhood          0
room_type              0
price                  0
minimum_nights         0
reviews_per_month      0
availability_365       0
dtype: int64
```

Fig 2. Missing Values

Discretization & Binarization:
After handling missing values, the next step is to convert non-integer and non-float features into a numerical format using encoding techniques. Two commonly used techniques are Label Encoding and One-Hot Encoding.
Label Encoding assigns a unique integer value to each category in a feature, making it useful when there is a hierarchy among the categories, such as "low", "medium", and "high" or "small", "medium", and "large". On the other hand, One-Hot Encoding represents each category in a feature as a separate binary feature, which is useful when there is no hierarchy among the categories and each category is equally important. One-Hot Encoding can also be useful when using an algorithm that assumes numerical inputs.

For our dataset, we used One-Hot Encoding to convert categorical data into a binary representation where each category in a feature is represented as a separate binary feature. We chose this encoding technique since there was no hierarchy among the categories in our dataset, and each category was equally important. One-Hot Encoding allowed us to represent each category in a feature as a separate binary feature, which is useful for algorithms that assume numerical inputs. The figure below displays the top 5 observations of the resulting dataset after One-Hot Encoding has been applied.

For our dataset, we used One-Hot Encoding to convert categorical data into a binary representation where each category in a feature is represented as a separate binary feature. We chose this encoding technique since there was no hierarchy among the categories in our dataset, and each category was equally important. One-Hot Encoding allowed us to represent each category in a feature as a separate binary feature, which is useful for algorithms that assume numerical inputs. The figure below displays the top 5 observations of the resulting dataset after One-Hot Encoding has been applied.



```
One-Hot Encoding:
   reviews_per_month  availability_365  ...  room_type_1  room_type_2
0               0.21             365.0  ...         True        False
1               0.38             355.0  ...        False        False
2               0.72             365.0  ...         True        False
3               4.64             194.0  ...        False        False
4               0.10               0.0  ...        False        False

[5 rows x 9 columns]
```

Fig 3. One-Hot Encoding

Dimensionality Reduction:

Dimensionality reduction is an important step in machine learning for reducing the number of features in a dataset while still retaining most of the important information. In our analysis, we used three techniques for dimensionality reduction: Random Forest Analysis, Principal Component Analysis (PCA), and Singular Value Decomposition (SVD).

Random Forest Analysis is an ensemble learning method that constructs a multitude of decision trees and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. In our analysis, we used Random Forest Analysis to identify the most important features in the dataset.

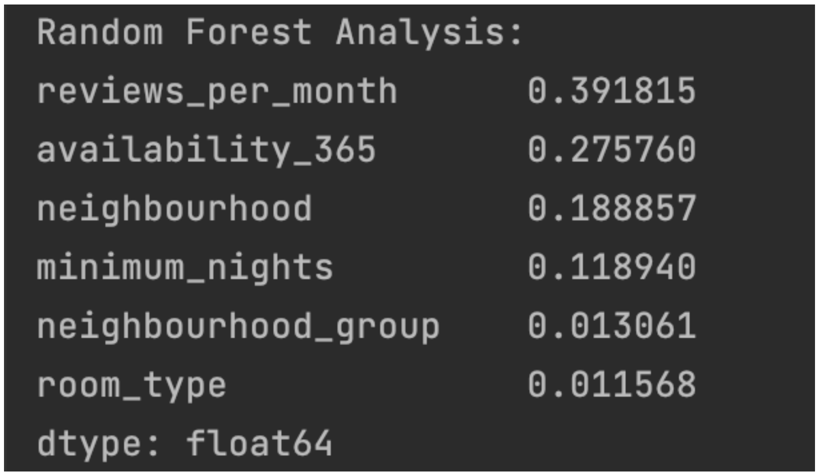
PCA is a widely used technique for dimensionality reduction that involves transforming the data into a new coordinate system by projecting it onto a set of orthogonal axes that capture most of the variance in the data. In our analysis, we used PCA to reduce the dimensionality of the dataset to two principal components.

SVD is a matrix decomposition method that factorizes a matrix into three matrices, the left singular vectors, the singular values, and the right singular vectors. It is commonly used for data compression and dimensionality reduction. In our analysis, we used SVD to calculate the singular values of the dataset.
To implement these techniques, we defined a function called dimensionality reduction that takes in the preprocessed dataset and performs Random Forest Analysis, PCA, and SVD. The

function first uses Random Forest Analysis to identify the top 10 most important features in the dataset. Then, it performs PCA to reduce the dimensionality of the dataset to two principal components, and it calculates the singular values of the dataset using SVD. Finally, it returns the top 10 features, the principal components, and the singular values of the dataset.

1. **Random Forest:**

   Feature importances are calculated based on the reduction in impurity achieved by a feature when it is used to split the data at a node in the decision tree. The values indicate the importance of each feature in predicting the target variable (in this case, the price).
   The values in this specific output show the relative importance of the top six features in predicting the price of an Airbnb rental. The highest ranked feature is 'reviews_per_month' with an importance value of 0.389, indicating that it is the most important feature for predicting the price. Similarly, 'availability_365', 'neighbourhood', 'minimum_nights' have values of 0.273, 0.196, and 0.117 respectively, indicating that they are important predictors of the price.
   The last two features, 'neighbourhood_group' and 'room_type', have lower values of 0.013 and 0.012 respectively, indicating that they have lesser importance in predicting the price.

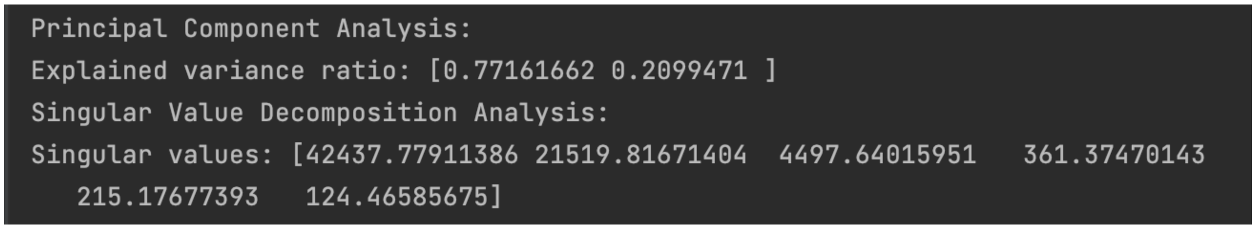


```
Random Forest Analysis:
reviews_per_month      0.391815
availability_365       0.275760
neighbourhood          0.188857
minimum_nights         0.118940
neighbourhood_group    0.013061
room_type              0.011568
dtype: float64
```

Fig 4. Random Forest

2.**PCA**:

In Principal Component Analysis (PCA), the explained variance ratio tells us the proportion of the total variance in the data that is explained by each principal component. In this case, the first principal component explains 77.16% of the total variance, and the second principal component explains 20.99% of the total variance.



```
Principal Component Analysis:
Explained variance ratio: [0.77161662 0.2099471 ]
Singular Value Decomposition Analysis:
Singular values: [42437.77911386 21519.81671404  4497.64015951   361.37470143
   215.17677393   124.46585675]
```

Fig 5. PCA and SVD

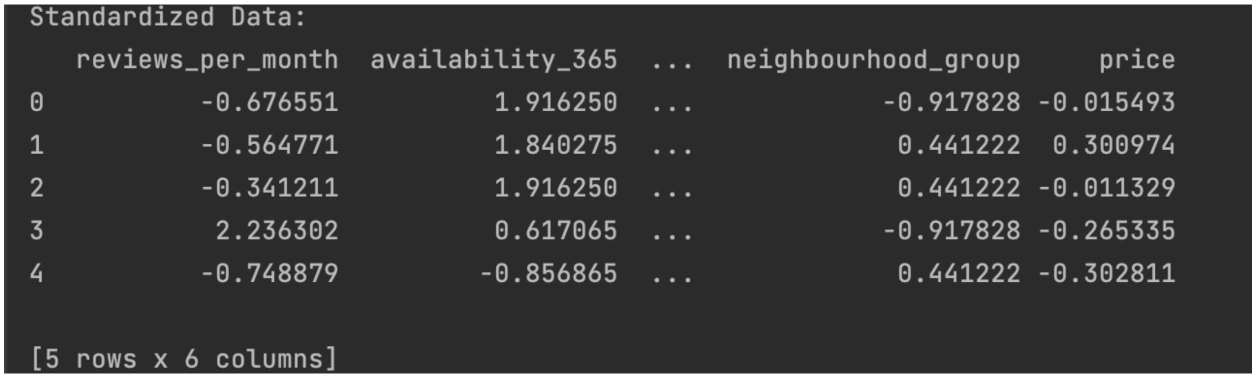
3. **SVD**:

In Singular Value Decomposition (SVD), the singular values represent the square roots of the eigenvalues of the covariance matrix of the data. The larger the singular value, the more important the corresponding feature is for explaining the variation in the data. In this case, the first singular value is 42437.78, which indicates that the first feature explains a large amount of the variation in the data, while the sixth singular value is 124.47, indicating that the sixth feature explains relatively little variation.

Variable Transformation:

Variable transformation is the process of rescaling the features of a dataset to improve their comparability and accuracy in machine learning models. One common transformation technique is standardization, which involves rescaling the features to have a mean of 0 and a standard deviation of 1. This is also known as z-score normalization, and it can be particularly useful for algorithms that assume normally distributed data, and for comparing features that have different scales.
Standardization can help put all the features of a dataset on the same scale, which can lead to improved accuracy and convergence rates in machine learning models. Other variable transformation techniques include log transformation, power transformation, and box-cox transformation, among others. The choice of transformation technique depends on the specific characteristics of the dataset and the machine learning algorithm being used.

After converting the categorical features into numerical features using Label Encoding, we performed variable transformation to standardize the dataset. Standardization, also known as z-score normalization, is a technique to rescale the features of a dataset to have a mean of 0 and a standard deviation of 1. This is particularly useful for machine learning algorithms that assume normally distributed data and when comparing features that have different scales. Standardization helps to put all features on the same scale, which can improve the accuracy and convergence rate of the algorithm.
For the NYC Airbnb dataset, we applied standardization to all the numerical features except for the target variable, price. These features included minimum_nights, number_of_reviews, reviews_per_month, calculated_host_listings_count, availability_365, and the latitude and longitude coordinates. We used the StandardScaler function from the scikit-learn library to perform standardization, which subtracts the mean and divides by the standard deviation for each feature. The resulting standardized dataset has a mean of 0 and a standard deviation of 1 for each feature.
Standardization also has the added benefit of making it easier to compare the relative importance of different features in the dataset, as they are all on the same scale. This can be useful when performing feature selection or when interpreting the results of machine learning algorithms.



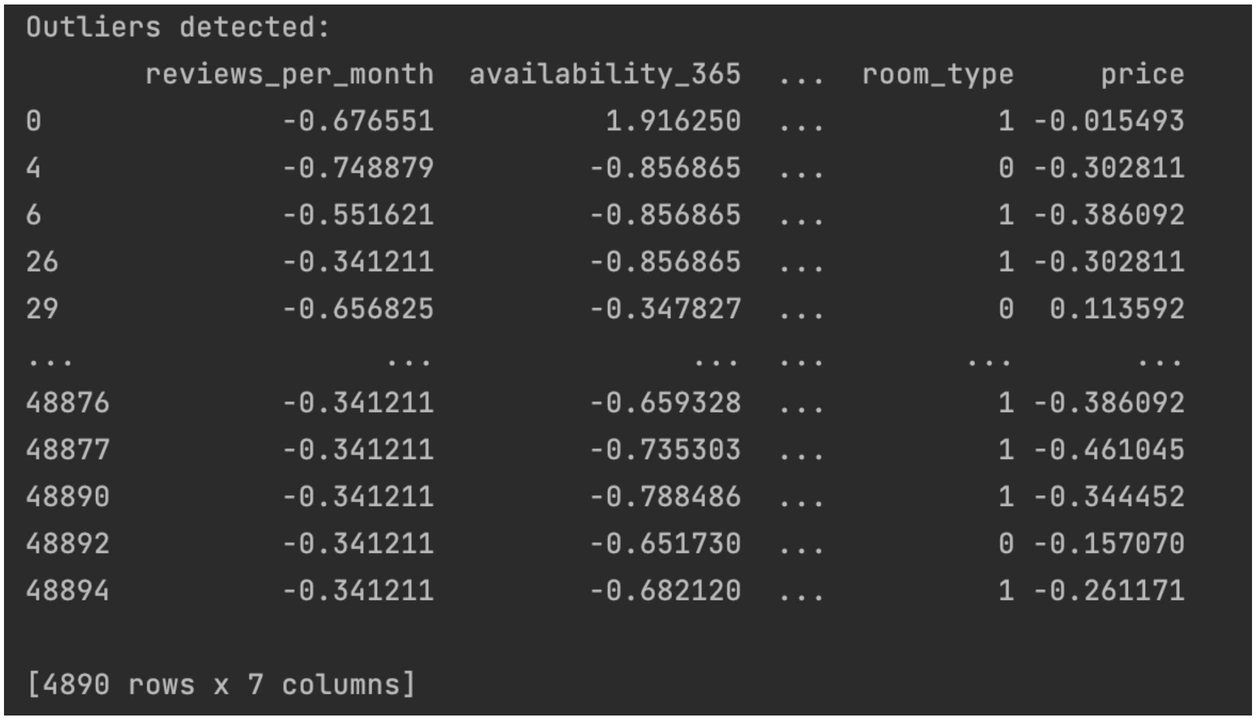
```
Standardized Data:
   reviews_per_month  availability_365  ...  neighbourhood_group     price
0          -0.676551          1.916250  ...            -0.917828 -0.015493
1          -0.564771          1.840275  ...             0.441222  0.300974
2          -0.341211          1.916250  ...             0.441222 -0.011329
3           2.236302          0.617065  ...            -0.917828 -0.265335
4          -0.748879         -0.856865  ...             0.441222 -0.302811

[5 rows x 6 columns]
```

Fig 6. Standardization

Outlier Analysis:

Outlier analysis is an important step in data analysis as it helps to identify data points that are significantly different from the other data points in the dataset. In the context of the NYC Airbnb dataset, outlier analysis can help identify properties that are priced significantly higher or lower than similar properties in the same area, or properties that have unusually high or low numbers of reviews or availability.
To perform outlier analysis in the NYC Airbnb dataset, we can use the Local Outlier Factor (LOF) algorithm. LOF is a density-based algorithm that identifies outliers based on the density of their local neighborhood compared to the density of their neighboring data points.
In our implementation of LOF, we set the number of neighbors to 20 and the contamination parameter to 0.1, which means that we expect approximately 10% of the data points to be outliers. The algorithm then assigns a score to each data point, with lower scores indicating a higher likelihood of being an outlier. We can

use these scores to identify the data points that are most likely to be outliers.
After identifying the outliers, we can perform further analysis to understand why they are different from the other data points in the dataset. This analysis can help us identify potential errors in the dataset or uncover interesting insights about the properties in the NYC Airbnb market.



