MAST6100: MACHINE LEARNING AND DEEP LEARNING

Final Project - Report

DECEMBER 1, 2025

SARAH LEE LWP – WL259

Table of Contents

Executive Summary

Hospitals worldwide are increasingly facing pressure from overcrowding, extended waiting times, and rising healthcare costs. Hospital admissions are the pivotal for patients to access timely care and to efficiently use the limited healthcare resources. Inappropriate or delayed admissions can negatively affect patient health and strain hospital capacity which highlights the importance of better understanding the factors that influence admission decisions.

Hospital admissions are shaped by a combination of patients' clinical, demographic, and administrative factors. Gaining insight into how these factors can affect the type of hospital admission can potentially more efficiently support the allocation of resources, improve the delivery of healthcare services, and better inform healthcare policymakers. Motivated by these challenges, this study investigates whether the available patient data can be used to differentiate between the different hospital admission types through data-driven analytical approaches.

The study starts with an exploratory examination of the dataset to assess variable distributions, relationships, and overall data quality. The initial analysis provides an understanding of the underlying structure of the data and supports the subsequent modelling decisions. Then, a range of statistical and machine learning techniques are applied to evaluate the extent to which admission types can be explained or predicted using the given or chosen predictors. Model performance is assessed using standard evaluation metrics and cross-validation to ensure robustness and generalisability of the model.

Instead of using just a single modelling approach, this study implements a more comparative framework by examining both the traditional statistical models and more flexible machine learning and deep learning methods. This approach allows to assess whether an increased model complexity may lead to improvements in performance, or if the data itself has inherent limitations.

Overall, this report provides a logical evaluation of the relationship between available patients' characteristics and hospital admission types. While the findings mainly highlight the challenges of predicting hospital admission types using routinely collected data, they also offer valuable insight into the strengths and limitations of different analytical approaches. These insights can help guide future research, advise data collection strategies, and support healthcare providers and policymakers to improve hospital efficiency and patient care.

Introduction

Date of report: 1 December 2025

University name: University of Kent

Written by: Sarah Lwp Lee (24022677 – Stage 3 Actuarial Science undergraduate)

This report was prepared by Sarah Lee, in their capacity as a Stage 3 Actuarial Science undergraduate student at the University of Kent with the responsibility of carrying out an analysis on a self-chosen dataset.

Intended recipients(s): Alfred Kume (Lecturer)

Background and Motivation

Globally, hospitals are currently facing multiple challenges such as overcrowding, extended waiting times and increasing healthcare costs. Amongst these challenges, hospital admissions play a vital role in determining patients' access to immediate medical care. Hence, both avoidable admissions and delayed necessary admissions have emerged as a significant issue in ensuring the appropriate use of hospital services, improving patient outcomes, and maintaining the efficiency of healthcare systems.

Admission decisions can be influenced by multiple factors such as clinical results, demographic and socioeconomic background. Therefore, there is a need for improved understanding of the factors driving hospital admissions.

The motivation for conducting this research is to promote accessible healthcare and more efficiently allocate the limited hospital resources. By identifying and analysing both clinical and non-clinical factors, the findings are expected to inform healthcare providers and policymakers in developing strategies that enhance the quality and efficiency of patient care.

Aim

The aim of this research is to examine what are the factors that potentially may affect the type of hospital admission (Elective, Urgent or Emergency admissions).

Objectives and Scope

Analysing patient-related information's association to hospital admission type, including:

- **Age (Years):** Age of the patient at the time of admission, expressed in years.
- **Gender:** Indicates the gender of the patient, either "Male" or "Female."
- **Blood Type:** Patient's blood type
- **Medical Condition:** Primary medical condition or diagnosis associated with the patient, including "Diabetes", "Hypertension", "Asthma", “Obesity”, “Arthritis” and “Cancer”

- **Insurance Provider:** Patient's insurance provider, including "Aetna", "Blue Cross", "Cigna", "UnitedHealthcare" and "Medicare"
- **Billing Amount:** The amount of money billed for the patient's healthcare services during their admission
- **Room Number:** The room number of the admitted patient
- **Medication:** The medication prescribed or administered to the patient during their admission, including "Aspirin", "Ibuprofen", "Penicillin", "Paracetamol" and "Lipitor"
- **Test Results:** The results of a medical test conducted during the patient's admission, including "Normal", "Abnormal," or "Inconclusive"

Sources of Data

In this report, data is primarily sourced from kaggle.com. Additionally, analysis is conduct using:

- Version of R in use: R 4.5.2
- Version of RStudio in use: 2025.09.2+418

To ensure the Rmarkdown file runs properly and avoid errors from outdated versions, please use the same version of R and RStudio.

Github link to access all original files: https://github.com/Sarah-Lwp-Lee/MAST6100---Final-Project.git

Limitations

This report is subject to (but not limited to) the following limitations:

- **Methodological Constraints:** Methods used in this report may or may not show or capture all potential relationships
- **Data Quality Issues:**
  - **Data Coverage:** Data only covers admitted patients
  - **Test Results:** Do not know what type of medical test was conducted
- **Time Constraints:** Due to the limited time available, this analysis did not permit a deeper examination and understanding of the data. Hence, the conclusions drawn may not fully reflect all aspects of the dataset.

While this report provides valuable insights, it is important to recognise that some assumptions made during the analysis, such as ignoring the non-existing data, could influence the accuracy of the results and analysis. Therefore, the interpretation of the discussion and analysis should be done with caution due to the inherent limitations of data and the assumptions used.

To address the limitations of this report, please refer to the recommendations section.

Methodology

Data Cleaning and Quality Checking

(Please refer to Appendix 2)

Data cleaning and quality checking is used to ensure the data is free of major logical errors and ensure that the data is easy to work with.

Data cleaning and quality checking were done using the following steps:

1. **Quick Data Diagnose (Appendix 2.2):** A preliminary check on the data
2. **Check For Missing Values (Appendix 3.3):** Removing the missing values
3. **Correcting Letters (Appendix 2.4):** For consistency and accuracy in data cleaning and further analysis, all letters are changed to small letters
4. **Check For Duplicates (Appendix 2.5):** To ensure that duplicates will not affect the models, duplicates are removed
5. **Checking For Negative Values (Appendix 2.6):** Some negative values are not logical, so they are removed to prevent it from affecting the models
6. **Check The Type of Data (Appendix 2.7):** Each column's type was changed to its appropriate type so that models will not have errors
7. **Removing Unnecessary Columms (Appendix 2.8):** Some columns were only useful for data cleaning and quality checking, so they are removed

Exploratory Data Analysis (EDA)

(Please refer to Appendix 3)

Summary of data and multiple types of plots, such as boxplots, histograms, pie charts and bar charts were used to examine the key points and basic patterns in each variable. Correlation between the numeric variables were also used to look for preliminary relationships.

Variable Selection and Regression

(Please refer to Appendix 4)

From here, admission type is defined as the response variable and all other variables and defined as independent variables (variables that can affect the response variable).

LASSO and Elastic Net was used to select the variables used in the regression model. However, it was proven that the selected variables are not any better than using all the variables. Hence, the full model using all the variables was fitted and used.

The regression model used was multinomial logistic regression as the response variable has 3 categorical outcomes. Dataset was split into 80% training set and 20% testing data. Then, model was fitted using the training set. After that, the fitted model was cross-validated using k-fold cross-validation and the confusion matrix to determine the accuracy and how much better was the model than just random guessing.

The odds ratio, standard errors and confidence intervals of each independent variable was also considered to determine the degree of effect the variable has on admission types.

Classification and Deep Learning

(Please refer to Appendix 5 and Appendix 6)

Dataset was split into 80% training set and 20% testing data. Then, instructions were set on how to train and evaluate the models. After that, the training set was used to make the classification models while the test set was used to evaluate the model's accuracy and how much better was the classification model than just random guessing.

The classification models used are:

- **Random Forests**
- **K-Nearest Neighbours (KNN)**
- **Linear Discriminant Analysis (LDA)**

Similarly in deep cleaning, the dataset was split into 80% training set and 20% testing data. Thereafter, the data was standardised before being used in the neural network.

Neural network has a 30% dropout rate in each of the 2 layers before the final layer to prevent overfitting. Then, the neural network model is compiled and trained while the validation accuracy and validation loss is tracked in the plot below:

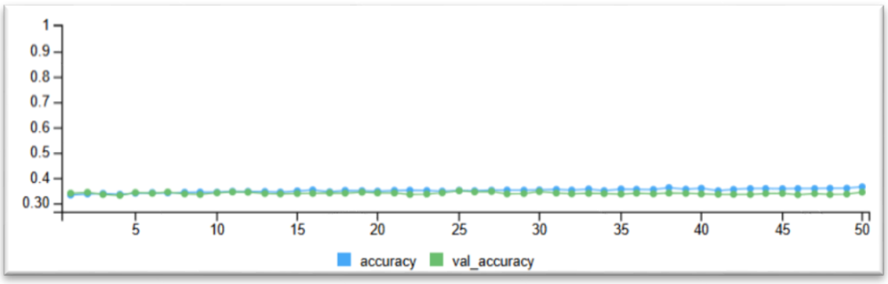


Figure 1: Accuracy Plot

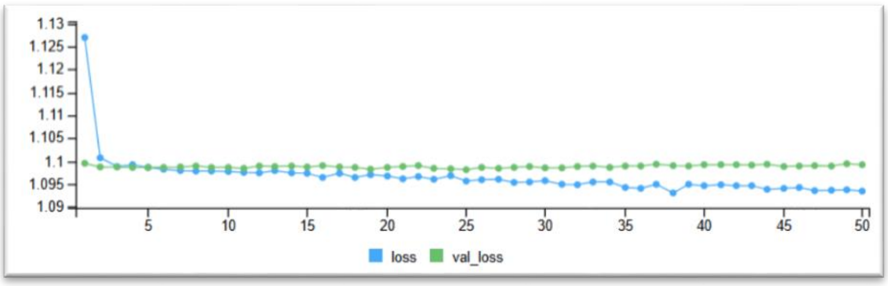


Figure 2: Loss Plot

Then, the neural network is evaluated on the test data. Finally, the overall evaluation is made using the confusion matrix to see the overall accuracy and how much better the deep learning outcomes are than just random guessing.

