Thi Lan Vien Doan

Leveraging K-Nearest Neighbors for Rental Price Classification in the US Real Estate Market

1. **INTRODUCTION**

The US real estate market is a dynamic and complex environment in which many factors influence rental prices. Factors such as apartment location, square footage, number of bedrooms, and availability of amenities play a significant role in determining rental prices. However, due to the diversity of these factors, accurately classifying apartments into appropriate rental segments, whether low, mid, or high, can be challenging. Such classification is important for both property managers and tenants, allowing them to make informed decisions about marketing strategies, pricing optimization, and real estate investment.

Given the importance of rental prices in the real estate market, this project aims to develop a classification model that classifies apartments into different rental segments. By leveraging machine learning techniques, specifically the K-Nearest Neighbors (KNN) algorithm, the model will classify apartments based on their attributes.

The report continues the exploration of the "Apartment for Rent Classified" dataset, which is available through the UCI Machine Learning Repository. It investigates existing methods by exploring two popular tools: Python and R. The focus will be on implementing the KNN algorithm to classify apartments into rent segments. The goal is to develop an accurate and efficient classification method proving valuable insights into the rental market. Additionally, the report presents key results and a comparison of the KNN algorithm's performance in both Python and R. Finally, it will conclude with a summary of the findings and insights gained.

1. **SOLUTION**
   1. **Investigation on Existing Methods**

This section focuses on the implementation of the K-Nearest Neighbors (KNN) algorithm. As Adnan Abdulrahman et al. (2020) state, Python can be used for a wide range of mathematical and statistical calculations, as well as for classification pre-processing, data extraction, and prediction. In Python, Scikit-learn is one of the most widely used libraries for machine learning, including classification tasks such as KNN. Its popularity can be attributed to its ease of use, comprehensive documentation, and integration with other essential Python libraries such as Pandas and Numpy (Adnan Abdulrahman et al., 2020).

The KNeighborsClassifier function within Scikit-learn is particularly efficient for KNN classification. It allows users to configure various key parameters, for example, the number of neighbors (n\_neighbors), distance metrics, and weighting functions (Scikit-learn, n.d.). Zollanvari (2023) claims that among these, the most crucial parameter for kNN is k, which represents the number of nearest neighbors of a test point. In the KNeighborsClassifier constructor, this is specified by the n\_neighbors parameter.

In terms of R, it is a language for the analysis and manipulation of statistical data (Theuβl & K. Hornik, 2012, as cited in Moldagulova & Sulaiman, 2017). In R, KNN can be implemented using the class package, which provides the knn() function for classification tasks. Awan (2023) points out that this package is popular due to its simple implementation and flexibility in configuring parameters such as the number of neighbors (k) and distance metrics. The knn() function from the class package is used to classify data from a dataset. The user can specify the training and testing data, as well as the number of neighbors to consider, making the function easy to use for both basic and more complex classification problems. Overall, both Python and R provide powerful implementations of the KNN algorithm through their respective libraries.

* 1. **The Specific Implementations: KNN Algorithm in Python and R**

In this section, the KNN algorithm is implemented in Python (via the scikit-learn library) and R (using the class package) to classify apartments into different rent segments, for example, low, medium, and high, based on attributes such as square\_feet, bathrooms, and bedrooms. The goal is to compare the two implementations, evaluate their performance, and highlight any differences in ease of use, functionality, and accuracy.

* + 1. **KNN Implementation in Python (using Scikit-learn)**
* **Step 1: Importing libraries**

The KNN algorithm is implemented in Python using the scikit-learn library as shown in the screenshot below, which provides utilities for model training, evaluation, and data preprocessing. The required libraries include pandas for data processing, train\_test\_split for splitting the dataset, StandardScaler for normalizing the features, KNeighborsClassifier for the KNN model, and accuracy\_score for evaluating the model performance.



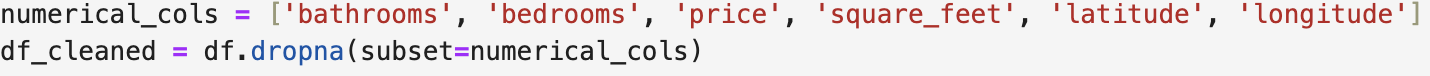
* **Step 2: Loading the dataset**

The dataset is loaded into a pandas DataFrame as shown in the codes below. The focus is on key attributes, including square\_feet, bathrooms, and bedrooms, to predict the price of the apartment.