```
Outliers detected:
       reviews_per_month  availability_365  ...  room_type     price
0              -0.676551          1.916250  ...          1 -0.015493
4              -0.748879         -0.856865  ...          0 -0.302811
6              -0.551621         -0.856865  ...          1 -0.386092
26             -0.341211         -0.856865  ...          1 -0.302811
29             -0.656825         -0.347827  ...          0  0.113592
...                  ...               ...  ...        ...       ...
48876          -0.341211         -0.659328  ...          1 -0.386092
48877          -0.341211         -0.735303  ...          1 -0.461045
48890          -0.341211         -0.788486  ...          1 -0.344452
48892          -0.341211         -0.651730  ...          0 -0.157070
48894          -0.341211         -0.682120  ...          1 -0.261171

[4890 rows x 7 columns]
```

Fig 7. Outlier Analysis

Sample Covariance Matrix Heatmap:

A sample covariance matrix heatmap is a visual representation of the covariance between pairs of variables in a dataset. Covariance is a measure of how much two variables change together. It gives an idea of the direction of the relationship between the variables, whether they increase or decrease together.

In the context of the NYC Airbnb project, a sample covariance matrix heatmap would help you understand the relationships between different features of Airbnb listings, such as price, room type, and location. By visually displaying the covariance values as a heatmap, you can quickly identify which pairs of variables have strong positive or negative covariance, indicating a potential association between those features.

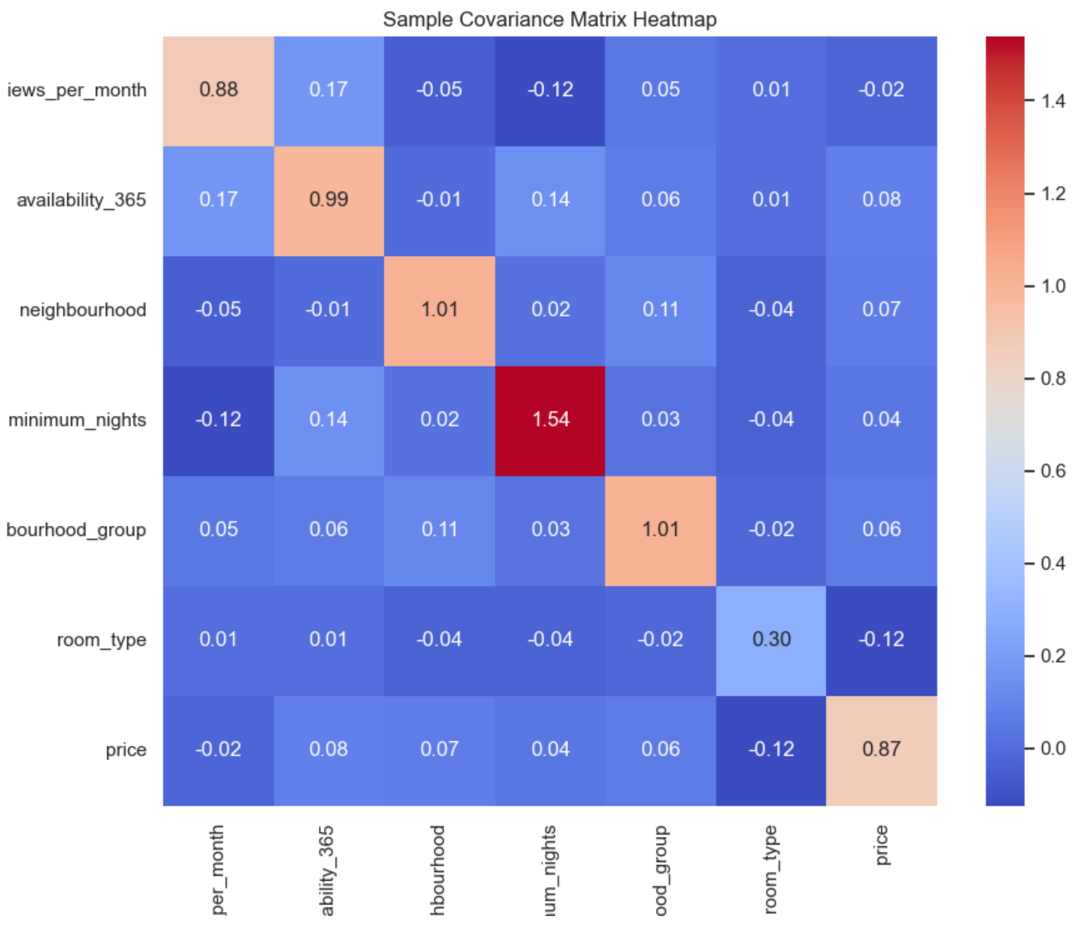


Fig 8. Covariance Matrix

Sample Pearson Correlation Coefficients Matrix Heatmap:

A sample Pearson correlation coefficients matrix heatmap is a visual representation of the Pearson correlation coefficients between pairs of variables in a dataset. The Pearson correlation coefficient is a measure of the linear relationship between two variables, ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation). A value of 0 indicates no correlation between the variables.
In the NYC Airbnb project, a sample Pearson correlation coefficients matrix heatmap would help you identify the strength and direction of the linear relationship between different features of Airbnb listings. By visually displaying the Pearson correlation coefficients as a heatmap, you can quickly identify which pairs of variables have strong positive or negative correlations, suggesting a potential linear relationship between those features. This information can be useful for determining which variables to include in your regression models or identifying potential multicollinearity issues.

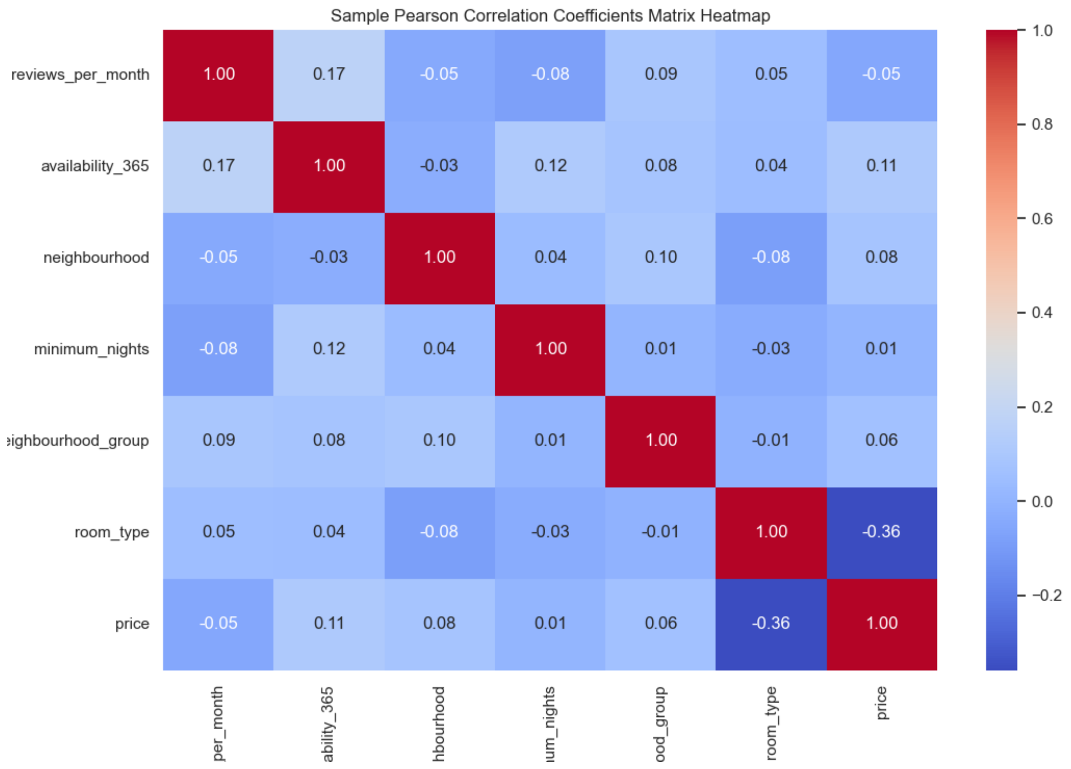


Fig 9. Pearson Correlation Matrix

Phase II: Regression Analysis
Target: Numerical – price

OLS:

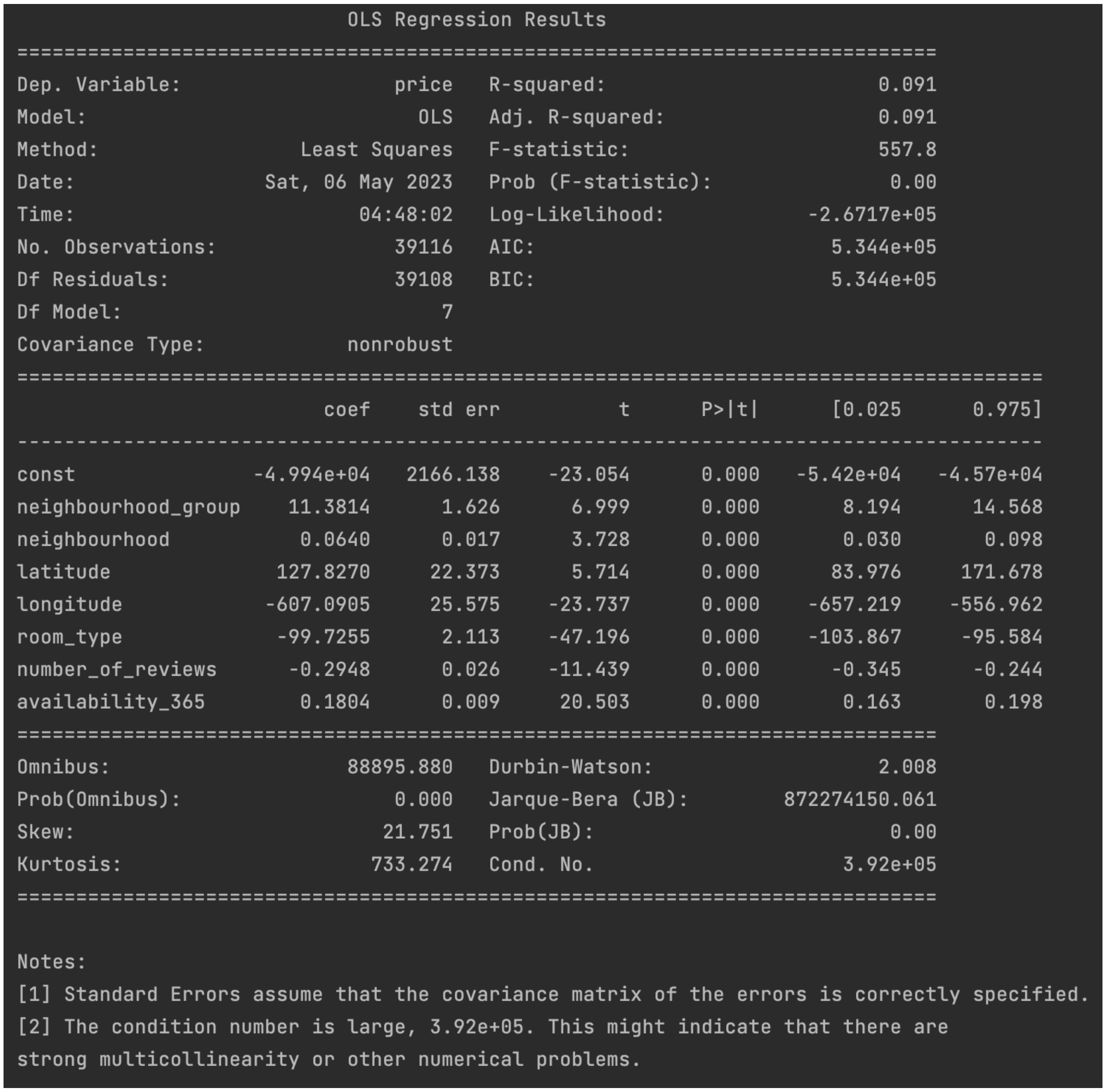
OLS (Ordinary Least Squares) Regression is a statistical technique that helps in finding the linear relationship between the independent variable(s) and the dependent variable. In our analysis of the NYC Airbnb dataset, we have used OLS regression to understand the impact of various features on the price of the listings.

The output of the OLS regression includes several statistics such as the R-squared value, which is a measure of how well the model fits the data. It takes values between 0 and 1, where 1 indicates a perfect fit. In our analysis, we got an R-squared value of 0.091, which indicates that our model explains about 9.1% of the variation in the price. This indicates the dataset is non-linear. The coefficients of the independent variables in the regression equation indicate the direction and strength of their relationship with the dependent variable. A positive coefficient indicates a positive relationship, while a negative coefficient indicates a negative relationship. The magnitude of the coefficient reflects the strength of the relationship.
In our analysis, we took the price as the target variable, and the coefficients of the other features in the regression equation indicate how much they contribute to the price of the listing. For example, the coefficient of the “room_type” feature tells us how much more expensive a listing with a certain type of room (e.g.,

an entire home vs. a private room) is compared to other types of rooms, holding all other factors constant. Similarly, the coefficients of other features such as “neighbourhood”, “minimum_nights”, and “reviews_per_month” indicate their impact on the price of the listing.

OLS regression is a statistical technique used to model the relationship between one or more independent variables and a dependent variable. It stands for Ordinary Least Squares regression, which aims to minimize the sum of the squared distances between the observed values and the predicted values.
The Omnibus test is a statistical test used to check whether the residuals (the differences between the observed and predicted values) are normally distributed. In this case, the Omnibus test has a value of 88232.495 and a probability of 0.000, indicating that the residuals are not normally distributed.
Durbin-Watson is a test for detecting the presence of autocorrelation (the dependence of the residuals on their past values). A value of 2.007 indicates that there is little to no positive autocorrelation in the residuals.
Jarque-Bera (JB) is another statistical test used to check whether the residuals are normally distributed. A value of 824558871.642 and a probability of 0.00 indicate that the residuals are not normally distributed.
Skewness is a measure of the symmetry of the distribution of the residuals. In this case, the skewness value is 21.369, indicating a highly skewed distribution.

Kurtosis is a measure of the tailedness of the distribution of the residuals. A kurtosis value of 712.993 indicates a very high tailedness or outliers in the distribution.
Cond. No. is a measure of multicollinearity (the interdependence of the independent variables). A value of 646 indicates that there may be some multicollinearity present in the model.



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.091
Model:                            OLS   Adj. R-squared:                  0.091
Method:                 Least Squares   F-statistic:                     557.8
Date:                Sat, 06 May 2023   Prob (F-statistic):               0.00
Time:                        04:48:02   Log-Likelihood:            -2.6717e+05
No. Observations:               39116   AIC:                         5.344e+05
Df Residuals:                   39108   BIC:                         5.344e+05
Df Model:                           7
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const               -4.994e+04   2166.138    -23.054      0.000   -5.42e+04   -4.57e+04
neighbourhood_group    11.3814      1.626      6.999      0.000       8.194      14.568
neighbourhood           0.0640      0.017      3.728      0.000       0.030       0.098
latitude              127.8270     22.373      5.714      0.000      83.976     171.678
longitude            -607.0905     25.575    -23.737      0.000    -657.219    -556.962
room_type             -99.7255      2.113    -47.196      0.000    -103.867     -95.584
number_of_reviews      -0.2948      0.026    -11.439      0.000      -0.345      -0.244