Analysis and Results

From the EDA, the dataset shows mostly an even distribution across variables. Numeric variables show no significant correlations, indicating that interaction terms are unnecessary for variable selection later. No outliers were detected. In terms of distribution shape of numeric variables, most are uniformly distributed with only a slight skew. As for the categorical variables, they are all approximately evenly split.

During variable selection, the results indicate that the best-fit model and the full model (using all the independent variables) perform similarly. The residuals indicated a moderate-to-poor fit for the best-fit model. Hence, the full model was used as there were little benefits in using the best-fit model.

After fitting the full regression model, the model shows a low prediction accuracy of around 34%, which was expected given the evenly distributed predictors. Furthermore, cross-validation confirms the weak predictive power of the model, with 5-fold and 10-fold accuracies being around 33.5-33.8% and Kappa values near zero, which indicates the model performance to be close to random guessing. Exploratory effects only suggest minor patterns with older patients, males, those who are obese, diabetic, asthmatic, certain insurance providers (notably Cigna and Medicare) and some medications show slightly higher likelihoods of emergency and urgent hospital admissions while other predictors have weak or inconsistent effects.

Across the classification models, the performance remains at chance level, indicating no meaningful improvement over the multinomial regression. The random forest model achieves about 33.1% accuracy and has a negative Kappa which gives worse-than-random performance. Similarly, KNN has around 32.9% accuracy and a negative Kappa. LDA only shows a slightly higher accuracy (33.6%) and a marginally positive Kappa (0.0003) effectively no better than random guessing.

Additionally, the neural network model also performs at chance level, with training and validation accuracy at around 33–35%. Referring to Figure 1, the training and validation accuracy remains close which indicates no overfitting of the neural network. Figure 2 support this, as the training loss drops initially but quickly levels off to 1.1 (approximately log(3), the expected loss from a random classification) with the validation loss, shows that the model fails to extract useful patterns from the data. The test accuracy is 33.5% with a Kappa of 0.0092, the model's performance is no better than random guessing. Although the network appears slightly more sensitive to emergency admissions, its low precision leads to frequent misclassifications.

Overall, all models, including the neural network, perform at chance level and fail to reliably predict admission types. Hence, the weak performance indicates that the available predictors lack informative power to meaningfully distinguish between admission types.

Conclusion

This study assesses whether available patient information could be meaningful in predicting hospital admission types using a range of statistical and machine learning models.

The EDA revealed that the dataset was mostly well-balanced. With numeric variables showing no strong correlations and no outliers, they were approximately uniformly distributed with only slight skewness. Whereas categorical variables were almost evenly split across their respective categories. These characteristics suggested limited inherent structure in the data.

Following that, the fitted multinomial logistic regression model achieved an accuracy of approximately 34%, which was weak but still within expectations. Both the 5-fold and 10-fold cross-validation confirmed the model's weak predictive power. The results indicated that the model's performance was comparable to random guessing. While further exploratory analysis suggested minor tendencies, these effects were minor and inconsistent, offering limited practical predictive value.

When using classification models including random forests, KNN and LDA, all models produced a similar chance-level performance. Random forests and KNN achieved accuracies of approximately 33% with negative Kappa values, indicating worse-than-random predictions. LDA showed a slightly higher accuracy but an effectively zero Kappa, showing no meaningful improvement over other approaches or the baseline regression model.

Finally, a neural network model was built to determine whether more complex, non-linear relationships could be learned from the data. The neural network also performed at the chance level and showed that predictions was no better than random guessing even though there were no indications of overfitting. Although the model did show a slightly higher sensitivity to emergency admissions, its low precision resulted in frequent misclassifications.

In summary, all the models evaluated in this study failed to predict hospital admission types beyond the chance level. This consistent pattern strongly suggests that the primary limitation lies not in model choice or complexity, but that the available variables lack informative power to capture the signal to distinguish between elective, urgent and emergency hospital admissions. Further research would be required with other clinically relevant features, such as detailed medical history or severity indicators, to improve the models' predictive performance.

Recommendations

1. **Improve Data Collection and Integration Practices:** Healthcare systems should invest in better integration of health records and clinical databases. Improved collection of data (e.g. knowing what type of medical test was conducted) can form a more comprehensive patient profile, allowing more effective analytical and predictive modelling.
2. **Enhance Data Quality and Clinical Detail:** Future analyses should include other clinical variables that better reflect the patients' status. These factors may provide stronger predictive signals than just demographic or administrative variables alone.
3. **Expand Socioeconomic and Access-to-Care Variables:** While basic demographic variables were included, more detailed socioeconomic indicators (e.g. income, employment status or access to primary care) may help explain admission patterns, particularly avoidable or delayed admissions.
4. **Reconsider Outcome Definition and Class Structure:** The current admission type categories may be too broad. Future studies could explore alternative groupings (e.g. emergency vs non-emergency) or hierarchical classification approaches to better capture distinctions in admission decisions.

By implementing these recommendations, future reports and analysis could be more reliable, accurate and actionable.

References

- Kaggle. (n.d.). *Healthcare dataset* [Data set]. Kaggle. Retrieved December 17, 2025, from https://www.kaggle.com/datasets/prasad22/healthcare-dataset
- *Finding duplicate values in a data frame in R: A guide using base R and dplyr*. (2023, July 16). R-bloggers. https://www.r-bloggers.com/2023/07/finding-duplicate-values-in-a-data-frame-in-r-a-guide-using-base-r-and-dplyr/
- *Identify and remove duplicate data in R*. (2018, October 19). Datanovia. https://www.datanovia.com/en/lessons/identify-and-remove-duplicate-data-in-r/
- (n.d.). The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/e1071/e1071.pdf
- (n.d.). The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/tidyverse/tidyverse.pdf
- (n.d.). The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/dlookr/dlookr.pdf
- (n.d.). The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/dplyr/dplyr.pdf
- (n.d.). The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf
- (n.d.). The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/tidyr/tidyr.pdf
- *A Short Introduction to the caret Package*. (n.d.). https://cran.r-project.org/web/packages/caret/vignettes/caret.html
- Liaw, A. & Matthew Wiener. (2024). *Breiman and Cutlers random forests for classification and regression*. CRAN. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf
- *Package 'kernlab.'* (2025). https://cran.r-project.org/web/packages/kernlab/kernlab.pdf
- *Getting Started with Keras*. (n.d.). https://cran.r-project.org/web/packages/keras3/vignettes/getting_started.html
- Allaire, J., Kalinowski, T., Falbel, D., Eddelbuettel, D., Yuan Tang, Golding, N., & Google Inc. (2025). *Package 'tensorflow.'* CRAN. https://cran.r-project.org/web/packages/tensorflow/tensorflow.pdf
- Ripley, B., Venables, W., & R-project.org. (2025). *Functions for classification* [Functions for Classification]. https://cran.r-project.org/web/packages/class/class.pdf
- Silge, J., & Robinson, D. (2023, January 7). *Introduction to tidytext*. The Comprehensive R Archive Network. https://cran.r-project.org/web/packages/tidytext/vignettes/tidytext.html
- McQuire, P., & Kume, A. (2023). *R programming for actuarial science*. John Wiley & Sons.

Appendices

Appendix 1: Installing Packages and Loading Data into R

Appendix 1.1: Set Working Directory

Setting the working directory enables R where to retrieve the data file from, so that we can load the data set for analysis.

Set working directory of where the data file is located by: ‘Session’ –> ‘Set Working Directory’ –> ‘Choose Directory’ –> choose the file of where the data file is located OR use the code below by inserting the file’s location and removing the # to run the code.

```
# setwd("insert_file_location_here")
```

Appendix 1.2: Installing Necessary Packages

Before we start, we need to install the packages used in each section to use the functions we need for analysis. Later on in each section, we will load the packages used for the section.

Note: Some packages are used in different sections as well, but a package only needs to be installed once.

```
# Data Cleaning and Quality Checking
install.packages("tidyverse", dependencies = TRUE, repos = "http://cran.rs
tudio.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'tidyverse' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("dlookr", dependencies = TRUE, repos = "http://cran.rstud
io.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'dlookr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

# Exploratory Data Analysis (EDA)
install.packages("dplyr", dependencies = TRUE, repos = "http://cran.rstudi
o.com")
```

```
## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'dplyr' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'dplyr'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\dplyr\libs\x64\dplyr.dll to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\dplyr\libs\x64\dplyr
.dll:
## Permission denied

## Warning: restored 'dplyr'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages
```

```
install.packages("ggplot2", dependencies = TRUE, repos = "http://cran.rstu
dio.com")
```

```
## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'ggplot2' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages
```

```
install.packages("tidyr", dependencies = TRUE, repos = "http://cran.rstudi
o.com")
```

```
## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'tidyr' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'tidyr'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\tidyr\libs\x64\tidyr.dll to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\tidyr\libs\x64\tidyr
.dll:
## Permission denied

## Warning: restored 'tidyr'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages
```

```
install.packages("scales", dependencies = TRUE, repos = "http://cran.rstud
io.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'scales' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("janitor", dependencies = TRUE, repos = "http://cran.rstu
dio.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'janitor' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

# Variable Selection and Regression
install.packages("MASS", dependencies = TRUE, repos = "http://cran.rstudio
.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'MASS' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'MASS'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\MASS\libs\x64\MASS.dll to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\MASS\libs\x64\MASS.d
ll:
## Permission denied

## Warning: restored 'MASS'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("glmnet", dependencies = TRUE, repos = "http://cran.rstud
io.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'glmnet' successfully unpacked and MD5 sums checked
## Warning: cannot remove prior installation of package 'glmnet'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\glmnet\libs\x64\glmnet.dll
to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\glmnet\libs\x64\glmn
et.dll:
## Permission denied

## Warning: restored 'glmnet'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages
```

```
install.packages("Matrix", dependencies = TRUE, repos = "http://cran.rstud
io.com")
```

```
## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'Matrix' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'Matrix'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\Matrix\libs\x64\Matrix.dll
to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\Matrix\libs\x64\Matr
ix.dll:
## Permission denied

## Warning: restored 'Matrix'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages
```

```
install.packages("nnet", dependencies = TRUE, repos = "http://cran.rstudio
.com")
```

```
## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'nnet' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'nnet'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\nnet\libs\x64\nnet.dll to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\nnet\libs\x64\nnet.d
```

```
ll:
## Permission denied

## Warning: restored 'nnet'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("caret", dependencies = TRUE, repos = "http://cran.rstudi
o.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'caret' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'caret'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\caret\libs\x64\caret.dll to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\caret\libs\x64\caret
.dll:
## Permission denied

## Warning: restored 'caret'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("hnp", dependencies = TRUE, repos = "http://cran.rstudio.
com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## Warning: dependency 'glmmADMB' is not available

## package 'hnp' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

# Classification
install.packages("e1071", dependencies = TRUE, repos = "http://cran.rstudi
o.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'e1071' successfully unpacked and MD5 sums checked
## Warning: cannot remove prior installation of package 'e1071'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\e1071\libs\x64\e1071.dll to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\e1071\libs\x64\e1071
.dll:
## Permission denied

## Warning: restored 'e1071'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("ranfomForest", dependencies = TRUE, repos = "http://cran
.rstudio.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## Warning: package 'ranfomForest' is not available for this version of R
##
## A version of this package for your version of R might be available else
where,
## see the ideas at
## https://cran.r-project.org/doc/manuals/r-patched/R-admin.html#Installin
g-packages

install.packages("class", dependencies = TRUE, repos = "http://cran.rstudi
o.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'class' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'class'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\class\libs\x64\class.dll to
## C:\Users\Acer User\AppData\Local\R\win-library\4.5\class\libs\x64\class
.dll:
## Permission denied

## Warning: restored 'class'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("kernlab", dependencies = TRUE, repos = "http://cran.rstu
dio.com")
```

```
## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'kernlab' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'kernlab'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\00LOCK\kernlab\libs\x64\kernlab.dl
l to
## C:\Users\Acer
## User\AppData\Local\R\win-library\4.5\kernlab\libs\x64\kernlab.dll: Perm
ission
## denied

## Warning: restored 'kernlab'

##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

# Deep Learning
install.packages("keras3", dependencies = TRUE, repos = "http://cran.rstud
io.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'keras3' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages

install.packages("tensorflow", dependencies = TRUE, repos = "http://cran.r
studio.com")

## Installing package into 'C:/Users/Acer User/AppData/Local/R/win-library
/4.5'
## (as 'lib' is unspecified)

## package 'tensorflow' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Acer User\AppData\Local\Temp\RtmpCGvgyq\downloaded_packages
```