* **Step 3: Data preprocessing**

The dataset first preprocesses the dataset to handle missing values by dropping rows with missing values in key numerical columns.



After cleaning, to classify apartments into low, medium, and high price segments, the dataset is divided into three categories based on the price column. The dataset is then updated with a new column, price\_segment, which will serve as the target variable.



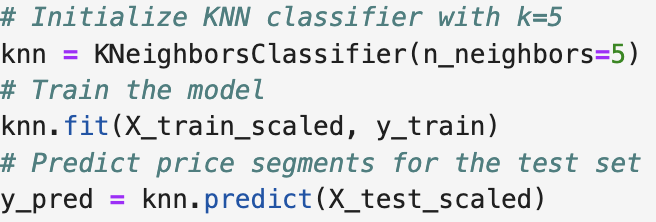
Next, relevant features are selected for training the model, and the dataset is split into training and test sets (80% for training and 20% for testing).

A screenshot of a computer program

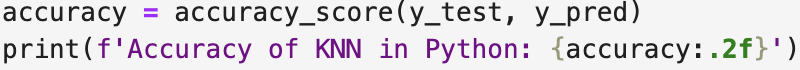
Description automatically generated

* **Step 4: Implementing the KNN Model**

A KNN classifier is initialized with 5 neighbors (k=5). The model is trained using the training data, and predictions are made on the test data.



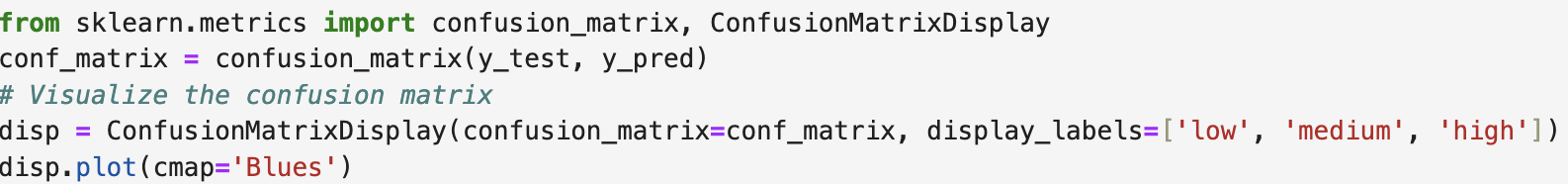
Then, the accuracy of the KNN model is calculated by comparing the predicted price segments with the actual price segments in the test data using the accuracy\_score function.



* **Step 5: Visualizing Results**

This section of the report aims to visualize the distribution of apartments across different rental price segments (low, medium, high). The analysis employs a confusion matrix, bar chart, and model evaluation summary. Particularly, the confusion matrix evaluates the accuracy of the K-Nearest Neighbors (KNN) model's predictions. It illustrates how effectively the model classifies apartments into the correct rental segments. In addition, a bar chart is used to visualize the number of apartments classified into each rental segment. This visual representation helps in understanding the distribution of apartments across the low, medium, and high rental price categories. Moreover, the model evaluation summary includes key metrics such as precision, recall, and f1-score.

* Confusion Matrix:



* Bar chart:



* Evaluation Metrics: Precision, Recall, and F1-Score:



* + 1. **KNN Implementation in R (using the class package)**
* **Step 1: Importing packages and libraries**

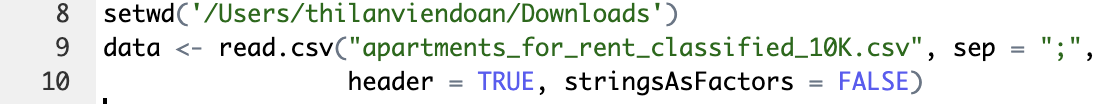
In R, packages and libraries are essential tools that extend the functionality of the base R environment. To install a package, this report show to use the install.packages() function. Once installed, then load the package into R session using the library() function (See codes below).