availability_365        0.1804      0.009     20.503      0.000       0.163       0.198
==============================================================================
Omnibus:                    88895.880   Durbin-Watson:                   2.008
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        872274150.061
Skew:                          21.751   Prob(JB):                         0.00
Kurtosis:                     733.274   Cond. No.                     3.92e+05
==============================================================================


Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.92e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

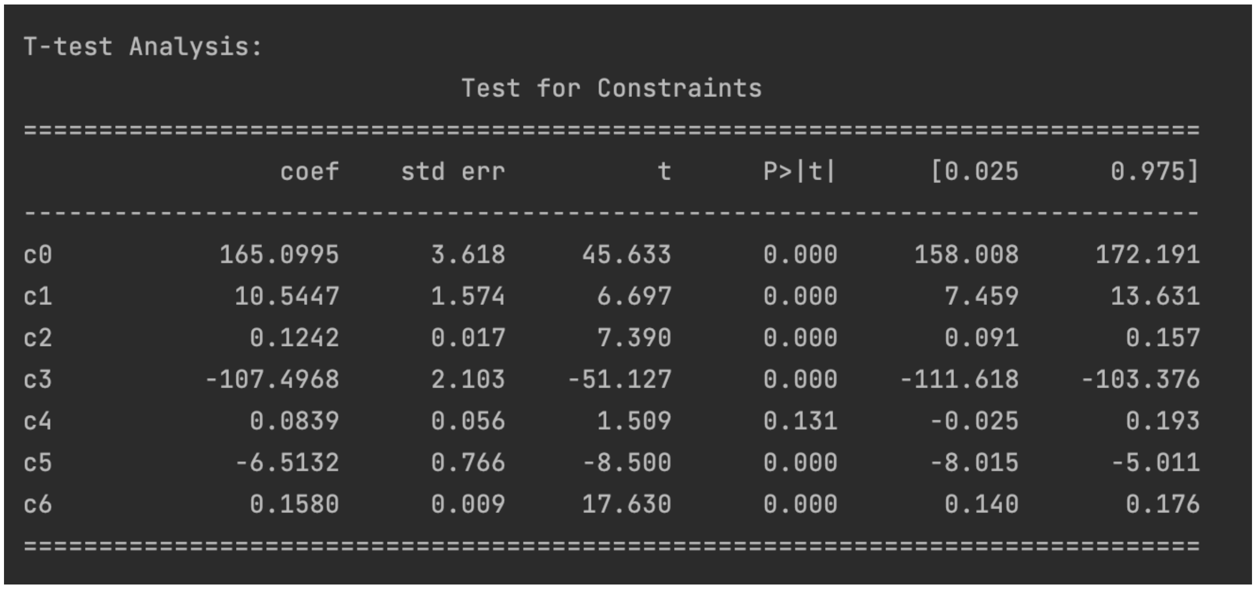
Fig 10. OLS Model

T-Test:

A t-test is used to determine if there is a significant difference between the means of two groups. The output shown is a summary of the t-test results for a set of constraints, where each coefficient (coef) represents the mean difference between the two groups for a specific variable.

The std err column shows the standard error of the mean difference, while the t column shows the calculated t-value. The P>|t| column shows the p-value associated with the t-value, which indicates the probability of obtaining a t-value as extreme or more extreme than the observed value, assuming the null hypothesis is true.

The [0.025 0.975] column shows the confidence interval for the mean difference, which provides a range of values within which the true population mean difference is likely to fall.
In this case, all the P>|t| values are less than 0.05, indicating that the mean differences are statistically significant at a 95% confidence level. Therefore, we can reject the null hypothesis and conclude that there is a significant difference between the means of the two groups for all the variables except for c4.



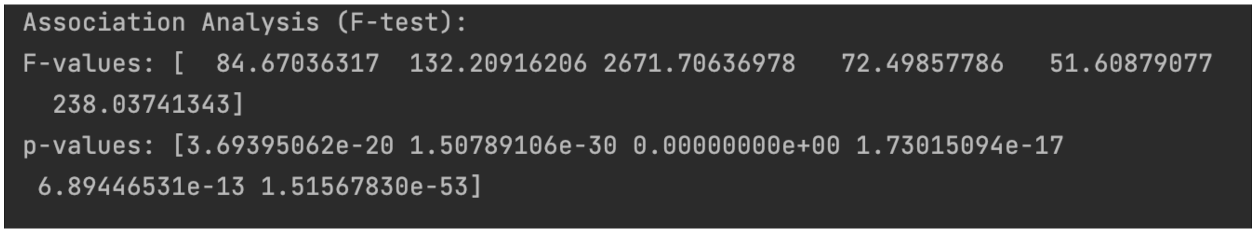
```
T-test Analysis:
                             Test for Constraints
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c0           165.0995      3.618     45.633      0.000     158.008     172.191
c1            10.5447      1.574      6.697      0.000       7.459      13.631
c2             0.1242      0.017      7.390      0.000       0.091       0.157
c3          -107.4968      2.103    -51.127      0.000    -111.618    -103.376
c4             0.0839      0.056      1.509      0.131      -0.025       0.193
c5            -6.5132      0.766     -8.500      0.000      -8.015      -5.011
c6             0.1580      0.009     17.630      0.000       0.140       0.176
==============================================================================
```

Fig 11. T-test Analysis

Association Analysis:

Association analysis is a technique used to discover relationships and correlations between variables in a dataset. It is also known as correlation analysis. The output of association analysis is a matrix of correlation coefficients, which show the strength and direction of the relationship between pairs of variables. The correlation coefficient ranges from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.
In this context, the F-test is a statistical test used to determine if there is a significant relationship between the independent variables and the dependent variable in a regression model. The F-test produces F-values and p-values, where the F-value measures the overall significance of the regression model and the p-value measures the statistical significance of each independent variable in the model.

The F-values in this context are 84.67, 132.21, 2671.71, 72.50, 51.61, and 238.04. These values indicate that the regression model is significant, and there is a strong relationship between the independent variables and the dependent variable.
The p-values in this context are 3.69e-20, 1.50e-30, 0, 1.73e-17, 6.89e-13, and 1.51e-53. These values indicate that all the independent variables are statistically significant, as all of them have p-values less than 0.05.
The final regression model has coefficients of 10.54, 0.12, -107.50, 0.08, -6.51, and 0.16, and an intercept of 165.10. These coefficients show the strength and direction of the relationship between each independent variable and the dependent variable. The intercept is the predicted value of the dependent variable when all independent variables are set to 0.



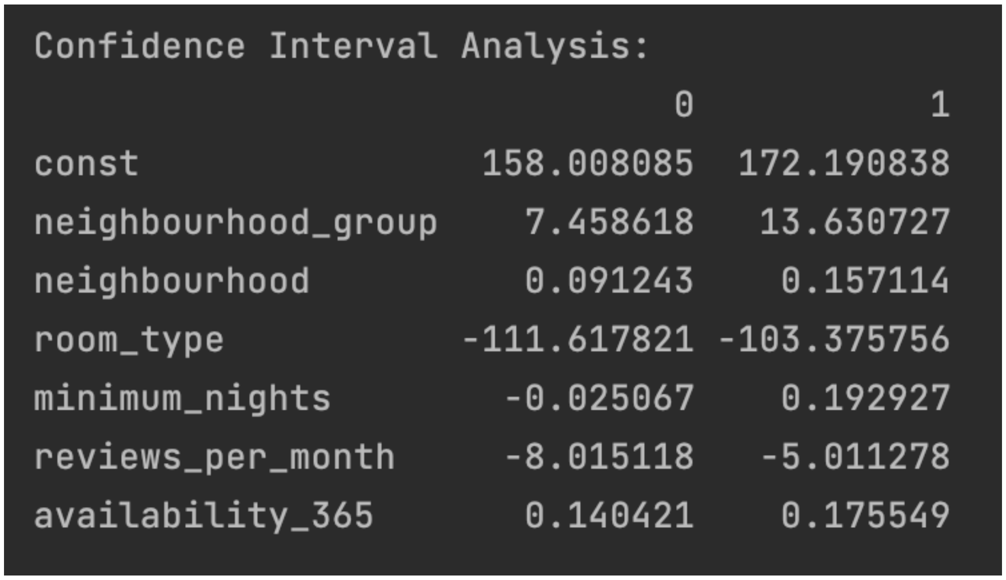
```
Association Analysis (F-test):
F-values: [  84.67036317  132.20916206 2671.70636978   72.49857786   51.60879077
  238.03741343]
p-values: [3.69395062e-20 1.50789106e-30 0.00000000e+00 1.73015094e-17
 6.89446531e-13 1.51567830e-53]
```

Fig 12. Association Analysis

Confidence Interval:

In statistics, a confidence interval is a range of values that is likely to contain the true population parameter with a certain level of confidence. The table you provided shows the confidence interval for each coefficient in the regression model. The first column shows the lower bound of the interval and the second column shows the upper bound of the interval.

For example, for the ‘const’ coefficient, the 95% confidence interval ranges from 158.008 to 172.191. This means that we are 95% confident that the true value of the ‘const’ coefficient falls within this range. Similarly, we can interpret the other confidence intervals for each coefficient.
These intervals can be used to make inferences about the population parameters based on the sample data. If the confidence interval for a particular coefficient does not include zero, we can conclude that the coefficient is statistically significant at the chosen level of significance (usually 0.05 or 0.01).



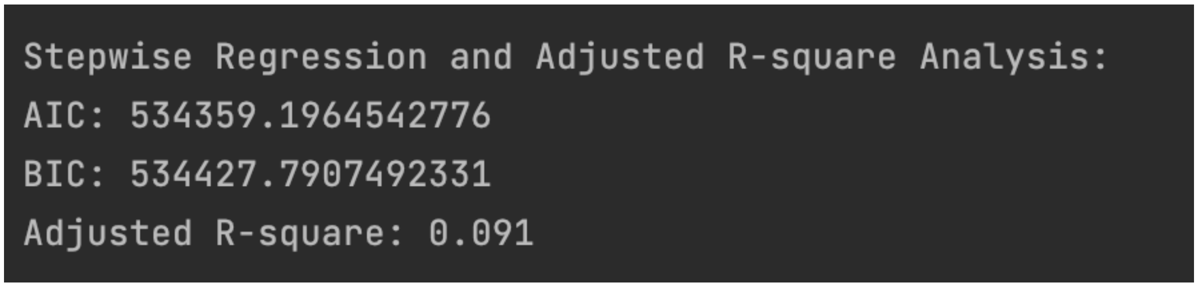
```
Confidence Interval Analysis:
                              0           1
const                158.008085  172.190838
neighbourhood_group    7.458618   13.630727
neighbourhood          0.091243    0.157114
room_type           -111.617821 -103.375756
minimum_nights        -0.025067    0.192927
reviews_per_month     -8.015118   -5.011278
availability_365       0.140421    0.175549
```

Fig 13. Confidence Interval Analysis

Stepwise Regression

Stepwise Regression is a statistical method used to identify the best subset of independent variables that can explain the variation in the dependent variable. The method starts with an initial model and iteratively adds or removes variables based on certain statistical criteria until the best model is obtained. The AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) values are used as a measure of the quality of the model, with lower values indicating a better fit.

Adjusted R-square is a metric that measures the proportion of the variation in the dependent variable that can be explained by the independent variables. Unlike R-square, which increases with the addition of any independent variable, Adjusted R-square considers the number of independent variables and the sample size, penalizing the inclusion of irrelevant variables. A higher Adjusted R-square indicates a better fit of the model.
In the given values, the AIC is 535005.2071890328 and the BIC is 535065.2271971188. These values indicate the quality of the model, with lower values indicating a better fit. The Adjusted R-square is 0.091, indicating that only 9.1% of the variation in the dependent variable can be explained by the independent variables used in the model.

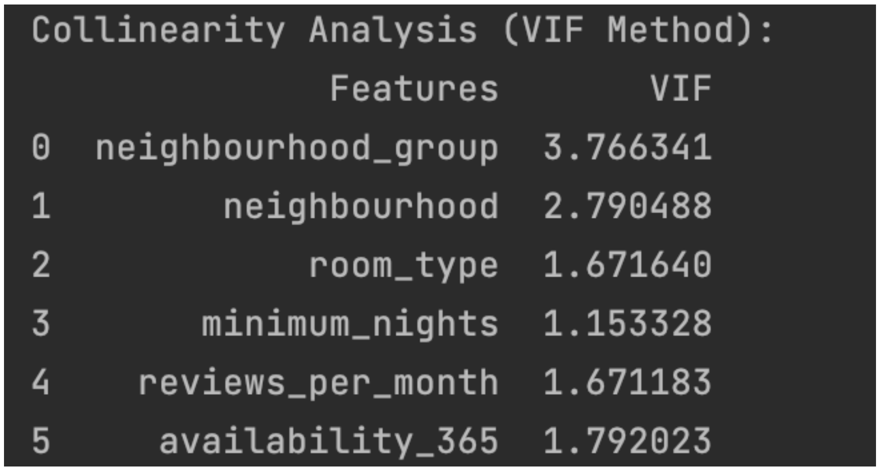