Appendix 1.3: Loading Data into R

Load the data file into RStudio using the code below and look at the beginning of the data to see some features of the data. (E.g.: what are the variable names in the file, the types of data)

```
# To Load the data into R for a csv file first remove #, then insert file
name:
```

```
# raw_data <- read.csv("insert_name_of_your_data.csv")
raw_data <- read.csv("healthcare_dataset.csv") # Name the loaded data as "
raw data" as the data has not been checked

head(raw_data) # To show beginning of data

##              Name Age Gender Blood.Type Medical.Condition Date.of.Admissi
on
## 1 Bobby JacksOn  30   Male         B-            Cancer         2024-01-
31
## 2  LesLie TErRy  62   Male         A+           Obesity         2019-08-
20
## 3   DaNnY sMitH  76 Female         A-           Obesity         2022-09-
22
## 4  andrEw waTtS  28 Female         O+          Diabetes         2020-11-
18
## 5 adrIENNE bEll  43 Female        AB+            Cancer         2022-09-
19
## 6 EMILY JOHNSOn  36   Male         A+            Asthma         2023-12-
20
##               Doctor                    Hospital Insurance.Provider Billin
g.Amount
## 1    Matthew Smith             Sons and Miller         Blue Cross
18856.28
## 2  Samantha Davies                     Kim Inc           Medicare
33643.33
## 3 Tiffany Mitchell                    Cook PLC              Aetna
27955.10
## 4      Kevin Wells Hernandez Rogers and Vang,           Medicare
37909.78
## 5   Kathleen Hanna                 White-White              Aetna
14238.32
## 6     Taylor Newton             Nunez-Humphrey   UnitedHealthcare
48145.11
##   Room.Number Admission.Type Discharge.Date  Medication Test.Results
## 1         328         Urgent     2024-02-02 Paracetamol       Normal
## 2         265      Emergency     2019-08-26   Ibuprofen Inconclusive
## 3         205      Emergency     2022-10-07     Aspirin       Normal
## 4         450       Elective     2020-12-18   Ibuprofen     Abnormal
## 5         458         Urgent     2022-10-09  Penicillin     Abnormal
## 6         389         Urgent     2023-12-24   Ibuprofen       Normal
```

Appendix 2: Data Cleaning and Quality Checking

Appendix 2.1: Loading Necessary Packages

```
library(tidyverse) # For data manipulation, visualization and analysis

## ── Attaching core tidyverse packages ──────────────────────── tidyverse
 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.1     ✓ stringr   1.5.2
## ✓ ggplot2   4.0.1     ✓ tibble    3.3.0
## ✓ lubridate 1.9.4     ✓ tidyr     1.3.1
## ✓ purrr     1.1.0
## ── Conflicts ────────────────────────────────────────── tidyverse_confl
icts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force
all conflicts to become errors
```

```
library(dlookr)    # This allows us to create a table with the number of m
issing values and gives us other insights
```

```
## Registered S3 methods overwritten by 'dlookr':
##   method          from
##   plot.transform  scales
##   print.transform scales
##
## Attaching package: 'dlookr'
##
## The following object is masked from 'package:tidyr':
##
##     extract
##
## The following object is masked from 'package:base':
##
##     transform
```

Appendix 2.2: Quick Data Diagnosis

In this section, we will do a preliminary check on the data to see what might need to be done in data cleaning and quality checking.

```
str(raw_data) # Checking the structure of data frame
```

```
## 'data.frame':    55500 obs. of  15 variables:
##  $ Name              : chr  "Bobby JacksOn" "LesLie TErRy" "DaNnY sMitH
" "andrEw waTtS" ...
##  $ Age               : int  30 62 76 28 43 36 21 20 82 58 ...
##  $ Gender            : chr  "Male" "Male" "Female" "Female" ...
##  $ Blood.Type        : chr  "B-" "A+" "A-" "O+" ...
##  $ Medical.Condition : chr  "Cancer" "Obesity" "Obesity" "Diabetes" ...
##  $ Date.of.Admission : chr  "2024-01-31" "2019-08-20" "2022-09-22" "202
0-11-18" ...
##  $ Doctor            : chr  "Matthew Smith" "Samantha Davies" "Tiffany
Mitchell" "Kevin Wells" ...
##  $ Hospital          : chr  "Sons and Miller" "Kim Inc" "Cook PLC" "Her
nandez Rogers and Vang," ...
##  $ Insurance.Provider: chr  "Blue Cross" "Medicare" "Aetna" "Medicare"
...
##  $ Billing.Amount    : num  18856 33643 27955 37910 14238 ...
##  $ Room.Number       : int  328 265 205 450 458 389 389 277 316 249 ...
##  $ Admission.Type    : chr  "Urgent" "Emergency" "Emergency" "Elective"
 ...
##  $ Discharge.Date    : chr  "2024-02-02" "2019-08-26" "2022-10-07" "202
0-12-18" ...
##  $ Medication        : chr  "Paracetamol" "Ibuprofen" "Aspirin" "Ibupro
```

```
fen" ...
##  $ Test.Results      : chr  "Normal" "Inconclusive" "Normal" "Abnormal"
 ...

dim(raw_data) # Checking the dimensions of the data

## [1] 55500    15

diagnosis <- diagnose(raw_data) # This creates a table of variable names,
the type of data, number of missing values and unique values, along with t
he rate of unique values
diagnosis # This allows us to see the whole diagnosis.

## # A tibble: 15 × 6
##    variables        types missing_count missing_percent unique_count un
ique_rate
##    <chr>            <chr>         <int>           <dbl>        <int>
    <dbl>
##  1 Name             char…             0               0        49992
0.901
##  2 Age              inte…             0               0           77
0.00139
##  3 Gender           char…             0               0            2
0.0000360
##  4 Blood.Type       char…             0               0            8
0.000144
##  5 Medical.Conditi… char…             0               0            6
0.000108
##  6 Date.of.Admissi… char…             0               0         1827
0.0329
##  7 Doctor           char…             0               0        40341
0.727
##  8 Hospital         char…             0               0        39876
0.718
##  9 Insurance.Provi… char…             0               0            5
0.0000901
## 10 Billing.Amount   nume…             0               0        50000
0.901
## 11 Room.Number      inte…             0               0          400
0.00721
## 12 Admission.Type   char…             0               0            3
0.0000541
## 13 Discharge.Date   char…             0               0         1856
0.0334
## 14 Medication       char…             0               0            5
0.0000901
## 15 Test.Results     char…             0               0            3
0.0000541
```

Appendix 2.3: Check For Missing Values

There may be certain values in the data that are not recognised as missing values, so we will first replace all these possible values. Then, we can see the true missing data count to determine if the data needs to be cleaned for missing values.

```
raw_data[raw_data == "NA"] <- NA
# If there are any values that says NA but recognises it as text, it might
 cause the missing count to be 0, so this fixes this issue
raw_data[raw_data == "[]"] <- NA
# If there are any values that says [] but recognises it as text, it might
 cause the missing count to be 0, so this fixes this issue
raw_data[raw_data == ""] <- NA
# If there are any values that says empty cells but recognises it as text,
 it might cause the missing count to be 0, so this fixes this issue

# Now we can see the true missing data count in diagnose again
diagnosis <- diagnose(raw_data)
diagnosis

## # A tibble: 15 × 6
##    variables          types missing_count missing_percent unique_count un
ique_rate
##    <chr>              <chr>         <int>           <dbl>        <int>
    <dbl>
##  1 Name               char…             0               0        49992
0.901
##  2 Age                inte…             0               0           77
0.00139
##  3 Gender             char…             0               0            2
0.0000360
##  4 Blood.Type         char…             0               0            8
0.000144
##  5 Medical.Conditi… char…             0               0            6
0.000108
##  6 Date.of.Admissi… char…             0               0         1827
0.0329
##  7 Doctor             char…             0               0        40341
0.727
##  8 Hospital           char…             0               0        39876
0.718
##  9 Insurance.Provi… char…             0               0            5
0.0000901
## 10 Billing.Amount     nume…             0               0        50000
0.901
## 11 Room.Number        inte…             0               0          400
0.00721
## 12 Admission.Type     char…             0               0            3
0.0000541
## 13 Discharge.Date     char…             0               0         1856
0.0334
## 14 Medication         char…             0               0            5
0.0000901
## 15 Test.Results       char…             0               0            3
0.0000541

# We can see that there are no missing observations in any of the data. To
 confirm this, we can use another line of code as below:
sum(is.na(raw_data)) # Shows total number of missing observations, which i
s 0 and confirms our diagnosis above
```

```
## [1] 0
```

Appendix 2.4: Correcting Letters

We can see from the beginning of data that the patient names do not have consistent letters. So, to make data cleaning more accurate, we will make all the letters in the name into small letters.

```
# Changing patient names to all small letters
raw_data$Name <- tolower(raw_data$Name)
```

Appendix 2.5: Check For Duplicates

Check if all data in the dataset is unique and there are no repeats.