A screenshot of a computer

Description automatically generated

* **Step 2: Loading the dataset**

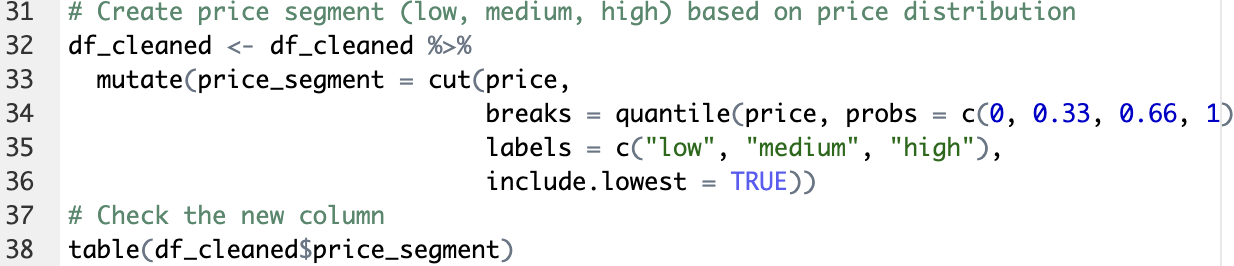
In R, the dataset is loaded using the read.csv() function (See codes below). The focus will again be on the square\_feet, bathrooms and bedrooms columns as features and the price column will be converted to segments similar to the Python implementation.



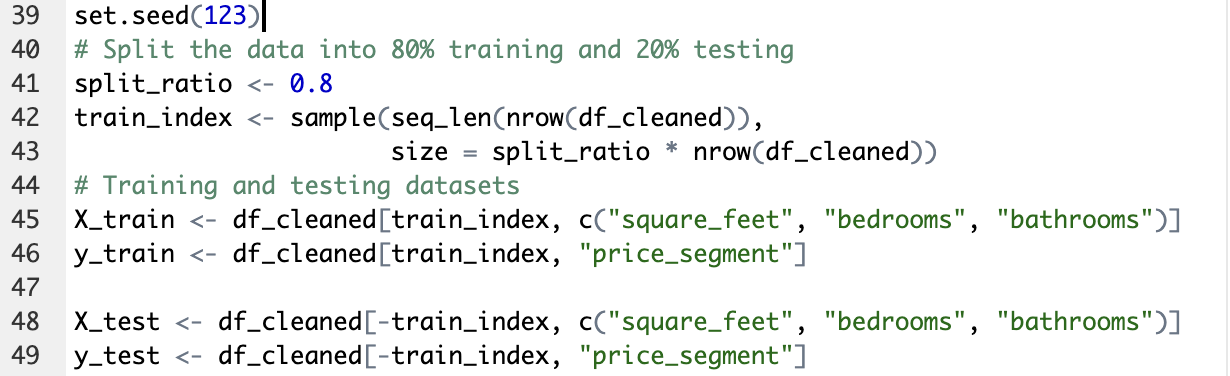
* **Step 3: Preprocessing**

This step is about checking for missing values ​​in the dataset. First, select the key numerical columns then remove rows with missing values in those important columns (square\_feet, bedrooms, bathrooms, price). After that, create the price segment based on the price column by dividing it into three categories: low, medium, and high (See codes below)



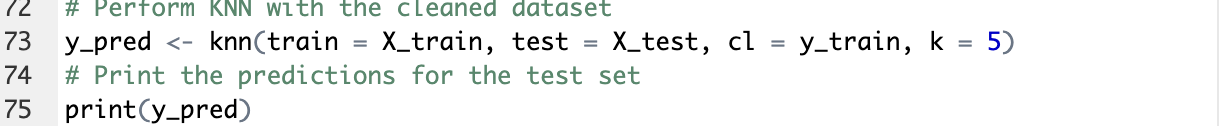


Now, split the data into training and test sets. This part will use the same features (square\_feet, bedrooms, bathrooms) as in Python.



* **Step 4: Implementing the KNN Model**

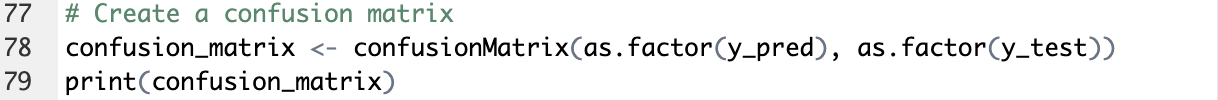
Now that the data is preprocessed, this step will apply the KNN algorithm with k=5.



* **Visualizing Results**

To visualize the confusion matrix in R, this section uses the ggplot2 package to create a detailed chart that clearly shows the performance of the model across different classes. Additionally, the distribution of segments (low, medium, high) in the test set can be visualized using a bar chart, providing details on the number of apartments falling within each rental price segment. Evaluating the performance of the KNN model involves using the confusion matrix to calculate key metrics such as accuracy, precision, recall, and F1 score, which together evaluate the model's effectiveness in predicting the correct segments.