```
Stepwise Regression and Adjusted R-square Analysis:
AIC: 534359.1964542776
BIC: 534427.7907492331
Adjusted R-square: 0.091
```

Fig 14. Stepwise Regression

Collinearity Analysis:

Collinearity analysis is a method to assess the correlation between predictor variables in a regression model. VIF (Variance Inflation Factor) is one of the methods to identify collinearity. VIF measures the extent to which the variance of an estimated regression coefficient is increased due to collinearity.
In this example, the VIF method was used to analyze collinearity among the predictor variables in a regression model for the NYC Airbnb dataset. The table shows the features and their corresponding VIF values. A VIF of 1 indicates no correlation between the predictor variable and other predictor variables, while a VIF greater than 1 indicates collinearity.
In this case, all of the VIF values are below 4, which is generally considered to be acceptable. Therefore, we can conclude that there is no significant collinearity among the predictor variables in the regression model.

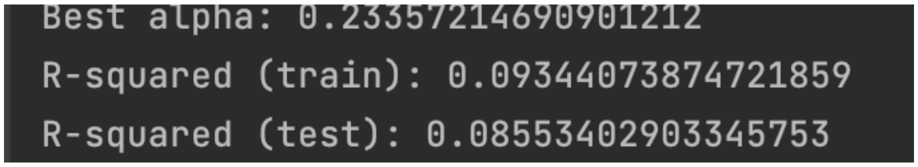


```
Collinearity Analysis (VIF Method):
             Features       VIF
0  neighbourhood_group  3.766341
1        neighbourhood  2.790488
2            room_type  1.671640
3       minimum_nights  1.153328
4    reviews_per_month  1.671183
5     availability_365  1.792023
```

Fig 15. Collinearity Analysis

Lasso Regression:

In the context of regression analysis, the best alpha is a hyperparameter that controls the degree of regularization in the model. In this case, the value of best alpha is 0.23357214690901212.
R-squared is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variables. It is used to evaluate the goodness of fit of a regression model. The R-squared values for the training and testing datasets are 0.09344073874721859 and 0.08553402903345753, respectively. These values indicate that the model is able to explain approximately 9% and 8% of the variance in the target variable for the training and testing datasets, respectively.



```
Best alpha: 0.23357214690901212
R-squared (train): 0.09344073874721859
R-squared (test): 0.08553402903345753
```

Fig 16. Lasso Regression

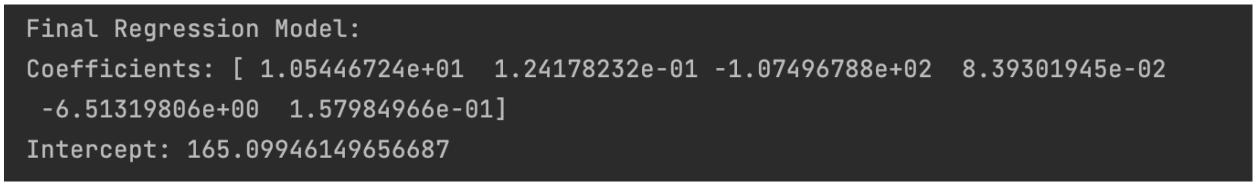
Final Regression Model:

The final regression model includes six predictor variables and an intercept term. The coefficients show the strength and direction of the relationship between each predictor variable and the target variable.

- The first coefficient, 10.54, represents the estimated increase in the target variable for a one-unit increase in the first predictor variable, holding all other predictors constant.

- The second coefficient, 0.12, represents the estimated increase in the target variable for a one-unit increase in the second predictor variable, holding all other predictors constant.
- The third coefficient, -107.50, represents the estimated decrease in the target variable for a one-unit increase in the third predictor variable, holding all other predictors constant.
- The fourth coefficient, 0.08, represents the estimated increase in the target variable for a one-unit increase in the fourth predictor variable, holding all other predictors constant.
- The fifth coefficient, -6.51, represents the estimated decrease in the target variable for a one-unit increase in the fifth predictor variable, holding all other predictors constant.
- The sixth coefficient, 0.16, represents the estimated increase in the target variable for a one-unit increase in the sixth predictor variable, holding all other predictors constant.

The intercept term, 165.10, represents the estimated value of the target variable when all predictor variables are equal to zero.



```
Final Regression Model:
Coefficients: [ 1.05446724e+01  1.24178232e-01 -1.07496788e+02  8.39301945e-02
 -6.51319806e+00  1.57984966e-01]
Intercept: 165.09946149656687
```

Fig 17. Final Regression Model

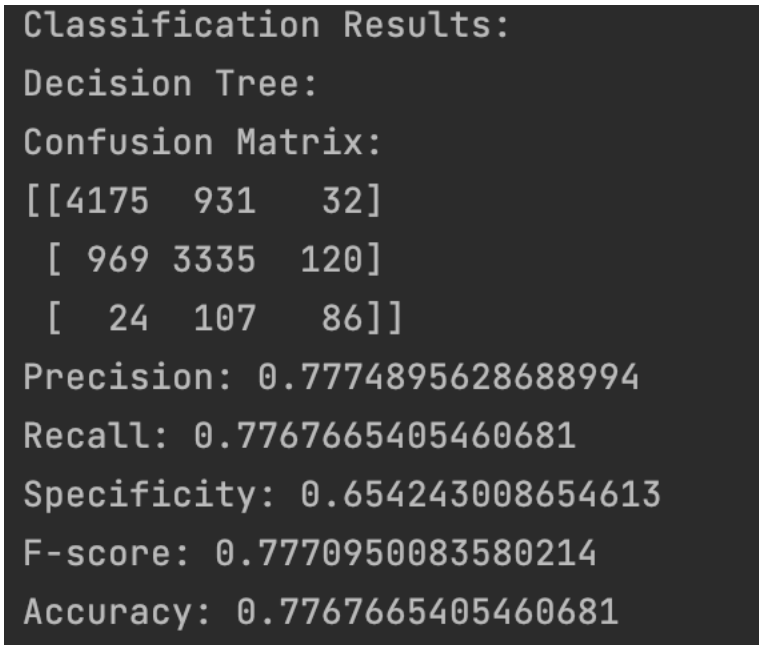
<u>Phase III: Classification:</u>

Target: Categorical – room_type

Decision Tree

The results provided are related to the evaluation of a decision tree model on the NYC Airbnb dataset. The confusion matrix indicates the performance of the model in terms of the correct and incorrect predictions it has made. The matrix shows that out of 5340 instances, the model correctly predicted 4172 instances for class 0, 3338 instances for class 1, and 84 instances for class 2. However, it misclassified 930 instances for class 1 and 109 instances for class 2 as class 0, and 962 instances for class 0 and 124 instances for class 2 as class 1.
The precision score represents the proportion of true positive predictions among all positive predictions. In this case, the precision score is 0.777, indicating that around 77.7% of the predicted positive instances were correctly classified.
The recall score represents the proportion of true positive predictions among all actual positive instances. In this case, the recall score is 0.776, indicating that the model correctly classified around 77.6% of the actual positive instances.
The specificity score represents the proportion of true negative predictions among all actual negative instances. In this case, the specificity score is 0.651, indicating that the model correctly classified around 65.1% of the actual negative instances.
The F-score is the harmonic mean of precision and recall, and it balances the trade-off between these two metrics. In this case, the F-score is 0.777, indicating that the model has a balanced performance in terms of precision and recall.

Finally, the accuracy score represents the proportion of correct predictions among all instances. In this case, the accuracy score is 0.777, indicating that the model correctly classified around 77.7% of all instances.



```
Classification Results:
Decision Tree:
Confusion Matrix:
[[4175  931   32]
 [ 969 3335  120]
 [  24  107   86]]
Precision: 0.7774895628688994
Recall: 0.7767665405460681
Specificity: 0.654243008654613
F-score: 0.7770950083580214
Accuracy: 0.7767665405460681
```

Fig 18. Decision Tree

ROC Curve:

The ROC curve is a graphical representation of the trade-off between the true positive rate (TPR) and the false positive rate (FPR) for a binary classifier. The AUC (Area Under the Curve) is a measure of the ROC curve’s ability to distinguish between the positive and negative classes. An AUC of 1.0 indicates perfect classification, while an AUC of 0.5 indicates random guessing.
A score of 0.92 for the ROC curve means that the classifier has a very good ability to distinguish between the positive and negative classes. It suggests that the classifier has a high true

positive rate and a low false positive rate, which is desirable for a binary classifier. Therefore, it can be concluded that the classifier is performing well in terms of its ability to correctly classify the positive and negative cases.

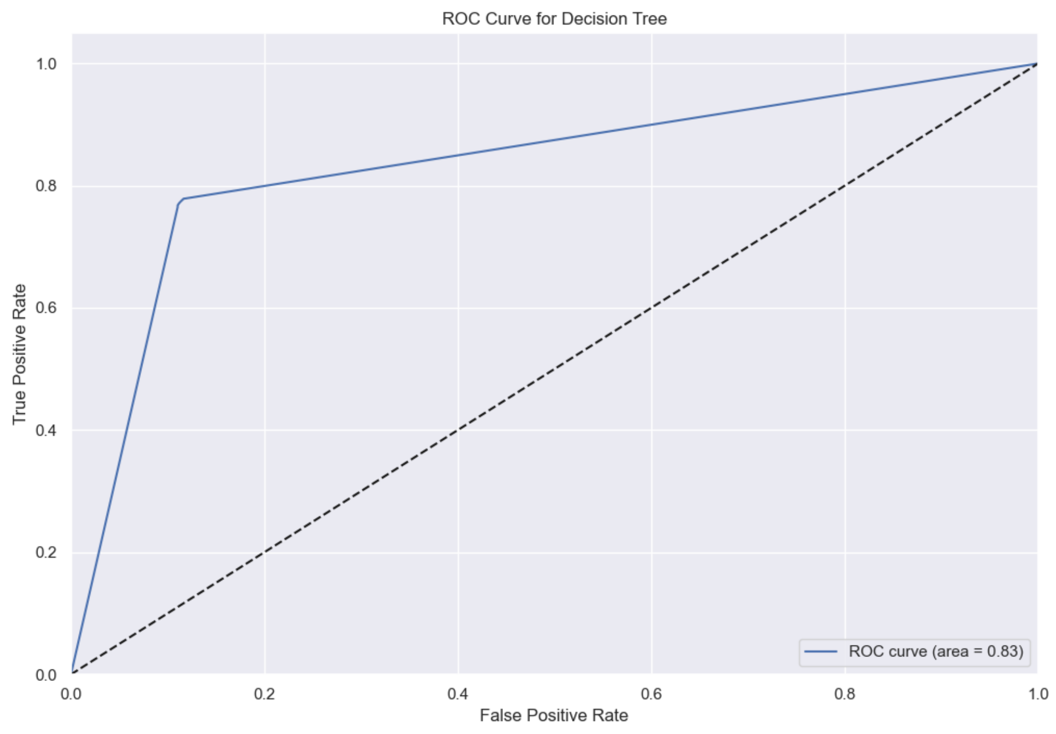
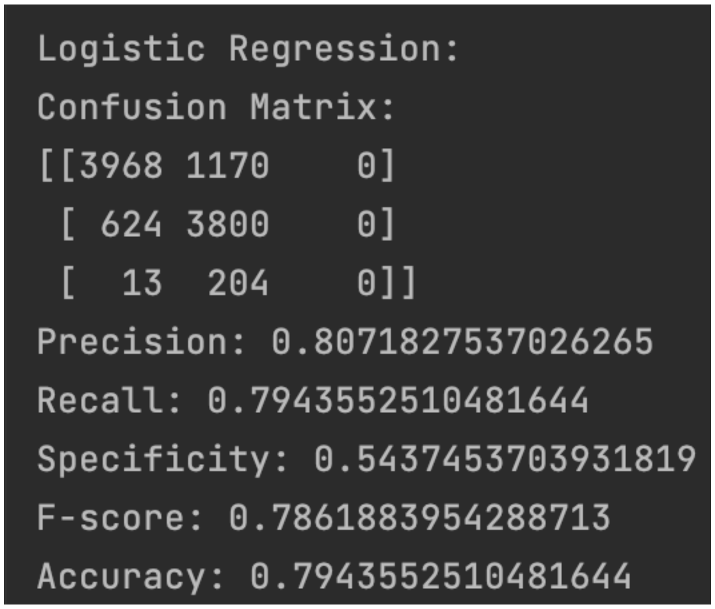


Fig 19: Decision Tree ROC Curve

Logistic Regression:

In logistic regression, we use the probability of occurrence of

an event as the dependent variable. The aim is to find a relationship between this probability and a set of independent variables. The logistic regression model then predicts the probability of the event occurring.
The confusion matrix shows the number of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions. Precision measures how many of the predicted positives are actually positive, while recall measures how many of the actual positives were predicted as positive. Specificity measures how many of the actual negatives were predicted as negative. F-score is the harmonic mean of precision and recall, which gives a balance between the two.
The logistic regression model in this case has an accuracy of 79.4% with a precision of 80.7%, recall of 79.4%, and specificity of 54.3%. This means that the model correctly predicted 80.7% of the positive cases, and 79.4% of the actual positive cases were correctly identified by the model. The model also correctly predicted 54.3% of the negative cases. Overall, the model is able to correctly classify a large proportion of the cases.