If all data taken is unique, then the patients' names should be all unique and match the number of rows. So, we will check for the names of all the patients and ensure there are no repeats.

```
# Step 1: Check if all patient names are unique
# If all patient names are unique, then the number of unique rows should m
atch with the number of rows of the data
if(n_distinct(raw_data$Name) == nrow(raw_data)) {
  print(paste("No duplicates in patient names"))
} else {
  num_duplicates <- nrow(raw_data)-n_distinct(raw_data$Name)
  print(paste("Number of potential duplicates in patient names:", num_dupl
icates))
}

## [1] "Number of potential duplicates in patient names: 15265"

# Output shows that there is a duplicate patient names, so we are going to
 identify it using the duplicated function with a condition to show the fi
rst duplicate line. Then, we match the admission data and discharge date o
f the patients with the same name to identify the difference between these
 duplicated names.

duplicated_rows1 <- raw_data %>%
  filter(
    duplicated(across(c(Name))) |
      duplicated(across(c(Name,)), fromLast = TRUE)
  ) %>%
  arrange(Name)

print(head(duplicated_rows1))

##              Name Age Gender Blood.Type Medical.Condition Date.of.Admiss
ion
## 1   aaron archer  47 Female         B-            Cancer        2021-01
-10
## 2   aaron archer  49 Female         B-            Cancer        2021-01
-10
## 3    aaron baker  73   Male         B+            Cancer        2019-06
-18
```

```
## 4    aaron baker  84 Female         A+          Asthma       2022-06
-04
## 5 aaron bradshaw  25 Female         O+       Arthritis       2019-11
-30
## 6 aaron bradshaw  78   Male         B-       Arthritis       2019-11
-15
##               Doctor                   Hospital Insurance.Provider
## 1 Cynthia Villanueva    Montes Case and Mendez,           Medicare
## 2 Cynthia Villanueva    Montes Case and Mendez,           Medicare
## 3       Tracy Torres      Wise and Todd, Parker           Medicare
## 4     Brittany Smith Carter, Abbott and Fuentes           Medicare
## 5 Mackenzie Phillips                  Olson LLC              Aetna
## 6       Sheila Smith             Group Martinez   UnitedHealthcare
##   Billing.Amount Room.Number Admission.Type Discharge.Date  Medication
## 1      10602.077         108         Urgent     2021-01-17 Paracetamol
## 2      10602.077         108         Urgent     2021-01-17 Paracetamol
## 3      10135.885         234       Elective     2019-07-10     Aspirin
## 4       6826.677         496      Emergency     2022-06-10     Lipitor
## 5      16342.364         255      Emergency     2019-12-20     Lipitor
## 6      50132.996         393      Emergency     2019-11-17  Penicillin
##   Test.Results
## 1 Inconclusive
## 2 Inconclusive
## 3 Inconclusive
## 4       Normal
## 5 Inconclusive
## 6     Abnormal
```

```
# Here, we can see that most of the duplicated names, but it is hard to te
ll which to remove
# So, we will filter it again to look for the patients that have the exact
 same details for all their columns. This is to enable us to know which of
 them are definitely duplicates that need to be removed
exact_duplicates <- raw_data %>%
  group_by(across(everything())) %>%
  filter(n() > 1) %>%
  ungroup() %>%
  arrange(Name)

print(head(exact_duplicates))
```

```
## # A tibble: 6 × 15
##   Name          Age Gender Blood.Type Medical.Condition Date.of.Admissi
on Doctor
##   <chr>       <int> <chr>  <chr>      <chr>             <chr>
   <chr>
## 1 abigail yo…    41 Female O+         Hypertension      2022-12-15
   Edwar…
## 2 abigail yo…    41 Female O+         Hypertension      2022-12-15
   Edwar…
## 3 adam thomas    75 Male   O+         Hypertension      2022-01-02
   Bever…
## 4 adam thomas    75 Male   O+         Hypertension      2022-01-02
   Bever…
```

```
## 5 alex black     51 Male   O+         Diabetes          2022-03-27
   Frank…
## 6 alex black     51 Male   O+         Diabetes          2022-03-27
   Frank…
## # ℹ 8 more variables: Hospital <chr>, Insurance.Provider <chr>,
## #   Billing.Amount <dbl>, Room.Number <int>, Admission.Type <chr>,
## #   Discharge.Date <chr>, Medication <chr>, Test.Results <chr>

# Here, we can see that there are 1068/2=534 pairs of duplicated names wit
h the eaxct same details, so we will remove one of each

raw_data <- raw_data %>%
  distinct()
# To check if this was successful, we will run the exact_duplicates again.
 If it prints to have no rows, we will know that this is successful

exact_duplicates <- raw_data %>%
  group_by(across(everything())) %>%
  filter(n() > 1) %>%
  ungroup() %>%
  arrange(Name)

print(head(exact_duplicates))

## # A tibble: 0 × 15
## # ℹ 15 variables: Name <chr>, Age <int>, Gender <chr>, Blood.Type <chr>
,
## #   Medical.Condition <chr>, Date.of.Admission <chr>, Doctor <chr>,
## #   Hospital <chr>, Insurance.Provider <chr>, Billing.Amount <dbl>,
## #   Room.Number <int>, Admission.Type <chr>, Discharge.Date <chr>,
## #   Medication <chr>, Test.Results <chr>

# There are no rows printed, so the removal was successful

# Now, we will check if the patients with the same name have any different
 details on each column
other_cols <- setdiff(names(raw_data), "Name") # Get all columns except Na
me

name_duplicates <- raw_data %>%
  group_by(Name) %>%
  # Check if any other column varies within the same Name
  filter(if_any(all_of(other_cols), ~ n_distinct(.) > 1)) %>%
  ungroup() %>%
  arrange(Name)

print(head(name_duplicates))

## # A tibble: 6 × 15
##   Name           Age Gender Blood.Type Medical.Condition Date.of.Admissi
on Doctor
##   <chr>        <int> <chr>  <chr>      <chr>             <chr>
   <chr>
## 1 aaron arch…    47 Female B-         Cancer            2021-01-10
   Cynth…
```

```
## 2 aaron arch…    49 Female B-         Cancer             2021-01-10
   Cynth…
## 3 aaron baker    73 Male   B+         Cancer             2019-06-18
   Tracy…
## 4 aaron baker    84 Female A+         Asthma             2022-06-04
   Britt…
## 5 aaron brad…    25 Female O+         Arthritis          2019-11-30
   Macke…
## 6 aaron brad…    78 Male   B-         Arthritis          2019-11-15
   Sheil…
## # ℹ 8 more variables: Hospital <chr>, Insurance.Provider <chr>,
## #   Billing.Amount <dbl>, Room.Number <int>, Admission.Type <chr>,
## #   Discharge.Date <chr>, Medication <chr>, Test.Results <chr>

# From what we can observe, most of the patients with the same name have a
lmost the same details except for their age. But since we cannot tell what
 is the real age of the patient, we will assume the first entry of the per
son in the data is correct as there are no missing data for each column.
raw_data <- raw_data %>%
  distinct(Name, .keep_all = TRUE)

# Now, we check again if there are any duplicated names to see if the remo
val was successful
if(n_distinct(raw_data$Name) == nrow(raw_data)) {
  print(paste("No duplicates in patient names"))
} else {
  num_duplicates <- nrow(raw_data)-n_distinct(raw_data$Name)
  print(paste("Number of potential duplicates in patient names:", num_dupl
icates))
}

## [1] "No duplicates in patient names"

# There are no duplicates in patient names, so the removal was successful
```