* Confusion matrix



* Bar chart: Segment distribution

A screen shot of a computer code

Description automatically generated

* Evaluation Metrics: Precision, Recall, and F1-Score



1. **KEY RESULTS AND METRICS**
   1. **Report the experimental results from Python**

|  |  |
| --- | --- |
| A blue squares with white text  Description automatically generated  Figure 1. Confusion Matrix | In terms of confusion matrix, the result above (see Figure 1) provides a detailed breakdown of the model’s performance by showing the actual versus predicted classifications for each rental segment (low, medium, high). |

* Low segment: The model correctly classified 358 apartments, misclassified 157 as medium and 167 as high. This indicates moderate success, but frequent confusion with high-priced apartments.
* Medium segment: The model correctly classified 364 apartments, misclassified 167 as low, and 123 as high. This segment has the highest number of correct classifications. However, there is still significant misclassifications with low-priced apartments
* High segment: The model correctly classified 182 apartments but misclassified 254 as low and 218 as medium. This segment has the lowest number of correct classifications and the highest misclassification rate, especially with low-priced apartments.

Regarding to segment distribution, the bar chart below (see Figure 2) illustrates the distribution of apartments across different rental price segments, showing 3327 apartments in the medium segment, 3320 in the high segment, and 3303 in the low segment. This relatively even distribution across the medium and high segments, with slightly fewer apartments in the low segment, is generally favourable for model training. However, the similar counts in the medium and high segments might contribute to the misclassification issues observed, as the model may struggle to distinguish between these two segments effectively.

A graph of a number of bars

Description automatically generated with medium confidence

Figure 2. Segment Distribution

|  |  |
| --- | --- |
| The performance metrics (see Figure 3), including precision, recall, and F1-score, provide a quantitative measure of the model’s effectiveness. | A screenshot of a graph  Description automatically generated  Figure 3.Overall Performance Metrics |

* Precision: 0.46 for the low segment, 0.49 for the medium segment, and 0.39 for the high segment. It indicates the highest accuracy for medium segment.
* Recall: 0.52 for the low segment, 0.56 for the medium segment, and 0.28 for the high segment. It correctly identifies 56% of medium segment but misses many high ones.
* F1-score: 0.49 for the low segment, 0.52 for the medium segment, and 0.32 for the high segment. The model performs best in the medium segment.
  1. **Report the experimental results from R**

The confusion matrix results (See Figure 4) show that the KNN model correctly predicts the rental price segments (low, medium, high) 46.15% of the time. Specifically, it accurately identified 370 low, 205 medium, and 348 high apartments. However, there were also misclassifications, such as predicting 233 medium apartments as low and 199 high apartments as medium. The model's accuracy falls within a 95% confidence interval of 43.95% to 48.36%, indicating some variability in performance. The No Information Rate (NIR), which represents the accuracy of always predicting the most frequent class, is 35.2%. The p-value of less than 2.2e-16 shows that the model's accuracy is significantly better than random guessing.

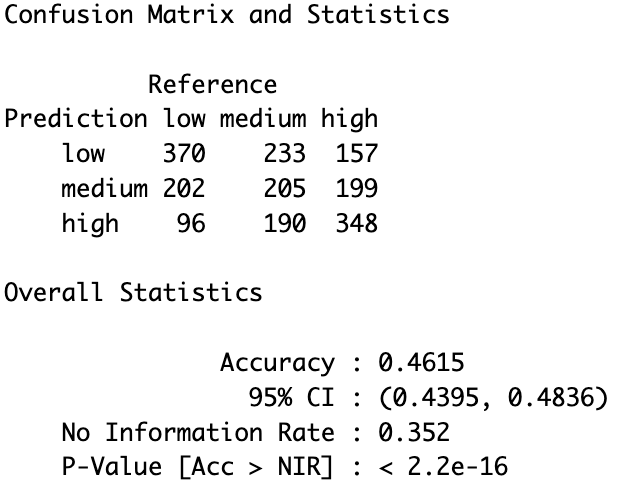


Figure 4. Overall performance Metrics

|  |  |
| --- | --- |
| A graph of a number of colored squares  Description automatically generated with medium confidence  Figure 5. Segment Distribution | The bar chart (See Figure 5) shows the distribution of apartments across three rental price segments: low, medium, and high. The low segment has 3,314 apartments, the medium segment has 3,310, and the high segment has 3,376. This indicates a relatively even distribution of apartments across all price segments. |

Next, the confusion matrix analysis reveals the performance of a classification model across three classes: low, medium, and high. For the low class, the model shows a sensitivity of 55.8% and a specificity of 77.1%, indicating moderate accuracy in identifying low cases. The medium class has a lower sensitivity of 36.4% and a specificity of 81.3%, reflecting challenges in correctly identifying medium cases. The high class has a sensitivity of 49.4% and a specificity of 52.0%, suggesting the model struggles with high cases.