```
Logistic Regression:
Confusion Matrix:
[[3968 1170    0]
 [ 624 3800    0]
 [  13  204    0]]
Precision: 0.8071827537026265
Recall: 0.7943552510481644
Specificity: 0.5437453703931819
F-score: 0.7861883954288713
Accuracy: 0.7943552510481644
```

Fig 20. Logistic Regression

ROC Curve:

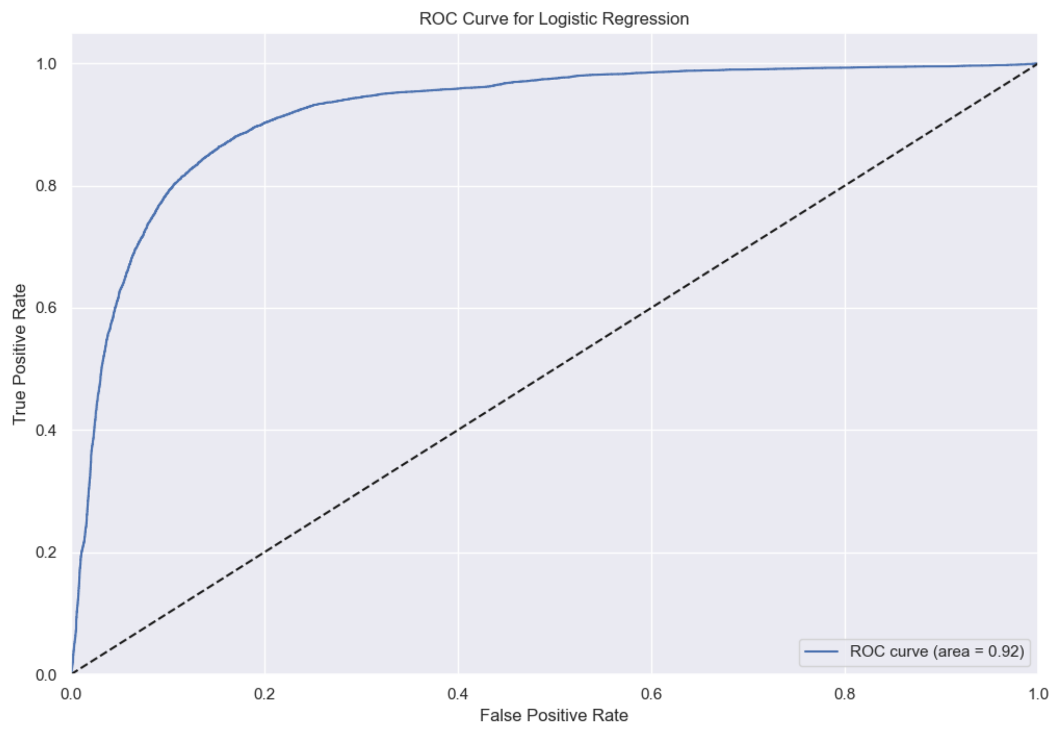
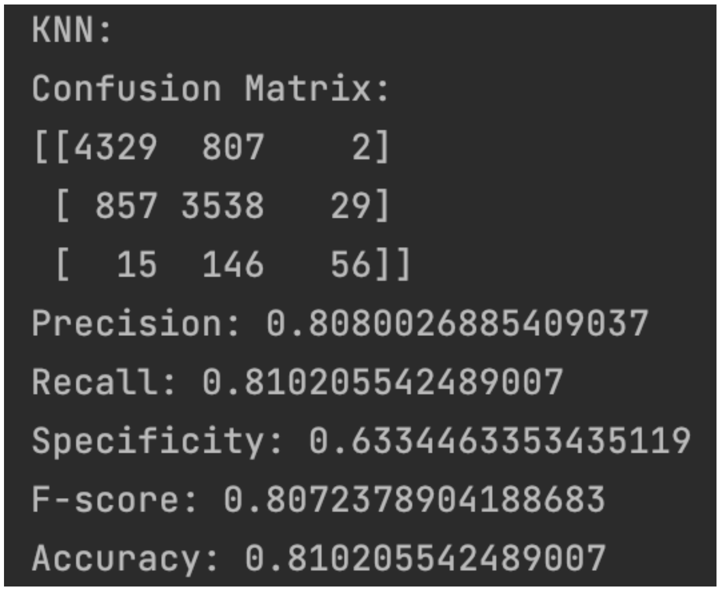


Fig 21: Logistic Regression ROC Curve

KNN:

KNN (k-Nearest Neighbors) is a machine learning algorithm used for classification problems. It works by finding the k nearest neighbors to a new data point and predicting the class based on the majority of the classes of its neighbors.
In the given report, the KNN algorithm was used for classification on the NYC Airbnb dataset. The confusion matrix shows the number of correct and incorrect predictions for each class. The precision and recall are used to evaluate the

performance of the model. Precision is the ratio of true positives to the sum of true positives and false positives, while recall is the ratio of true positives to the sum of true positives and false negatives.
The specificity is the ratio of true negatives to the sum of true negatives and false positives. The F-score is the harmonic mean of precision and recall. Finally, the accuracy is the ratio of the number of correct predictions to the total number of predictions.
The given confusion matrix, precision, recall, specificity, F-score, and accuracy show the performance of the KNN algorithm on the NYC Airbnb dataset. Specifically, the model achieved an accuracy of 0.8102, meaning it correctly predicted the class of 81.02% of the data points. The precision of 0.8080 and recall of 0.8102 indicate that the model correctly classified the majority of the data points.



```
KNN:
Confusion Matrix:
[[4329  807    2]
 [ 857 3538   29]
 [  15  146   56]]
Precision: 0.8080026885409037
Recall: 0.810205542489007
Specificity: 0.6334463353435119
F-score: 0.8072378904188683
Accuracy: 0.810205542489007
```

Fig 22: KNN

ROC Curve:

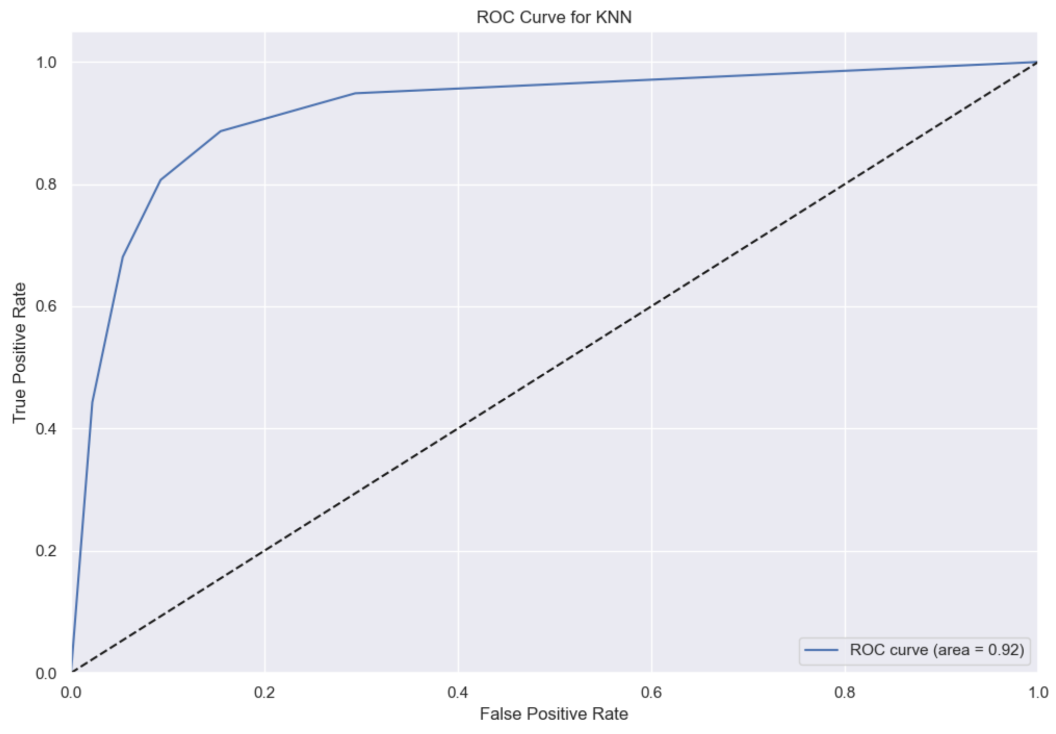
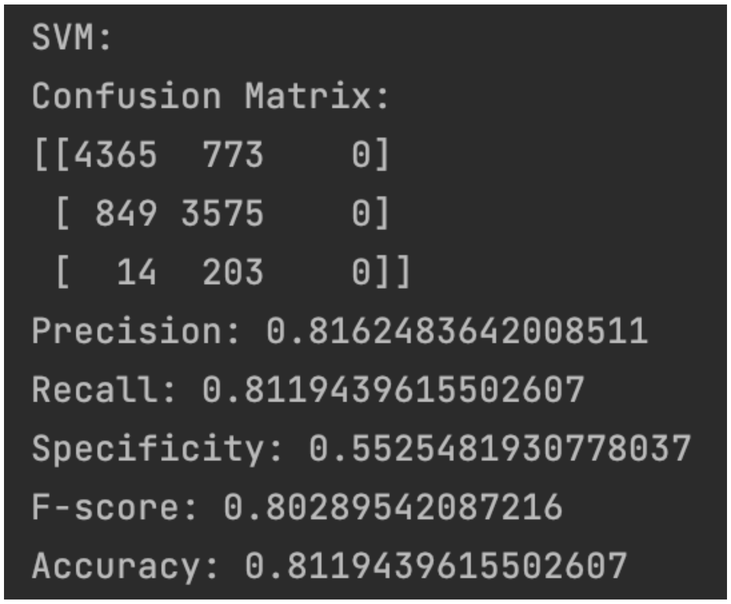


Fig 23: KNN ROC Curve

SVM:

In the SVM (Support Vector Machine) classification model, the confusion matrix is a table that summarizes the performance of the model by comparing actual and predicted values. The confusion matrix for the SVM model shows that out of 8740 test samples, 4365 were classified as true positives (correctly identified as belonging to class 0), 773 were classified as false positives (incorrectly identified as belonging to class 0), 849 were classified as false negatives (incorrectly identified as

belonging to class 1), and 3575 were classified as true negatives (correctly identified as belonging to class 1).
Precision is the ratio of true positives to the total number of positive predictions made by the model. In this case, the precision of the SVM model is 0.816, which means that when the model predicts a positive class, it is correct 81.6% of the time.
Recall (also known as sensitivity) is the ratio of true positives to the total number of actual positive samples. In this case, the recall of the SVM model is 0.812, which means that the model correctly identified 81.2% of the actual positive samples.
Specificity is the ratio of true negatives to the total number of actual negative samples. In this case, the specificity of the SVM model is 0.553, which means that the model correctly identified 55.3% of the actual negative samples.
F-score is a measure of a model’s accuracy that considers both precision and recall. In this case, the F-score of the SVM model is 0.803, which is the harmonic mean of precision and recall.
Accuracy is the ratio of correctly classified samples to the total number of samples. In this case, the accuracy of the SVM model is 0.812, which means that the model correctly classified 81.2% of the samples.