Appendix 2.6: Checking For Negative Values

We can see that all the numeric and integer columns should not have any negative scores, so we need to make sure that is true. The numeric and integer columns are age, billing amount and room number.

```
# The function below selects the chosen numeric and integer values that we
 can observe from the diagnose table above and gives us a summary of how m
any negative values are in each of the chosen columns

# Choose the numeric and integer columns
num_cols <- c("Age", "Billing.Amount", "Room.Number")

# Filter to see if there are any negative values
negative_values <- raw_data %>%
  filter(if_any(all_of(num_cols), ~ . < 0))

# Print to see all of them
print(negative_values)
```

```
##                        Name Age Gender Blood.Type Medical.Condition
## 1           ashley erickson  32 Female        AB-            Cancer
## 2         christopher weiss  49 Female        AB-            Asthma
## 3             ashley warner  60   Male         A+      Hypertension
## 4              jay galloway  74 Female         O+            Asthma
## 5         joshua williamson  72 Female         B-          Diabetes
## 6             scott vazquez  74   Male         B+          Diabetes
## 7            carol anderson  39 Female         B-      Hypertension
## 8  mr. christopher alvarado  77   Male        AB+           Obesity
## 9            alexandra khan  32   Male        AB+         Arthritis
## 10               joseph cox  23   Male        AB-          Diabetes
## 11        mitchell maldonado  41 Female         A+            Asthma
## 12          julie christian  28 Female        AB+          Diabetes
## 13        alexander gardner  45   Male        AB-            Cancer
## 14             angela allen  83 Female         B-          Diabetes
## 15               alan scott  48 Female         A-            Asthma
## 16             brandon hall  27   Male         B+      Hypertension
## 17            lucas sullivan  31 Female         A-           Obesity
## 18            rebecca cline  39 Female         B+            Asthma
## 19            valerie allen  82 Female         B-          Diabetes
## 20              brian young  85   Male         O-            Cancer
## 21            jennifer cruz  19 Female         A+      Hypertension
## 22              elaine tran  32 Female         O-         Arthritis
## 23            ashley ramsey  53   Male        AB+            Cancer
## 24             john ferrell  58 Female         O-      Hypertension
## 25    christopher hamilton  43   Male         O-            Asthma
## 26        michael castaneda  85 Female         A+          Diabetes
## 27               jason owen  57 Female         B+      Hypertension
## 28          nicole hurst dvm  43 Female        AB+           Obesity
## 29         gabrielle decker  69   Male        AB-         Arthritis
## 30              susan ellis  28   Male         O+      Hypertension
## 31           heather bryant  50 Female         A+            Asthma
## 32         ricardo reynolds  49 Female         O-           Obesity
## 33           whitney cooper  65   Male        AB-           Obesity
## 34               mark stone  33   Male         A+          Diabetes
## 35          frederick moore  83   Male        AB-           Obesity
## 36             stephen chan  71   Male         A+            Cancer
## 37      elizabeth thompson  54 Female         A-            Cancer
## 38             terry wilson  39   Male         O-          Diabetes
## 39            patrick perry  44   Male         B-      Hypertension
## 40             juan osborne  20 Female        AB+          Diabetes
## 41           brenda parrish  59   Male         B+          Diabetes
## 42             melissa diaz  82   Male         A-      Hypertension
## 43   mrs. michelle clark dvm  31 Female         A-            Asthma
## 44     mrs. andrea davis phd  25 Female         A+      Hypertension
## 45             daniel drake  67 Female         B+      Hypertension
## 46         alexander martin  46   Male         A+      Hypertension
## 47            walter french  78 Female         A-            Asthma
## 48            craig salazar  19 Female         O+          Diabetes
## 49              molly tapia  78 Female        AB-      Hypertension
## 50           karen williams  31 Female        AB-            Asthma
## 51            calvin campos  26   Male        AB-            Cancer
## 52            donna proctor  32 Female         O+           Obesity
## 53           joseph stevens  75 Female         O+           Obesity
## 54            susan ellison  60 Female          B-           Asthma
## 55             timothy ford  47   Male         AB-         Diabetes
## 56          madeline thomas  32 Female          O-        Arthritis
## 57               bryan rios  85 Female          A-           Cancer
## 58          johnathan smith  18 Female          A-        Arthritis
## 59          veronica kelley  20   Male          O-           Cancer
## 60                john hahn  80   Male          O-           Asthma
## 61            amanda morgan  35 Female         AB+     Hypertension
## 62        dr. michael mckay  67   Male          O+           Cancer
## 63           betty moore md  84 Female          A+          Obesity
## 64         jillian shepherd  84 Female          O-         Diabetes
## 65              susan solis  40 Female          A+           Cancer
## 66           joseph robbins  71   Male          O-     Hypertension
## 67              erica woods  57   Male          B-           Cancer
## 68             daniel david  32   Male          O+          Obesity
## 69              ryan potter  53   Male          A+         Diabetes
## 70             thomas pratt  57 Female          A+          Obesity
## 71               james byrd  79   Male          O+         Diabetes
## 72               james luna  64 Female         AB-           Cancer
## 73              wanda chase  60   Male          O-     Hypertension
## 74            krista conley  21 Female          O+         Diabetes
## 75              emma savage  57   Male         AB+     Hypertension
## 76             heather rios  81 Female          A-           Asthma
## 77            aaron flowers  42   Male          A+           Asthma
##    Date.of.Admission              Doctor                          Hospi
tal
## 1         2019-11-05       Gerald Hooper         and Johnson Moore, Bra
nch
## 2         2023-02-16      Kelly Thompson                      Hunter-Hug
hes
## 3         2021-12-21      Andrea Bentley             and Wagner, Lee Kl
ein
## 4         2021-01-20       Debra Everett                       Group Pet
ers
## 5         2021-03-21         Wendy Ramos           and Huff Reeves, Den
nis
## 6         2023-04-12        Edward Yates                          James 
Ltd
## 7         2020-04-03   Dr. Patrick Hines      Carter Carter, and Patter
son
## 8         2022-06-03 Mr. Dean Guzman DDS                        Johnson 
Inc
## 9         2022-07-14      Michael Vaughn            Bowen Lopez, and Te
rry
## 10        2019-10-13         Peter Smith                           Inc W
ard
## 11        2022-06-22       Melissa Woods                Richardson-Benn
ett
## 12        2019-06-30         Kathryn Ray                    Group Mcconn
ell
## 13        2023-03-23         Tracy Brown                    Maldonado Gr
oup
## 14        2019-08-21      Robert Kennedy          Brown and Randolph, Yo
ung
```

```
## 15         2019-10-23          Jacob Kirby                                Lopez-Hug
hes
## 16         2020-01-29          Cindy Smith                              Sons and Ev
ans
## 17         2022-04-05         Brenda Burke                             Sons and Har
ris
## 18         2020-05-07       Rebecca Miller   Stout Dougherty, and Washing
ton
## 19         2021-02-05          Judy Massey                      Hebert, Brown Kim 
and
## 20         2021-04-13  Dr. Jenna Caldwell                                  Hampton 
LLC
## 21         2021-06-09       Samantha Lloyd                               Ruiz and S
ons
## 22         2022-09-15      Alicia Thompson                           Ramirez-Thomp
son
## 23         2019-05-30          James Smith                             Barker-Mitch
ell
## 24         2019-05-20       Randy Calderon                                 Inc Spen
cer
## 25         2023-01-09        Pamela Obrien                                 Brock Gr
oup
## 26         2023-02-14          Amber Ochoa      Huber, Rodriguez Chapman 
and
## 27         2020-01-27        Matthew Clark                              Crawford Gr
oup
## 28         2021-04-19         Leah Russell                                Warner-Wa
tts
## 29         2022-11-07       William Krause                                 Diaz-Bis
hop
## 30         2019-12-05    Rebecca Valentine                               Anderson-B
all
## 31         2021-02-26         Morgan Scott                                 Scott Gr
oup
## 32         2019-11-20          Deborah Cox        Wyatt Murphy, and Gonza
les
## 33         2019-08-20         Amber Walker                              and Sons Bu
rns
## 34         2022-10-10     Stephanie Garcia                                  Cole-Pa
rks
## 35         2019-10-15         Mary Johnson                                Miller Gr
oup
## 36         2024-03-05        Allen Mcgrath                                 Jones-Sm
ith
## 37         2023-12-11    Jessica Mills DDS Fitzpatrick, Nielsen and Mcdon
ald
## 38         2024-02-19          Diana Smith       Medina and Elliott Stewa
rt,
## 39         2022-03-26  Miss Sandra Brooks                             Foster-Cleme
nts
## 40         2021-04-28            Joe Lawson                             Garcia-Erick
son
## 41         2024-04-14         Sherry Brown      and Dennis, Mcguire Johns
ton
```

```
## 42          2019-06-06        Stephen Levy                          Hill-Sali
nas
## 43          2019-08-23      Denise Gilbert                           Bruce Gr
oup
## 44          2024-03-26      Brian Alvarado          Poole, Poole Mendoza 
and
## 45          2020-04-24           Brett Ray                              Carr 
Ltd
## 46          2021-09-24         Tracy Smith           Carter, and Nguyen S
now
## 47          2023-08-05  Jeanette Rodriguez                           Lyons-Ca
sey
## 48          2022-01-18      Heidi Williams                          Group Mil
ler
## 49          2024-04-15     Steven Clements     and Rosales, Macdonald Han
son
## 50          2021-10-07        Danny French                          Smith-Can
non
## 51          2023-12-15           Eric Mccoy                      Clements-Bow
man
## 52          2023-11-28     Ronald Mcdonald                             Jones 
PLC
## 53          2023-01-30         John Watson                   Schneider and S
ons
## 54          2022-08-28         Vincent Cox                          Johnson-T
odd
## 55          2020-08-08        Steven Kirby                       Anderson-Mo
ore
## 56          2022-02-08         Carmen Mann                             Moore 
PLC
## 57          2022-04-10    Timothy Marshall                            Miller 
PLC
## 58          2020-11-24         Sara Watson                          Williams 
Ltd
## 59          2021-07-12      Amanda Ramirez                          Roberts-K
ing
## 60          2024-02-24          Mark Hines                            Ltd Wil
son
## 61          2020-09-27    Gregory Figueroa                   Petersen-Hernan
dez
## 62          2019-05-31        Dawn Navarro            Mcconnell and Rios, Cl
ark
## 63          2022-10-13       Crystal Baker                           Aguilar 
Inc
## 64          2019-12-28          George King                           Jones Gr
oup
## 65          2020-04-26      Frances Rosales                          Ltd Willi
ams
## 66          2024-02-10     Stacey Davenport                           Vaughn 
PLC
## 67          2022-10-08    Nathan Gutierrez                         LLC Carpen
ter
## 68          2022-05-25     Elizabeth Smith            Estes and Garza, Carr
oll
```

```
## 69         2022-02-04      Stephen Jensen                   Thomas Sons 
and
## 70         2021-03-27        Eduardo Hall          and Wade, Huffman Arn
old
## 71         2023-12-25          Adam Pitts                        Ayers-Sm
ith
## 72         2022-09-22         Todd Becker                       Juarez-Cl
ark
## 73         2021-03-26        Justin Gibbs          Jimenez and Ross Mas
on,
## 74         2021-01-03  Elizabeth Anderson                         Ltd Mcn
eil
## 75         2021-09-05       Maria Johnson                       Armstrong 
LLC
## 76         2021-08-19      Morgan Jackson        Jones, Edwards Jacobs 
and
## 77         2020-06-03       Ronald Patton          Brown, Santos and How
ard
##    Insurance.Provider Billing.Amount Room.Number Admission.Type Dischar
ge.Date
## 1               Aetna     -502.50781         376         Urgent     201
9-11-23
## 2               Aetna    -1018.24537         204       Elective     202
3-03-09
## 3               Aetna     -306.36493         426       Elective     202
2-01-11
## 4          Blue Cross     -109.09712         381      Emergency     202
1-02-09
## 5          Blue Cross     -576.72791         369         Urgent     202
1-04-17
## 6            Medicare     -135.98600         445       Elective     202
3-05-03
## 7          Blue Cross     -370.98367         203       Elective     202
0-04-10
## 8          Blue Cross    -1310.27289         257       Elective     202
2-06-13
## 9               Aetna     -692.40882         372       Elective     202
2-07-19
## 10         Blue Cross     -353.86519         271       Elective     201
9-10-25
## 11           Medicare     -378.96078         414         Urgent     202
2-06-30
## 12              Cigna     -367.20396         387         Urgent     201
9-07-09
## 13              Cigna     -198.28381         329       Elective     202
3-04-09
## 14              Cigna      -43.09852         189         Urgent     201
9-08-24
## 15           Medicare     -857.12593         309         Urgent     201
9-11-18
## 16              Aetna     -155.08202         368         Urgent     202
0-02-01
## 17   UnitedHealthcare     -211.72385         496      Emergency     202
2-04-11
```

```
## 18            Cigna     -656.15307          195       Emergency      202
0-05-17
## 19         Medicare     -227.99506          118          Urgent      202
1-02-25
## 20            Cigna     -147.07220          133       Emergency      202
1-04-20
## 21       Blue Cross     -124.75643          207          Urgent      202
1-06-28
## 22  UnitedHealthcare     -75.81945          482        Elective      202
2-10-12
## 23       Blue Cross     -135.71907          235        Elective      201
9-06-09
## 24         Medicare     -308.58427          394       Emergency      201
9-05-27
## 25         Medicare     -214.60542          398        Elective      202
3-01-25
## 26         Medicare     -224.63242          179       Emergency      202
3-03-09
## 27       Blue Cross     -100.60361          354        Elective      202
0-02-16
## 28  UnitedHealthcare    -211.28374          213          Urgent      202
1-04-23
## 29            Cigna     -676.85250          354          Urgent      202
2-11-30
## 30  UnitedHealthcare    -130.86753          451        Elective      202
0-01-02
## 31  UnitedHealthcare    -233.93073          289       Emergency      202
1-03-07
## 32            Aetna     -124.64904          222       Emergency      201
9-12-20
## 33  UnitedHealthcare    -577.72911          357       Emergency      201
9-09-02
## 34            Cigna      -53.83209          457          Urgent      202
2-10-23
## 35            Cigna     -532.76112          132          Urgent      201
9-11-12
## 36            Cigna     -599.26536          119          Urgent      202
4-03-12
## 37            Aetna     -887.02422          402        Elective      202
3-12-13
## 38            Aetna    -1316.61858          491       Emergency      202
4-03-20
## 39  UnitedHealthcare     -75.62019          340          Urgent      202
2-03-29
## 40       Blue Cross     -317.63292          324       Emergency      202
1-05-20
## 41  UnitedHealthcare     -90.07890          221       Emergency      202
4-04-22
## 42            Aetna      -23.86673          453       Emergency      201
9-07-06
## 43            Cigna     -952.83119          136          Urgent      201
9-09-14
## 44            Aetna     -492.17839          372       Emergency      202
4-04-22
```

```
## 45              Aetna     -591.91742          426        Elective      202
0-04-26
## 46   UnitedHealthcare      -38.96615          401       Emergency      202
1-09-26
## 47   UnitedHealthcare     -136.16469          347       Emergency      202
3-08-30
## 48   UnitedHealthcare      -37.26775          409        Elective      202
2-01-26
## 49   UnitedHealthcare     -230.57928          211        Elective      202
4-05-05
## 50              Cigna     -407.80082          494       Emergency      202
1-10-15
## 51           Medicare    -1277.64534          339        Elective      202
4-01-13
## 52              Aetna     -786.62472          435        Elective      202
3-12-28
## 53              Cigna     -967.59471          476        Elective      202
3-02-13
## 54           Medicare     -416.91482          365          Urgent      202
2-09-08
## 55   UnitedHealthcare      -97.85312          169        Elective      202
0-08-23
## 56              Aetna     -964.79862          122          Urgent      202
2-02-25
## 57              Cigna     -311.75563          471          Urgent      202
2-04-11
## 58              Cigna      -85.47571          349          Urgent      202
0-12-03
## 59           Medicare     -808.46506          221       Emergency      202
1-07-28
## 60              Cigna    -1520.42055          403        Elective      202
4-03-14
## 61           Medicare     -652.18137          144        Elective      202
0-10-20
## 62   UnitedHealthcare     -199.66379          122          Urgent      201
9-06-12
## 63         Blue Cross     -529.96301          409          Urgent      202
2-10-26
## 64   UnitedHealthcare     -279.84241          142       Emergency      202
0-01-07
## 65              Cigna     -226.38112          132       Emergency      202
0-04-29
## 66   UnitedHealthcare    -1428.84394          205          Urgent      202
4-02-29
## 67              Cigna    -1049.01234          146          Urgent      202
2-10-14
## 68              Aetna      -36.21727          167       Emergency      202
2-06-22
## 69           Medicare     -614.94559          223          Urgent      202
2-03-01
## 70           Medicare     -228.54685          496       Emergency      202
1-04-22
## 71              Cigna     -483.70889          446          Urgent      202
3-12-29
```

```
## 72            Aetna    -2008.49214         162         Urgent     202
2-10-20
## 73         Medicare     -820.16335         357      Emergency     202
1-04-12
## 74            Aetna     -289.80162         340         Urgent     202
1-01-14
## 75       Blue Cross    -1660.00937         193       Elective     202
1-10-02
## 76       Blue Cross     -378.74681         249         Urgent     202
1-08-23
## 77         Medicare     -378.69641         115       Elective     202
0-06-17
##     Medication Test.Results
## 1   Penicillin       Normal
## 2   Penicillin Inconclusive
## 3    Ibuprofen       Normal
## 4    Ibuprofen     Abnormal
## 5      Aspirin     Abnormal
## 6    Ibuprofen     Abnormal
## 7    Ibuprofen     Abnormal
## 8  Paracetamol Inconclusive
## 9      Lipitor     Abnormal
## 10     Lipitor Inconclusive
## 11     Lipitor Inconclusive
## 12     Lipitor       Normal
## 13     Aspirin     Abnormal
## 14  Penicillin     Abnormal
## 15 Paracetamol     Abnormal
## 16     Aspirin     Abnormal
## 17     Aspirin     Abnormal
## 18   Ibuprofen     Abnormal
## 19     Aspirin Inconclusive
## 20     Aspirin     Abnormal
## 21     Aspirin       Normal
## 22     Lipitor     Abnormal
## 23  Penicillin Inconclusive
## 24 Paracetamol Inconclusive
## 25   Ibuprofen Inconclusive
## 26  Penicillin Inconclusive
## 27 Paracetamol Inconclusive
## 28 Paracetamol Inconclusive
## 29  Penicillin       Normal
## 30     Lipitor Inconclusive
## 31  Penicillin Inconclusive
## 32     Lipitor       Normal
## 33     Aspirin       Normal
## 34     Aspirin Inconclusive
## 35     Aspirin       Normal
## 36  Penicillin     Abnormal
## 37   Ibuprofen       Normal
## 38 Paracetamol Inconclusive
## 39     Aspirin Inconclusive
## 40   Ibuprofen Inconclusive
## 41  Penicillin     Abnormal
## 42   Ibuprofen     Abnormal
## 43  Penicillin       Normal
## 44     Aspirin Inconclusive
## 45     Lipitor     Abnormal
## 46     Lipitor Inconclusive
## 47     Aspirin     Abnormal
## 48     Aspirin     Abnormal
## 49  Penicillin Inconclusive
## 50     Aspirin       Normal
## 51   Ibuprofen       Normal
## 52 Paracetamol     Abnormal
## 53   Ibuprofen       Normal
## 54     Aspirin     Abnormal
## 55 Paracetamol Inconclusive
## 56 Paracetamol       Normal
## 57 Paracetamol     Abnormal
## 58 Paracetamol       Normal
## 59 Paracetamol Inconclusive
## 60     Lipitor     Abnormal
## 61  Penicillin     Abnormal
## 62   Ibuprofen     Abnormal
## 63     Aspirin Inconclusive
## 64  Penicillin Inconclusive
## 65     Lipitor Inconclusive
## 66   Ibuprofen     Abnormal
## 67     Lipitor       Normal
## 68  Penicillin Inconclusive
## 69  Penicillin     Abnormal
## 70 Paracetamol Inconclusive
## 71  Penicillin     Abnormal
## 72   Ibuprofen     Abnormal
## 73     Lipitor Inconclusive
## 74     Lipitor Inconclusive
## 75  Penicillin       Normal
## 76     Lipitor     Abnormal
## 77 Paracetamol Inconclusive

# The output shows that there are negative values, so there we will remove
 them

raw_data <- raw_data %>%
  filter(if_all(all_of(num_cols), ~ . >= 0))

# Run negative values again to check if removal was successful
negative_values <- raw_data %>%
  filter(if_any(all_of(num_cols), ~ . < 0))

print(negative_values)

##  [1] Name               Age                Gender             Blood.Typ
e
##  [5] Medical.Condition  Date.of.Admission  Doctor             Hospital

##  [9] Insurance.Provider Billing.Amount     Room.Number        Admission
```

```
.Type
## [13] Discharge.Date     Medication         Test.Results
## <0 rows> (or 0-length row.names)

# Since there are no rows printed, the removal was successful and there ar
e no more negative values where there should not be.
```