A screenshot of a computer screen

Description automatically generated

Figure 6. Overall Performance Metrics

* 1. **Compare results from Python and R**

Both models moderately succeeded in classifying rental segments, but their performance metrics differed significantly. The Python model demonstrates an overall accuracy of 45%, with the highest precision, recall, and F1 score in the medium segment, indicating better performance in identifying medium segment. However, it struggles significantly with high segment, showing frequent misclassifications.

In contrast, the R model achieves a slightly higher accuracy of 46.15%, with its 95% confidence interval. It also demonstrates significant improvement over random guessing, as evidenced by its p-value. The sensitivity and specificity highlight moderate accuracy in identifying low segment but challenges in accurately classifying medium and high segment. Both models perform consistently across all classes without bias. However, the R model’s statistical measures provide a more detailed understanding of its variability and reliability.

1. **SUMMARY**

This project uses the KNN algorithm in both Python and R to classify US apartments into rental segments (low, mid, high) based on factors such assquare feet, bedrooms, and bathrooms. The process includes importing relevant libraries, loading the dataset, preprocessing the data, training the model, and evaluating the performance on Python and R. The implementation has been successful, with efficient handling of missing values and categorization of apartments into rental segments. The models were trained and evaluated using accuracy, precision, recall, and F1-score. The results were visualized using confusion matrices and bar charts. However, challenges were encountered in finding the optimal number of neighbors (k) and distance metrics for the KNN algorithm, as well as dealing with potential data imbalances across different rental segments. The main differences are noted in ease of use, functionality, and accuracy. Python's KNN implementation was smooth and efficient, while R's faced challenges with dataset loading and cleaning. R provided more detailed statistical insights, including a 95% confidence interval (43.95% to 48.36%) and p-values, enhancing model reliability understanding.

**References**

Adnan Abdulrahman, S., Roushdy, M., & Salem, A. B. M. (2020). Using K-nearest neighbors and support vector machine classifiers in personal identification based on EEG signals. *International Journal of Computer Science and Information Security (IJCSIS)*, *18*(5).

Awan, A. A. (2023, June 14). *K-Nearest Neighbors (KNN) classification with R tutorial*. DataCamp. <https://www.datacamp.com/tutorial/k-nearest-neighbors-knn-classification-with-r-tutorial>

Moldagulova, A., & Sulaiman, R. B. (2017). Using KNN algorithm for classification of textual documents. In *2017 8th international conference on information technology (ICIT)* (pp. 665-671). IEEE.

Scikit-learn. (n.d.). *KNeighborsClassifier*. <https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

Zollanvari, A. (2023). Supervised Learning in Practice: the First Application Using Scikit-Learn. In *Machine Learning with Python: Theory and Implementation* (pp. 111-131). Cham: Springer International Publishing.