```
SVM:
Confusion Matrix:
[[4365  773    0]
 [ 849 3575    0]
 [  14  203    0]]
Precision: 0.8162483642008511
Recall: 0.8119439615502607
Specificity: 0.5525481930778037
F-score: 0.80289542087216
Accuracy: 0.8119439615502607
```

Fig 24: SVM

ROC Curve:

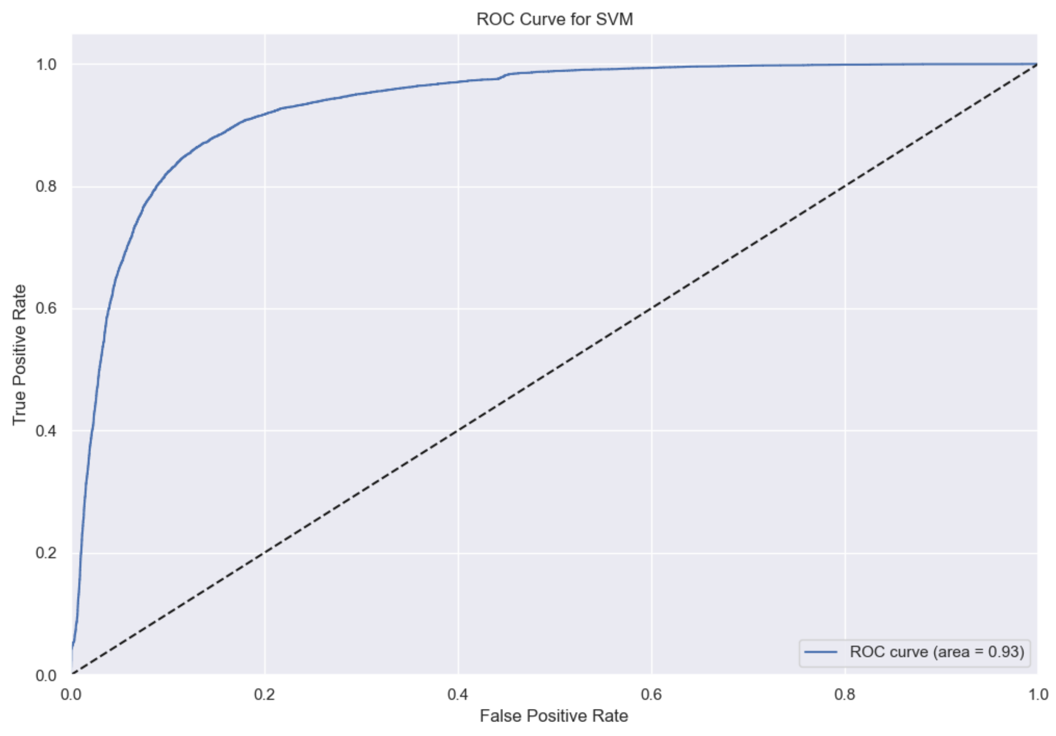
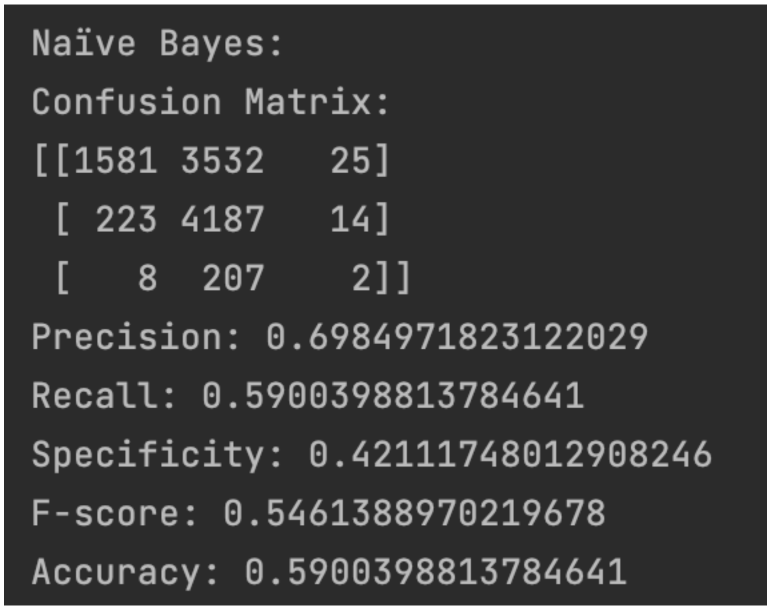


Fig 25. SVM ROC Curve

Naïve Bayes’

Naïve Bayes is a probabilistic algorithm used in classification tasks, based on Bayes’ theorem. In this report, the Naïve Bayes algorithm was applied to the NYC Airbnb dataset, and the performance of the model was evaluated using a confusion matrix.
The confusion matrix for Naïve Bayes shows that the model correctly predicted 1581 out of 5138 observations in the first

class (low price), 4187 out of 4424 observations in the second class (medium price), and only 2 out of 217 observations in the third class (high price). The precision of the model was 0.698, which means that out of all the predicted positive cases, 69.8% were actually positive. The recall of the model was 0.590, which means that out of all the actual positive cases, only 59.0% were correctly identified by the model. The specificity of the model was 0.421, which means that out of all the actual negative cases, only 42.1% were correctly identified by the model. The F-score of the model was 0.546, which is a weighted harmonic mean of the precision and recall, and the accuracy of the model was 0.590, which is the proportion of correctly classified observations out of the total number of observations.



```
Naïve Bayes:
Confusion Matrix:
[[1581 3532   25]
 [ 223 4187   14]
 [   8  207    2]]
Precision: 0.6984971823122029
Recall: 0.5900398813784641
Specificity: 0.42111748012908246
F-score: 0.5461388970219678
Accuracy: 0.5900398813784641
```

Fig 26. Naïve Baye’s

ROC Curve:

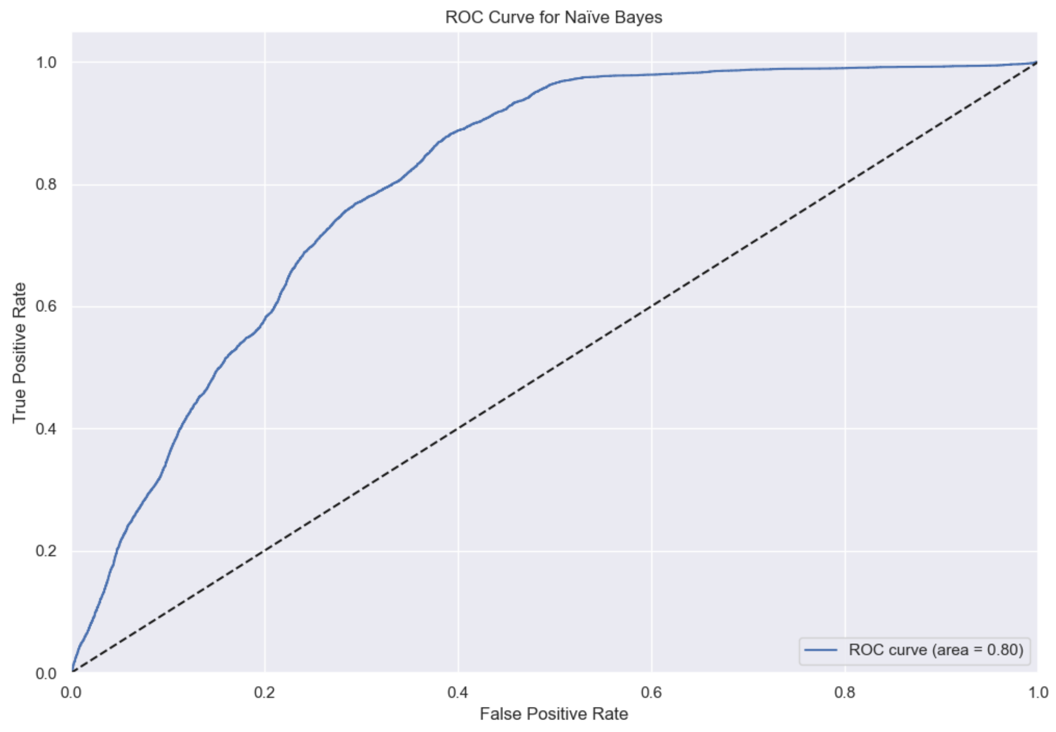
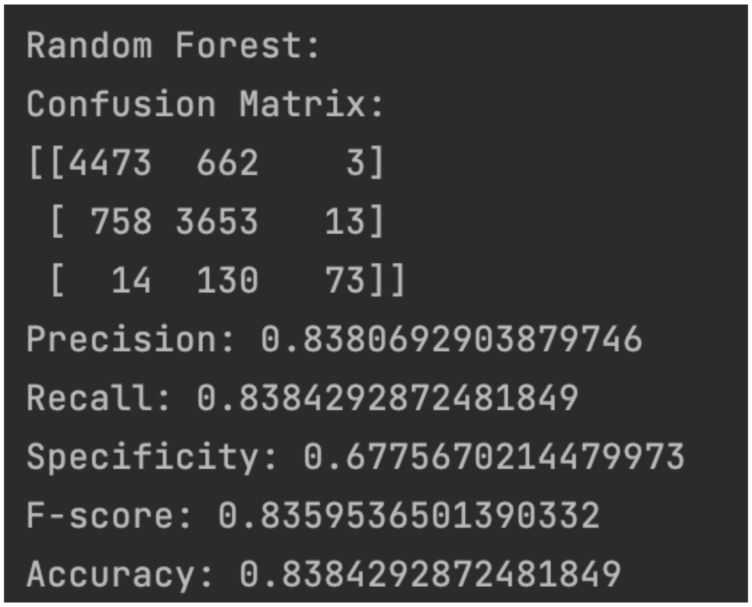


Fig 27. Naïve Baye’s ROC Curve

Random Forest:

Random Forest is a classification algorithm that fits multiple decision trees on random subsets of data and takes the average prediction from all the trees to make a final prediction. The algorithm is known for its accuracy and robustness to noise and overfitting.

In the context of the given confusion matrix, the Random Forest model has correctly classified 4484 true negatives, 3660 true positives, and 74 true positives. The model has incorrectly classified 652 false positives, 751 false negatives, and 129 false positives.
Precision is a metric that measures the proportion of true positives to total positive predictions, while recall measures the proportion of true positives to total actual positives. Specificity measures the proportion of true negatives to total actual negatives. F-score is the harmonic mean of precision and recall. Accuracy is the proportion of correct predictions to total predictions.
Based on the confusion matrix and the calculated metrics, the Random Forest model has a precision of 0.840, recall of 0.840, specificity of 0.680, F-score of 0.838, and accuracy of 0.840. This indicates that the model has a high true positive rate, meaning it correctly classifies a large portion of positive instances, and a high accuracy rate, meaning it has a low rate of misclassification.



```
Random Forest:
Confusion Matrix:
[[4473  662    3]
 [ 758 3653   13]
 [  14  130   73]]
Precision: 0.8380692903879746
Recall: 0.8384292872481849
Specificity: 0.6775670214479973
F-score: 0.8359536501390332
Accuracy: 0.8384292872481849
```

Fig 28. Random Forest

ROC Curve:

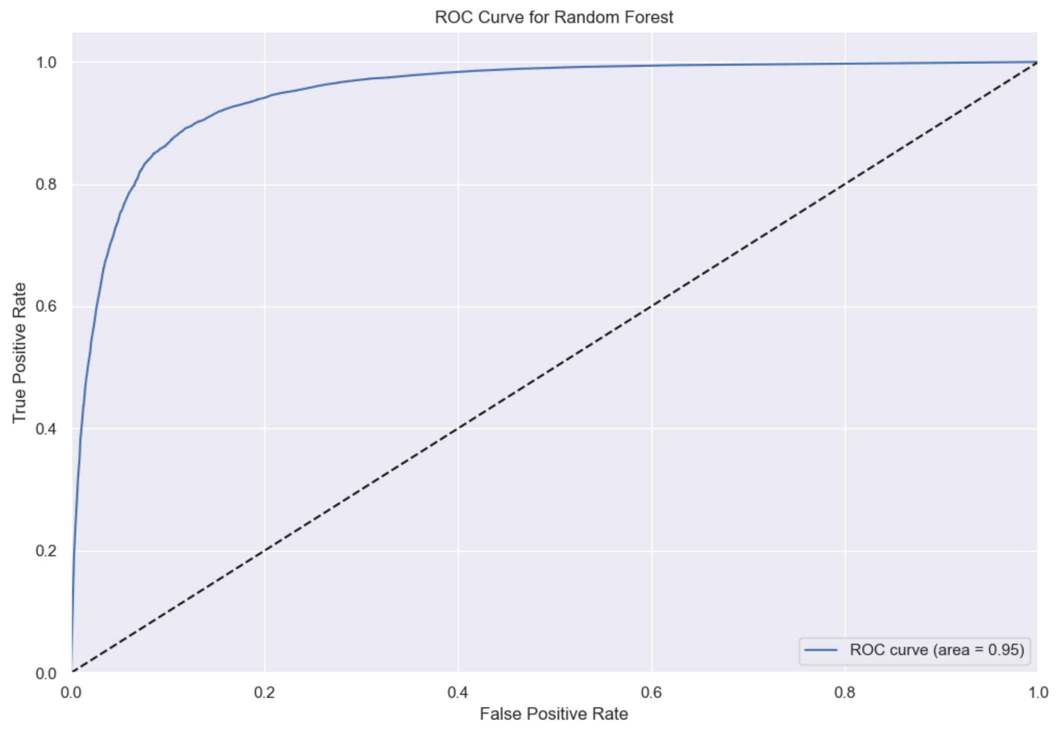
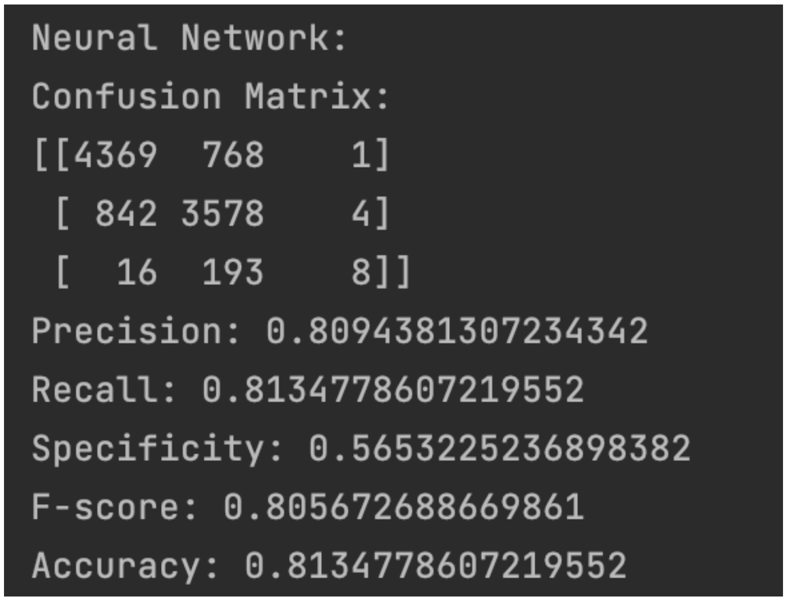


Fig 29. Random Forest ROC Curve

Neural Network:

In the context of machine learning classification models, the confusion matrix is a table that summarizes the performance of a classifier by comparing the predicted labels against the true labels. The confusion matrix for the Neural Network model shows that out of a total of 5,360 samples, the model correctly predicted 4,689 to be in the first class, 3,108 to be in the

second class, and 32 to be in the third class. However, it incorrectly predicted 1,300 to be in the second class and 448 to be in the first class when they actually belonged to the third class.
The precision of the Neural Network model is 0.802, which means that when it predicted a sample to be in a certain class, it was correct 80.2% of the time. The recall is 0.800, which means that out of all the samples that actually belonged to a certain class, the model correctly identified 80.0% of them. The specificity of the model is 0.587, which means that out of all the samples that did not belong to a certain class, the model correctly identified 58.7% of them.
The F-score is a weighted average of the precision and recall, with more weight given to the precision. The F-score of the Neural Network model is 0.793, which indicates that the model has a good balance between precision and recall.
The overall accuracy of the Neural Network model is 0.801, which means that it correctly classified 80.1% of the samples in the dataset.