Appendix 2.7: Check The Type Of The Data

```
diagnosis <- diagnose(raw_data)
diagnosis

## # A tibble: 15 × 6
##    variables        types missing_count missing_percent unique_count un
ique_rate
##    <chr>            <chr>         <int>           <dbl>        <int>
    <dbl>
##  1 Name             char…             0               0        40158
1
##  2 Age              inte…             0               0           68
0.00169
##  3 Gender           char…             0               0            2
0.0000498
##  4 Blood.Type       char…             0               0            8
0.000199
##  5 Medical.Conditi… char…             0               0            6
0.000149
##  6 Date.of.Admissi… char…             0               0         1827
0.0455
##  7 Doctor           char…             0               0        33502
0.834
##  8 Hospital         char…             0               0        32734
0.815
##  9 Insurance.Provi… char…             0               0            5
0.000125
## 10 Billing.Amount   nume…             0               0        40158
1
## 11 Room.Number      inte…             0               0          400
0.00996
## 12 Admission.Type   char…             0               0            3
0.0000747
## 13 Discharge.Date   char…             0               0         1856
0.0462
## 14 Medication       char…             0               0            5
0.000125
## 15 Test.Results     char…             0               0            3
0.0000747

# From here, we can tell that there are types of data that need to be chan
ged, so we assign the correct types to each variables

raw_data$Gender <- as.factor(raw_data$Gender)
raw_data$Blood.Type <- as.factor(raw_data$Blood.Type)
raw_data$Medical.Condition <- as.factor(raw_data$Medical.Condition)
raw_data$Date.of.Admission <- as.Date(raw_data$Date.of.Admission)
```

```
raw_data$Doctor <- as.factor(raw_data$Doctor)
raw_data$Hospital <- as.factor(raw_data$Hospital)
raw_data$Insurance.Provider <- as.factor(raw_data$Insurance.Provider)
raw_data$Admission.Type <- as.factor(raw_data$Admission.Type)
raw_data$Discharge.Date <- as.Date(raw_data$Discharge.Date)
raw_data$Medication <- as.factor(raw_data$Medication)
raw_data$Test.Results <- as.factor(raw_data$Test.Results)

diagnosis <- diagnose(raw_data)
diagnosis

## # A tibble: 15 × 6
##    variables        types missing_count missing_percent unique_count un
ique_rate
##    <chr>            <chr>         <int>           <dbl>        <int>
    <dbl>
##  1 Name             char…             0               0        40158
1
##  2 Age              inte…             0               0           68
0.00169
##  3 Gender           fact…             0               0            2
0.0000498
##  4 Blood.Type       fact…             0               0            8
0.000199
##  5 Medical.Conditi… fact…             0               0            6
0.000149
##  6 Date.of.Admissi… Date              0               0         1827
0.0455
##  7 Doctor           fact…             0               0        33502
0.834
##  8 Hospital         fact…             0               0        32734
0.815
##  9 Insurance.Provi… fact…             0               0            5
0.000125
## 10 Billing.Amount   nume…             0               0        40158
1
## 11 Room.Number      inte…             0               0          400
0.00996
## 12 Admission.Type   fact…             0               0            3
0.0000747
## 13 Discharge.Date   Date              0               0         1856
0.0462
## 14 Medication       fact…             0               0            5
0.000125
## 15 Test.Results     fact…             0               0            3
0.0000747
```