**APPENDIX**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer code

Description automatically generated

A green and orange bar chart

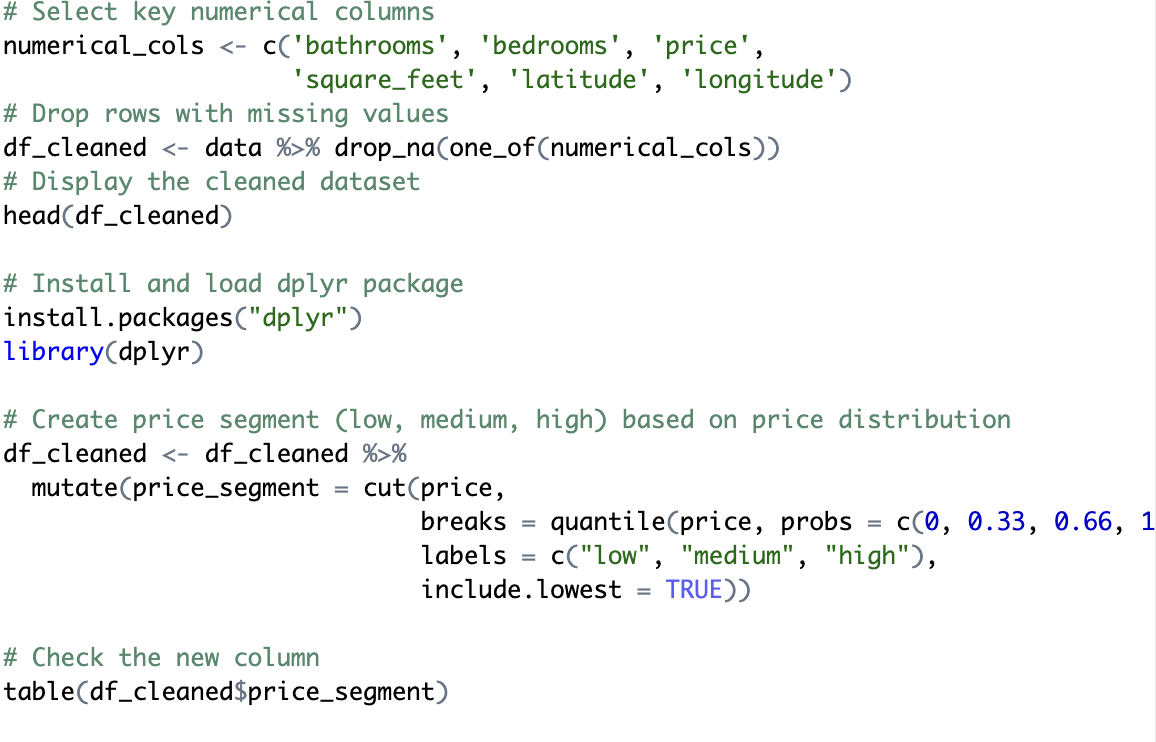
Description automatically generated

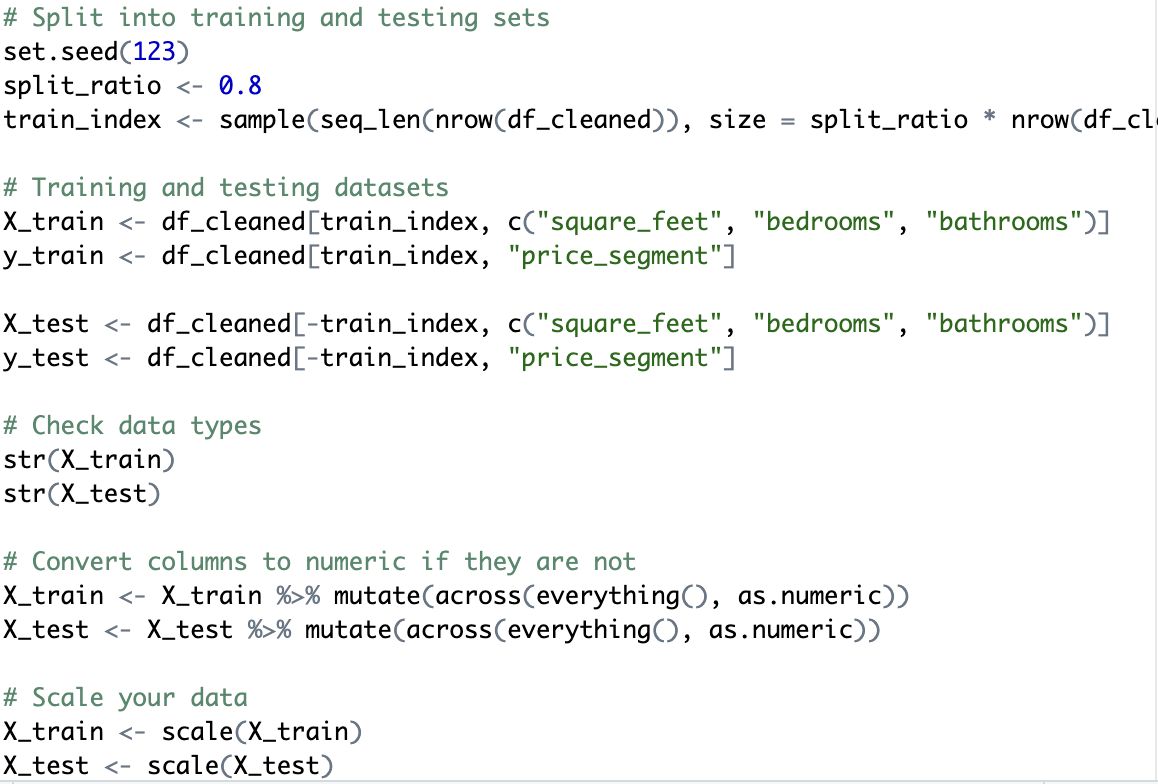
A screenshot of a computer