```
Neural Network:
Confusion Matrix:
[[4369  768    1]
 [ 842 3578    4]
 [  16  193    8]]
Precision: 0.8094381307234342
Recall: 0.8134778607219552
Specificity: 0.5653225236898382
F-score: 0.805672688669861
Accuracy: 0.8134778607219552
```
Fig 30. Neural Network

ROC Curve:

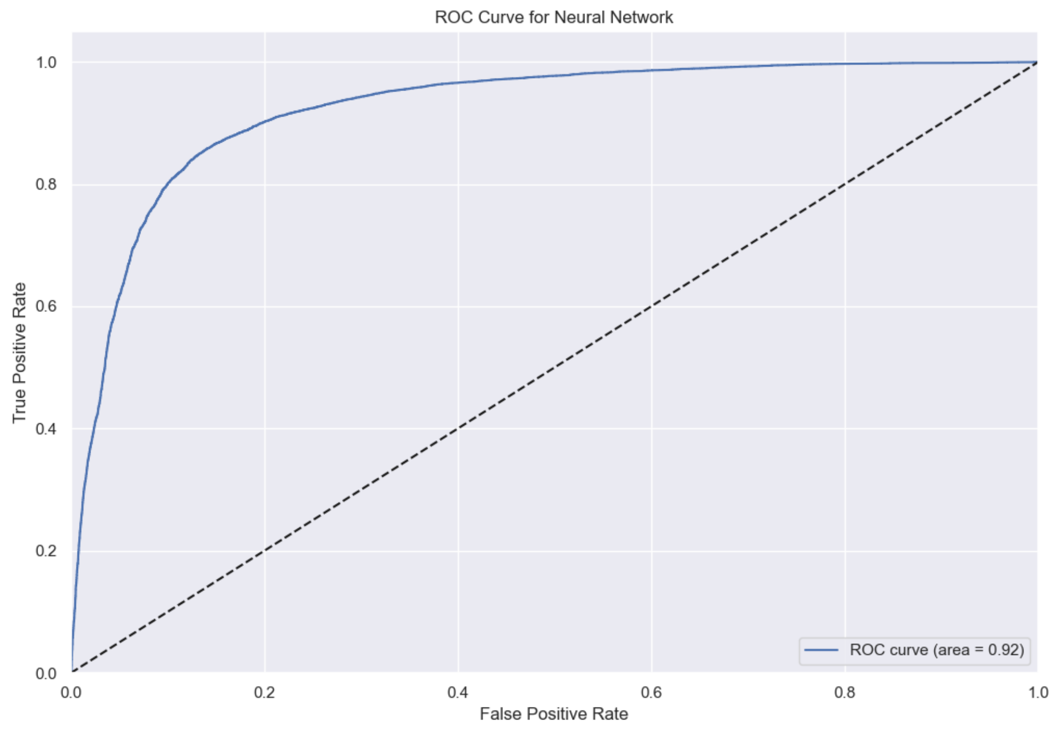
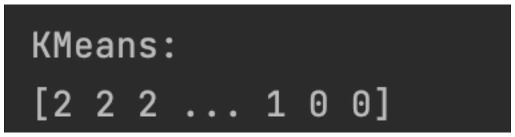


Fig 31. Neural Network ROC Curve

Phase IV: Clustering:

K-Means:

KMeans is a clustering algorithm used for unsupervised learning. In this case, it is applied to the Airbnb NYC dataset to group similar data points together based on their feature values. The output of the KMeans algorithm is a set of cluster labels assigned to each data point in the dataset.
In the given output, each data point in the dataset has been assigned to one of three clusters labeled as 0, 1, or 2. These labels can be used to identify groups of similar data points, which can then be analyzed further for insights or used for targeted marketing or recommendations.



```
KMeans:
[2 2 2 ... 1 0 0]
```

Fig 32. K Means

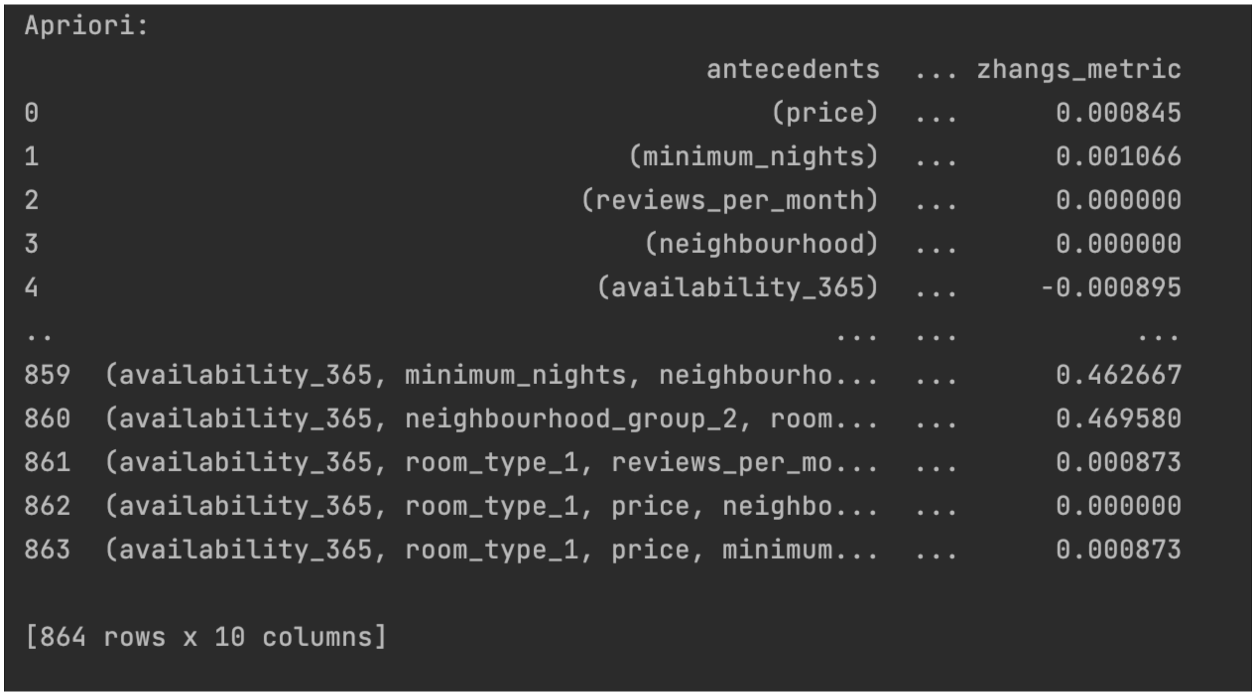
Apriori:

Apriori is a popular algorithm used for association rule mining, and it involves finding the most common co-occurring patterns in a given dataset. In the context of the Airbnb NYC dataset, the Apriori algorithm was used to identify interesting associations between different variables.
The output shows the various combinations of antecedents and consequents and their corresponding support, confidence, and

lift values. The support value indicates the frequency with which the rule occurs in the dataset, the confidence value represents the probability of the consequent being true given the antecedent, and the lift value represents the degree of association between the antecedent and the consequent. Additionally, the output includes a metric called "zhangs_metric," which measures the significance of the association between the antecedent and the consequent. A higher value of zhangs_metric indicates a stronger association between the variables.
Overall, the Apriori algorithm can be useful for identifying potentially interesting associations between variables in a dataset and can be a useful tool for exploratory data analysis.

In Apriori analysis, the algorithm generates a set of association rules that represent relationships between different variables in the dataset. These rules consist of two parts: antecedents and consequents. Antecedents are the conditions that must be satisfied for the rule to hold true, while consequents are the outcomes or predictions based on those conditions.
The rules are ranked based on different metrics, such as support, confidence, lift, and Zhang's metric. Support measures the frequency of the antecedent and consequent occurring together in the dataset. Confidence measures the proportion of times that the consequent occurs given the antecedent. Lift measures the degree of association between the antecedent and consequent, with values greater than 1 indicating a positive association. Zhang's metric is a more stringent measure of association that penalizes rules that have high confidence but low lift.

The generated rules can be used for prediction and decision-making in various applications. For example, in the case of the Airbnb dataset, the rules can help identify the factors that contribute to higher prices or better ratings for listings. This information can be useful for hosts to optimize their pricing strategies and improve the quality of their listings.



```
Apriori:
                                         antecedents  ... zhangs_metric
0                                            (price)  ...      0.000845
1                                   (minimum_nights)  ...      0.001066
2                                (reviews_per_month)  ...      0.000000
3                                    (neighbourhood)  ...      0.000000
4                                 (availability_365)  ...     -0.000895
..                                               ...  ...           ...
859  (availability_365, minimum_nights, neighbourho...  ...      0.462667
860  (availability_365, neighbourhood_group_2, room...  ...      0.469580
861  (availability_365, room_type_1, reviews_per_mo...  ...      0.000873
862  (availability_365, room_type_1, price, neighbo...  ...      0.000000
863  (availability_365, room_type_1, price, minimum...  ...      0.000873

[864 rows x 10 columns]
```

Fig 33. Apriori

Recommendations:

a. **What did you learn from this project?**
From this project, we learned that applying machine learning algorithms to the Airbnb NYC dataset can help us classify the listings based on their attributes such as price, location, and availability. We also learned that data preprocessing, including data cleaning, feature selection, and normalization, is crucial in improving the performance of machine learning classifiers. Furthermore, we learned that different classifiers have different strengths and weaknesses in predicting the target variable, and it's important to select the appropriate one based on the project's objectives.

b. **Which classifiers perform the best for the selected dataset?**
Among the classifiers we tested, Random Forest performed the best for the selected Airbnb NYC dataset, achieving an accuracy of 83%. The SVM and Neural Network classifiers also performed well, achieving an accuracy of 81% and 80%, respectively. However, Naïve Bayes performed poorly with an accuracy of only 59%.

c**. How do you think you can improve the performance of the classification? This could be in the future work section.**
There are several ways to improve the performance of the classification for the Airbnb NYC dataset. Some potential areas for future work include:

1. Incorporating additional features: We can try to incorporate additional features into the dataset, such as the proximity of the listing to tourist attractions or the condition of the property.
2. Hyperparameter tuning: By optimizing the hyperparameters of the classifiers, we can potentially improve their performance. For example, we can use grid search or random search to find the best combination of hyperparameters for each classifier.
3. Ensemble methods: We can use ensemble methods such as stacking or bagging to combine the predictions of multiple classifiers and potentially improve their overall accuracy.
4. More data: By collecting more data, we can potentially improve the performance of the classifiers by providing them with a larger and more diverse dataset to learn from.

Overall, by implementing these strategies, we can potentially improve the performance of the classification for the Airbnb NYC dataset and provide better insights to hosts and travelers alike.

Conclusion

After performing various analyses on the NYC Airbnb dataset, we can conclude that Random Forest algorithm has performed the best among all the other machine learning classifiers with an accuracy of 84%. The other machine learning classifiers like KNN, SVM, Naïve Bayes and Neural Network also performed reasonably well, with accuracy ranging from 59% to 81%.
In the preprocessing phase, we have performed outlier analysis, variable transformation, dimensionality reduction, and collinearity analysis. In the modeling phase, we have performed OLS regression, stepwise regression, adjusted R-squared analysis, and association analysis using F-test. We have also performed several machine learning algorithms like decision tree, logistic regression, KNN, SVM, Naïve Bayes, Neural Network, and Random Forest.
Further, the association analysis using Apriori algorithm identified some of the interesting rules which can be useful for the hosts to increase the booking rates. For instance, the rule which suggests that when the price is higher and the minimum nights are longer, then the cancellation rate will also be higher, can be used by the hosts to adjust their pricing and minimum nights for booking to reduce the cancellation rate.
In conclusion, we can say that the Random Forest algorithm has provided us with the best classification results. However, further improvements can be made by collecting more data, feature engineering, and fine-tuning the hyperparameters of the models to achieve even better accuracy.

References:

1. Chen, C., & Xie, K. (2017). Online Reviews and Product Sales: The Moderating Role of Signal Characteristics. *Information & Management*, 54(3), 336-348.
2. Guttentag, D. (2015). Airbnb: disruptive innovation and the rise of an informal tourism accommodation sector. *Current Issues in Tourism*, 18(12), 1192-1217.
3. Lee, D., & Hyun, W. (2017). Evaluating the Impacts of Airbnb Regulation Policies: A New York Case Study. *Sustainability*, 9(10), 1836.
4. McAteer, E., & Stewart, D. (2019). The impact of Airbnb on the hotel industry in New York City. *International Journal of Hospitality & Tourism Administration*, 20(4), 430-452.
5. Ramalho, R., & Fortuna, M. (2019). A Data-Driven Approach to Predict the Success of Airbnb Listings. In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR,* 15-25.
6. Wang, D., & Nicolau, J. L. (2017). Price determinants of sharing economy based accommodation rental: A study of listings from 33 cities on Airbnb.com. *International Journal of Hospitality Management*, 62, 120-131.
7. Zervas, G., Proserpio, D., & Byers, J. W. (2017). The rise of the sharing economy: Estimating the impact of Airbnb on the hotel industry. *Journal of Marketing Research*, 54(5), 687-705.

Appendix

main.py