Appendix 2.8: Removing Unneccessary Columns

There are some columns that were only used to clean the data, but are not relevant to the analysis, so we will remove them.

Here, we will remove the “Name”, “Doctor”, “Date of Admission”, “Hospital” and “Discharge Date” columns as they were only used to identify duplicates, but will not affect analysis at all.

```
# Removing "Name", "Doctor", "Date of Admission", "Hospital" and "Discharg
e Date" columns
raw_data <- raw_data[,-c(1,6:8,13)]

# Run diagnosis to see if "Name" column has been removed
diagnosis <- diagnose(raw_data)
diagnosis

## # A tibble: 10 × 6
##    variables          types missing_count missing_percent unique_count un
ique_rate
##    <chr>              <chr>         <int>           <dbl>        <int>
    <dbl>
##  1 Age                inte…             0               0           68
0.00169
##  2 Gender             fact…             0               0            2
0.0000498
##  3 Blood.Type         fact…             0               0            8
0.000199
##  4 Medical.Conditi… fact…             0               0            6
0.000149
##  5 Insurance.Provi… fact…             0               0            5
0.000125
##  6 Billing.Amount     nume…             0               0        40158
1
##  7 Room.Number        inte…             0               0          400
0.00996
##  8 Admission.Type     fact…             0               0            3
0.0000747
##  9 Medication         fact…             0               0            5
0.000125
## 10 Test.Results       fact…             0               0            3
0.0000747
```

Appendix 2.9: Renaming Checked And Cleaned Data Set

Hence, we can rename “raw_data” as “health”

```
health <- raw_data # Rename the "raw_data" to "health"

# Use diagnosis() function to check whether renaming is successful. If ren
aming is successful, it will show the same diagnosis table from above
diagnosis <- diagnose(health)
diagnosis

## # A tibble: 10 × 6
##    variables          types missing_count missing_percent unique_count un
ique_rate
##    <chr>              <chr>         <int>           <dbl>        <int>
    <dbl>
##  1 Age                inte…             0               0           68
```

```
0.00169
##  2 Gender          fact…          0          0          2
0.0000498
##  3 Blood.Type      fact…          0          0          8
0.000199
##  4 Medical.Conditi… fact…         0          0          6
0.000149
##  5 Insurance.Provi… fact…         0          0          5
0.000125
##  6 Billing.Amount  nume…          0          0      40158
1
##  7 Room.Number     inte…          0          0        400
0.00996
##  8 Admission.Type  fact…          0          0          3
0.0000747
##  9 Medication      fact…          0          0          5
0.000125
## 10 Test.Results    fact…          0          0          3
0.0000747
```

Appendix 3: Exploratory Data Analysis (EDA)

Appendix 3.1: Loading Necassary Packages

```
library(dplyr)   # For data manipulation
library(tidyr)   # To tidy data
library(ggplot2) # Load package ggplot2 for function ggplot to make plots
library(scales)  # This package allows us to transform any scientific nota
tion on axes to be displayed as normal numbers

##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor

library(janitor) # To get the proportion of each discrete variable

##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

Appendix 3.2: Basic Statistics and Key Points Of Data

Using the summary function, we can better understand the key points and basic statistics in each variable before going into further analysis.

Appendix 3.2.1: Basic Summary

```
summary(health) # Shows the type and general key characteristics of each v
ariable, though it is only more useful for the continuous distributions as
 it can show the key points in the continuous data

##       Age           Gender          Blood.Type      Medical.Condition
##  Min.   :18.0   Female:20084   AB+     :5093   Arthritis   :6816
##  1st Qu.:35.0   Male  :20074   B+      :5059   Asthma      :6603
##  Median :52.0                  B-      :5020   Cancer      :6675
##  Mean   :51.7                  A-      :5014   Diabetes    :6779
##  3rd Qu.:69.0                  A+      :5013   Hypertension:6661
##  Max.   :85.0                  O-      :5006   Obesity     :6624
##                                (Other):9953
##          Insurance.Provider Billing.Amount       Room.Number
##  Aetna           :7929     Min.   :    9.239   Min.   :101.0
##  Blue Cross      :8066     1st Qu.:13292.171   1st Qu.:202.0
##  Cigna           :8104     Median :25595.331   Median :302.0
##  Medicare        :8103     Mean   :25609.534   Mean   :300.9
##  UnitedHealthcare:7956     3rd Qu.:37862.835   3rd Qu.:401.0
##                            Max.   :52764.277   Max.   :500.0
##
##    Admission.Type        Medication         Test.Results
##  Elective :13538   Aspirin    :8038   Abnormal    :13457
##  Emergency:13099   Ibuprofen  :8016   Inconclusive:13278
##  Urgent   :13521   Lipitor    :8141   Normal      :13423
##                    Paracetamol:7963
##                    Penicillin :8000
##
##
```

Analysis: - Age: Has a range from 18 to 85 with an average of 51.7 years old - Billing amount: Has a range of 9.24 to 52764.28 with similar mean and meadian - Room number: Has a range from 101 to 500

Appendix 3.2.2: Discrete Data

We use the tabyl() function to create frequency tables (counts and proportions) of each variable in a data frame. This is for use to get an idea of how the discrete data is structured.

```
# First, we look for which column is a factor column. This will tell us wh
ich variables are discrete
sapply(health, is.factor)

##                Age             Gender         Blood.Type  Medical.Condi
tion
##              FALSE               TRUE               TRUE
TRUE
## Insurance.Provider     Billing.Amount        Room.Number     Admission.
Type
##               TRUE              FALSE              FALSE
TRUE
##         Medication       Test.Results
##               TRUE               TRUE
```

```
# Then, we can use the tabyl() function to create the frequency tables
tabyl(health$Gender)

##  health$Gender     n   percent
##         Female 20084 0.5001245
##           Male 20074 0.4998755

tabyl(health$Blood.Type)

##  health$Blood.Type    n   percent
##                 A- 5014 0.1248568
##                 A+ 5013 0.1248319
##                AB- 4989 0.1242343
##                AB+ 5093 0.1268240
##                 B- 5020 0.1250062
##                 B+ 5059 0.1259774
##                 O- 5006 0.1246576
##                 O+ 4964 0.1236117

tabyl(health$Medical.Condition)

##  health$Medical.Condition    n   percent
##                 Arthritis 6816 0.1697296
##                    Asthma 6603 0.1644255
##                    Cancer 6675 0.1662184
##                  Diabetes 6779 0.1688082
##              Hypertension 6661 0.1658698
##                   Obesity 6624 0.1649485

tabyl(health$Insurance.Provider)

##  health$Insurance.Provider    n   percent
##                      Aetna 7929 0.1974451
##                 Blue Cross 8066 0.2008566
##                      Cigna 8104 0.2018029
##                   Medicare 8103 0.2017780
##           UnitedHealthcare 7956 0.1981174

tabyl(health$Admission.Type)

##  health$Admission.Type     n   percent
##               Elective 13538 0.3371184
##              Emergency 13099 0.3261866
##                 Urgent 13521 0.3366951

tabyl(health$Medication)

##  health$Medication    n   percent
##            Aspirin 8038 0.2001594
##          Ibuprofen 8016 0.1996115
##            Lipitor 8141 0.2027242
##        Paracetamol 7963 0.1982917
##         Penicillin 8000 0.1992131

tabyl(health$Test.Results)
```

```
##  health$Test.Results     n   percent
##             Abnormal 13457 0.3351013
##         Inconclusive 13278 0.3306440
##               Normal 13423 0.3342547
```

Analysis: Overall, looks to be quite evenly spread out for all variables. This can be confirmed later on with pie charts and box plots

Appendix 3.2.3: Correlation

Further, we check for correlation among numeric predictors.