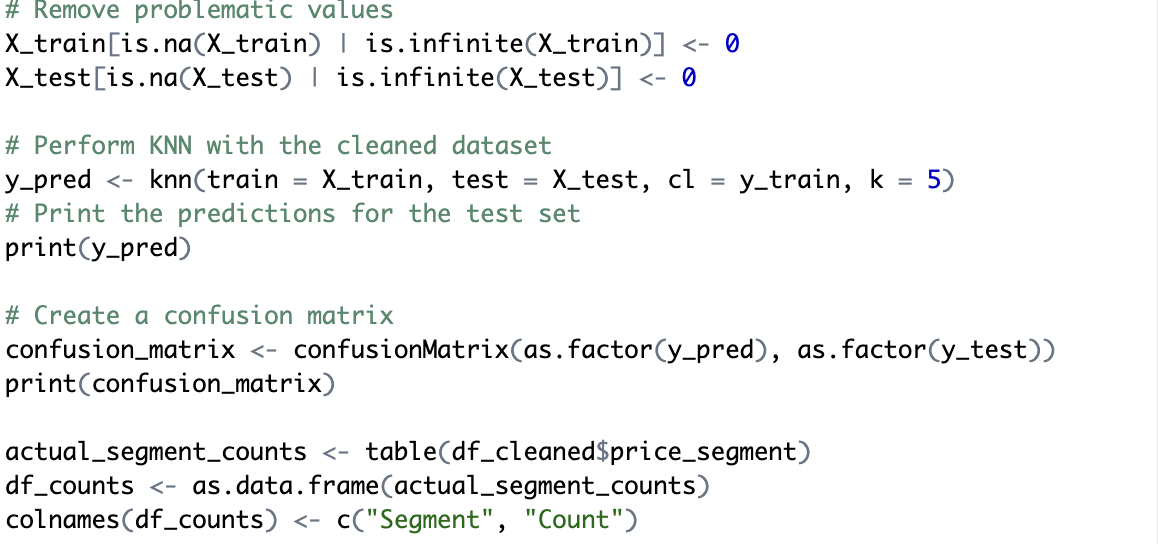
Description automatically generated

The next part is for R







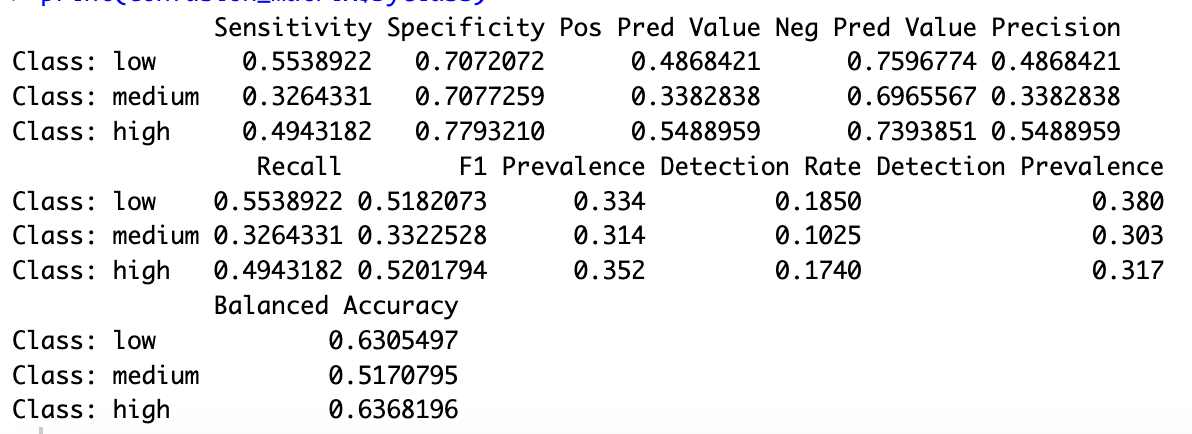


A computer code with green and white text

Description automatically generated

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