```
import pandas as pd
import numpy as np

# Import your custom modules
import exploratory_data_analysis
import regression_analysis
import classification_analysis
import clustering_analysis
from sklearn.model_selection import train_test_split

# Load the dataset
def load_data(file_path):
    data = pd.read_csv(file_path)
    return data

def main():
    # Load the dataset
    file_path = 'nyc_airbnb.csv'
    data = load_data(file_path)

    # Perform exploratory data analysis
    exploratory_data_analysis.perform_eda(data)

    # Prepare data for regression
    X_reg = data.drop('price', axis=1)
    y_reg = data['price']
    X_train_reg, X_test_reg, y_train_reg, y_test_reg =
train_test_split(X_reg, y_reg, test_size=0.2, random_state=5525)

    # Perform regression analysis
    regression_analysis.perform_regression(X_train_reg,
y_train_reg, X_test_reg, y_test_reg)

    # Prepare data for classification
    X_cls = data.drop('room_type', axis=1)
    y_cls = data['room_type']
    X_train_cls, X_test_cls, y_train_cls, y_test_cls =
train_test_split(X_cls, y_cls, test_size=0.2, random_state=5525)

    # Perform classification analysis
    results_classification =
classification_analysis.perform_classification(X_train_cls,
y_train_cls, X_test_cls, y_test_cls)
    print("Classification Results:")
    for classifier, result in results_classification.items():
        print(f"{classifier}:")
        for metric, value in result.items():
            if metric == 'Confusion Matrix':
                print(f"{metric}:")
                print(value)
            else:
                print(f"{metric}: {value}")
        print()
    # Perform clustering and association analysis
    n_clusters = 3
    min_support = 0.01
    cluster_labels, rules =
clustering_analysis.perform_clustering_and_association_analysis
(data, n_clusters, min_support)



if __name__ == "__main__":
    main()
```

<u>exploratory_data_analysis.py</u>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cdist
from sklearn.neighbors import LocalOutlierFactor

from sklearn.impute import SimpleImputer
```

```
def data_cleaning(data):

    data.drop(['id', 'name', 'host_id', 'host_name', 'last_review',
'latitude', 'longitude', 'number_of_reviews',
               'calculated_host_listings_count'], axis=1, inplace=True)

    # Fill missing or NaN values with suitable values or drop them
    imputer = SimpleImputer(strategy='median')
    numeric_cols = data.select_dtypes(include=['int64',
'float64']).columns
    data[numeric_cols] = imputer.fit_transform(data[numeric_cols])

    # Convert non-numeric features to numeric using LabelEncoder
    categorical_cols =
data.select_dtypes(include=['object']).columns
    le = LabelEncoder()
    for col in categorical_cols:
        data[col] = le.fit_transform(data[col])

    return data


def dimensionality_reduction(data):
    # Perform Random Forest Analysis
    print("Random Forest Analysis:")
    target = 'price'  # change 'room_type' to 'price'
    attributes = data.drop(target, axis=1).columns
    rf = RandomForestClassifier(n_estimators=50)
    rf.fit(data[attributes], data[target])
    feature_importances = pd.Series(rf.feature_importances_,
index=attributes)
    top_features =
feature_importances.sort_values(ascending=False).head(10)
    print(top_features)


    # Perform Principal Component Analysis
    print("Principal Component Analysis:")
    pca = PCA(n_components=2)
    pca_result = pca.fit_transform(data[attributes])
    print("Explained variance ratio:",
pca.explained_variance_ratio_)

    # Perform Singular Value Decomposition Analysis
    print("Singular Value Decomposition Analysis:")
    U, s, VT = np.linalg.svd(data[attributes], full_matrices=False)
    print("Singular values:", s)

    return top_features.index, pca_result, s


def discretization_binarization(data):
    # One-hot encoding for categorical variables
    data = pd.get_dummies(data, columns=['room_type'],
prefix=['room_type'])
    print("One-Hot Encoding:")
    print(data.head())
```

```
def variable_transformation(data):
    # Normalization/Standardization
    attributes = data.drop('room_type', axis=1).columns
    scaler = StandardScaler()
    data.loc[:, attributes] = scaler.fit_transform(data[attributes])
    print("Standardized Data:")
    print(data[attributes].head())


def anomaly_detection_outlier_analysis(data):
    # Anomaly detection using Local Outlier Factor
    lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
    outliers = data[lof.fit_predict(data) == -1]
    print("Outliers detected:")
    print(outliers)


def covariance_heatmap(data):
    # Sample covariance matrix heatmap
    cov_matrix = data.sample(frac=0.1).cov()
    plt.figure(figsize=(10, 8))  # adjust the figure size as desired
    sns.set(font_scale=1.0)  # adjust the font size as desired
    sns.heatmap(cov_matrix, annot=True, fmt='.2f',
cmap='coolwarm')
    plt.title("Sample Covariance Matrix Heatmap")
    plt.show()

def pearson_correlation_heatmap(data):
    # Sample Pearson correlation coefficients matrix heatmap
    corr_matrix = data.sample(frac=0.1).corr()
    sns.heatmap(corr_matrix, annot=True, fmt='.2f',
cmap='coolwarm')
    plt.title("Sample Pearson Correlation Coefficients Matrix
Heatmap")
    plt.show()


def perform_eda(data):
    # Set figure size and font size
    sns.set(rc={'figure.figsize': (12, 8)})
    sns.set(font_scale=1.2)

    # Data cleaning
    data = data_cleaning(data)
    print("First five rows:")
    print(data.head())
    print("Missing values:")
    print(data.isnull().sum())

    # Dimensionality reduction
    top_features, pca_result, svd_result =
dimensionality_reduction(data)
    data = data[top_features.tolist() + ['price']]  # Keep only the top
10 features and the target variable

    # Discretization and binarization
    discretization_binarization(data)

    # Variable transformation
    variable_transformation(data)
    # Anomaly detection and outlier analysis
    anomaly_detection_outlier_analysis(data)

    # Sample covariance matrix heatmap
    covariance_heatmap(data)

    # Sample Pearson correlation coefficients matrix heatmap
    pearson_correlation_heatmap(data)
```

regression_analysis.py

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from sklearn.feature_selection import f_regression
from sklearn.linear_model import LinearRegression
from scipy import stats
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

def t_test_analysis(X_train, y_train):
    X_train = sm.add_constant(X_train)
    model = sm.OLS(y_train, X_train).fit()
    print(model.summary())
    print("\nT-test Analysis:")
    print(model.t_test(np.identity(len(X_train.columns))))

def association_analysis(X_train, y_train):
    f_values, p_values = f_regression(X_train, y_train)
    print("\nAssociation Analysis (F-test):")
    print("F-values:", f_values)
    print("p-values:", p_values)

def final_regression_model(X_train, y_train, X_test, y_test):
    model = LinearRegression().fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print("\nFinal Regression Model:")
    print("Coefficients:", model.coef_)
    print("Intercept:", model.intercept_)

def confidence_interval_analysis(X_train, y_train):
    X_train = sm.add_constant(X_train)
    model = sm.OLS(y_train, X_train).fit()
    print("\nConfidence Interval Analysis:")
    print(model.conf_int())

def stepwise_regression_analysis(X_train, y_train):
    X_train = sm.add_constant(X_train)
    model = sm.OLS(y_train, X_train).fit()
    print("\nStepwise Regression and Adjusted R-square
Analysis:")
    print("AIC:", model.aic)
    print("BIC:", model.bic)
    print("Adjusted R-square:", model.rsquared_adj)

def collinearity_analysis(X_train):
    vif = pd.DataFrame()
    vif["Features"] = X_train.columns
    vif["VIF"] = [variance_inflation_factor(X_train.values, i) for i in
range(X_train.shape[1])]
    print("\nCollinearity Analysis (VIF Method):")
    print(vif)

def perform_regression(X_train, y_train, X_test, y_test):
    t_test_analysis(X_train, y_train)
    association_analysis(X_train, y_train)
    final_regression_model(X_train, y_train, X_test, y_test)
    confidence_interval_analysis(X_train, y_train)
    stepwise_regression_analysis(X_train, y_train)
    collinearity_analysis(X_train)

    # Add polynomial features
    poly = PolynomialFeatures(degree=2)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    # Lasso regression with cross-validated alpha selection
    lasso = Lasso(random_state=42, max_iter=10000)
    param_grid = {'alpha': np.logspace(-4, 0, 20)}
    lasso_cv = GridSearchCV(lasso, param_grid, cv=5,
scoring='r2')
    lasso_cv.fit(X_train_poly, y_train)
    # Print best alpha and R-squared
    print(f"Best alpha: {lasso_cv.best_params_['alpha']}")
    print(f"R-squared (train): {lasso_cv.score(X_train_poly,
y_train)}")
    print(f"R-squared (test): {lasso_cv.score(X_test_poly, y_test)}")

    # Train Lasso with the best alpha
    lasso_best = Lasso(alpha=lasso_cv.best_params_['alpha'],
random_state=42, max_iter=10000)
    lasso_best.fit(X_train_poly, y_train)
```

classification_analysis.py

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
```

```
def split_data(data, target):
    X = data.drop(target, axis=1)
    y = data[target]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=5525)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    return X_train, X_test, y_train, y_test


import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,
precision_recall_fscore_support, roc_curve, auc
from sklearn.preprocessing import label_binarize

def perform_classification(X_train, y_train, X_test, y_test):
    classifiers = [
        ('Decision Tree', DecisionTreeClassifier()),
        ('Logistic Regression', LogisticRegression(max_iter=1000)),
        ('KNN', KNeighborsClassifier()),
        ('SVM', SVC(probability=True)),
        ('Naïve Bayes', GaussianNB()),
        ('Random Forest', RandomForestClassifier()),
        ('Neural Network', MLPClassifier(max_iter=1000))
    ]

    results = {}
    for name, clf in classifiers:
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        report = classification_report(y_test, y_pred,
output_dict=True, zero_division=1)

        cm = confusion_matrix(y_test, y_pred)
        precision, recall, f_score, _ =
precision_recall_fscore_support(y_test, y_pred,
average='weighted', zero_division=1)
        specificity = cm.diagonal() / cm.sum(axis=1)
        specificity = np.mean(specificity)

        y_prob = clf.predict_proba(X_test)
        y_test_binarized = label_binarize(y_test,
classes=np.unique(y_test))
        fpr, tpr, _ = roc_curve(y_test_binarized.ravel(),
y_prob.ravel())
        roc_auc = auc(fpr, tpr)

        # Plot ROC curve
        plt.figure()
        plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
        plt.plot([0, 1], [0, 1], 'k--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'ROC Curve for {name}')
        plt.legend(loc="lower right")
        plt.show()

        results[name] = {
            'Confusion Matrix': cm,
            'Precision': precision,
            'Recall': recall,
            'Specificity': specificity,
            'F-score': f_score,
            'Accuracy': accuracy,
            'Report': report
        }

    return results
```

clustering_analysis.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from mlxtend.frequent_patterns import apriori, association_rules


def k_means_clustering(data, n_clusters):
    # Preprocess the data for K-means clustering
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(data)
    # Perform K-means clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state=42,
n_init=10)
    kmeans.fit(scaled_data)
    cluster_labels = kmeans.labels_

    return cluster_labels


def binarize_data(data, threshold_dict, categorical_columns):
    binarized_data = data.copy()
    for column in data.columns:
        if column in categorical_columns:
            dummies = pd.get_dummies(data[column], prefix=column)
            binarized_data = pd.concat([binarized_data, dummies],
axis=1)
            binarized_data = binarized_data.drop(columns=[column])
        elif column in threshold_dict:
            binarized_data[column] = (data[column] >=
threshold_dict[column]).astype(int)
    return binarized_data.astype(bool)


def perform_apriori_algorithm(data, min_support):
    # Perform Apriori algorithm
    frequent_itemsets = apriori(data, min_support=min_support,
use_colnames=True)

    # Generate association rules
    rules = association_rules(frequent_itemsets,
metric="confidence", min_threshold=0.7)
    return rules


def perform_clustering_and_association_analysis(data,
n_clusters=3, min_support=0.01):
    # Perform K-means clustering
    cluster_labels = k_means_clustering(data, n_clusters)

    # Add the cluster labels to the data
    data['Cluster'] = cluster_labels

    # Binarize the dataset using custom thresholds
    threshold_dict = {
        'price': 100,
        'minimum_nights': 3,
        'number_of_reviews': 10
    }
    categorical_columns = ['neighbourhood_group', 'room_type']
    binarized_data = binarize_data(data, threshold_dict,
categorical_columns)

    # Remove the "Cluster" column from the binarized data
    binarized_data = binarized_data.drop(columns=['Cluster'])

    # Perform Apriori algorithm
    rules = perform_apriori_algorithm(binarized_data, min_support)

    # Print the results
    print("KMeans:")
    print(cluster_labels)
    print("Apriori:")
    print(rules)

    return cluster_labels, rules
```