```
cor(health[, sapply(health, is.numeric)], use = "complete.obs")
```

```
##                          Age Billing.Amount  Room.Number
## Age            1.0000000000   0.0002776157  0.002706352
## Billing.Amount 0.0002776157   1.0000000000 -0.002252990
## Room.Number    0.0027063515  -0.0022529897  1.000000000
```

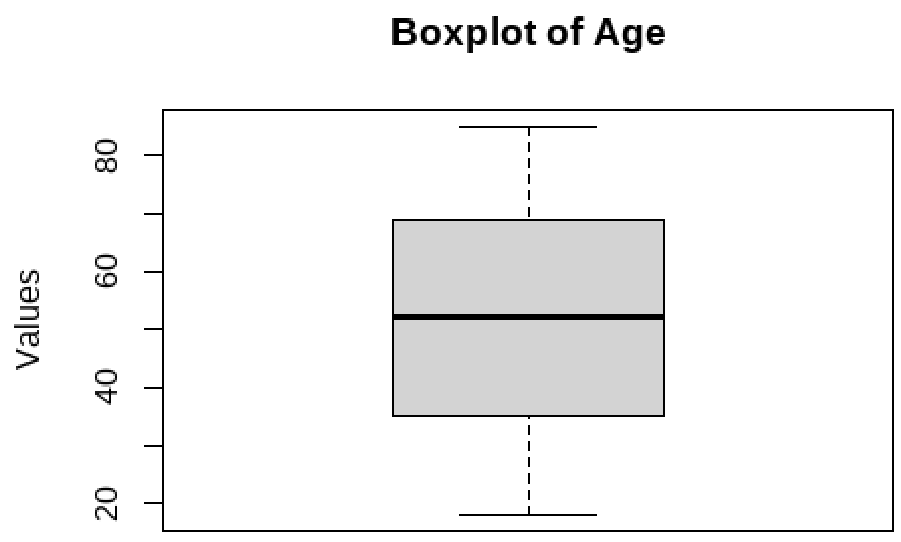
From what we can see there isn’t any significant correlation between any of the numeric variable as none of the correlation coefficients other than the diagonal (which is the correlation the variable itself) is more than 0.5.

Hence, when doing the variable selection in later sections, we will not use any interactions between the variables.

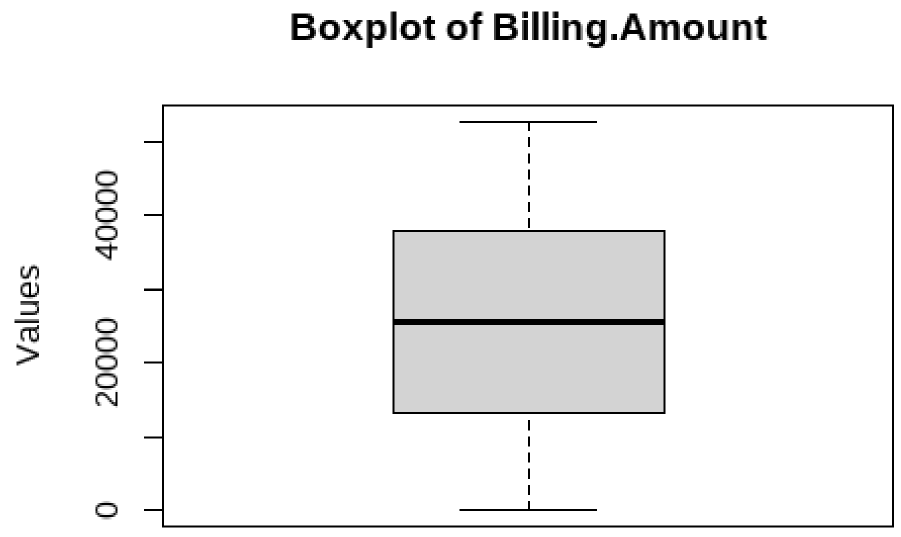
Appendix 3.3: EDA - Boxplots For Outliers

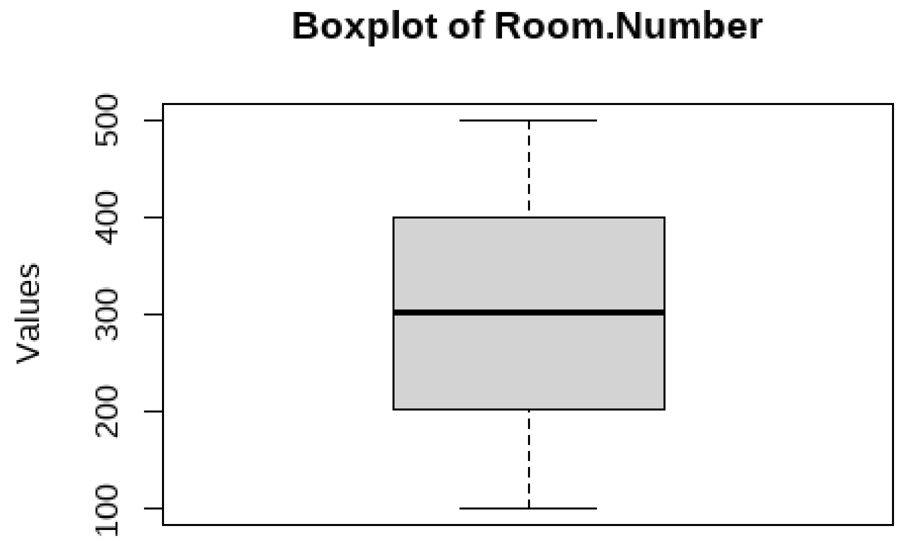
By plotting boxplots for the continuously distributed variables, we can see if there are any outliers present in the variable. Then, we will carry out further analysis to determine whether the outliers need to be removed from the variable to prevent it from affecting the further analysis and predictions.

```
# Loop to make boxplot for each variable
for(var in colnames(health)) {
  # Choose only numeric variables to look at the outliers as an overview
  if(is.numeric(health[[var]])) {
    # If column is numeric, then create boxplot
    var_box <- boxplot(health[[var]], main=paste("Boxplot of" , var), ylab
="Values") + theme_minimal()
    var_box # Show the boxplot created
  } else {
    # If column is not numeric, then says its not applicable
    print(paste("Boxplot is not applicable for", var))
  }
}
```



```
## [1] "Boxplot is not applicable for Gender"
## [1] "Boxplot is not applicable for Blood.Type"
## [1] "Boxplot is not applicable for Medical.Condition"
## [1] "Boxplot is not applicable for Insurance.Provider"
```





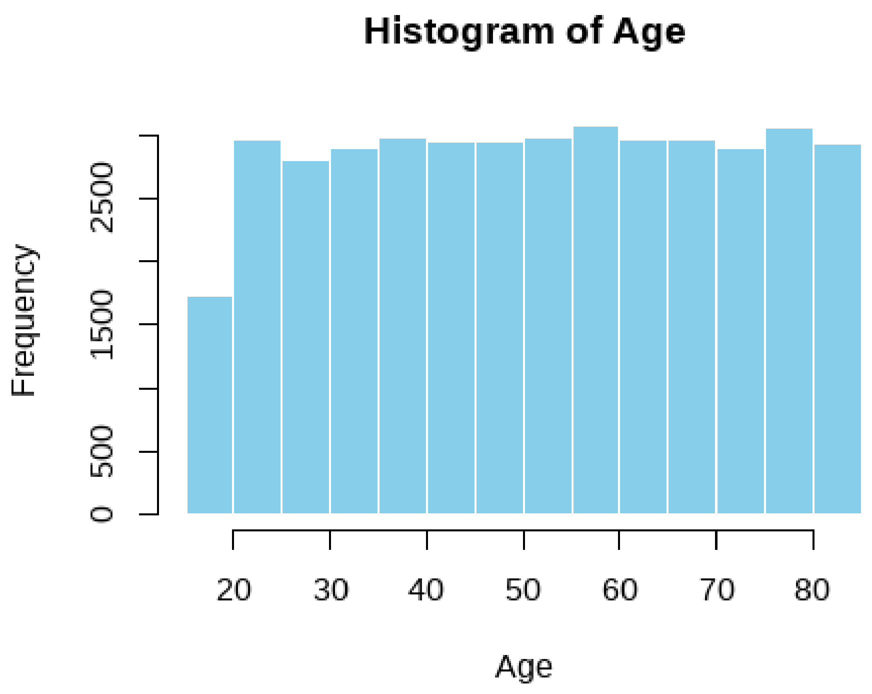
```
## [1] "Boxplot is not applicable for Admission.Type"
## [1] "Boxplot is not applicable for Medication"
## [1] "Boxplot is not applicable for Test.Results"
```

Analysis: Results show that there are no outliers. Hence, there is no need to check whether any outliers need to be removed.

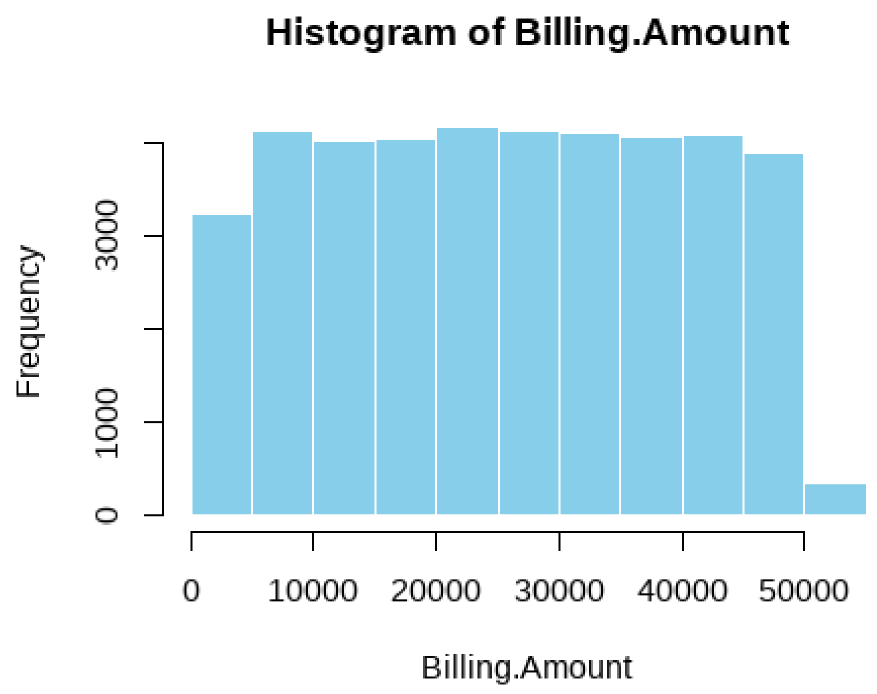
Appendix 3.4: EDA - Histogram

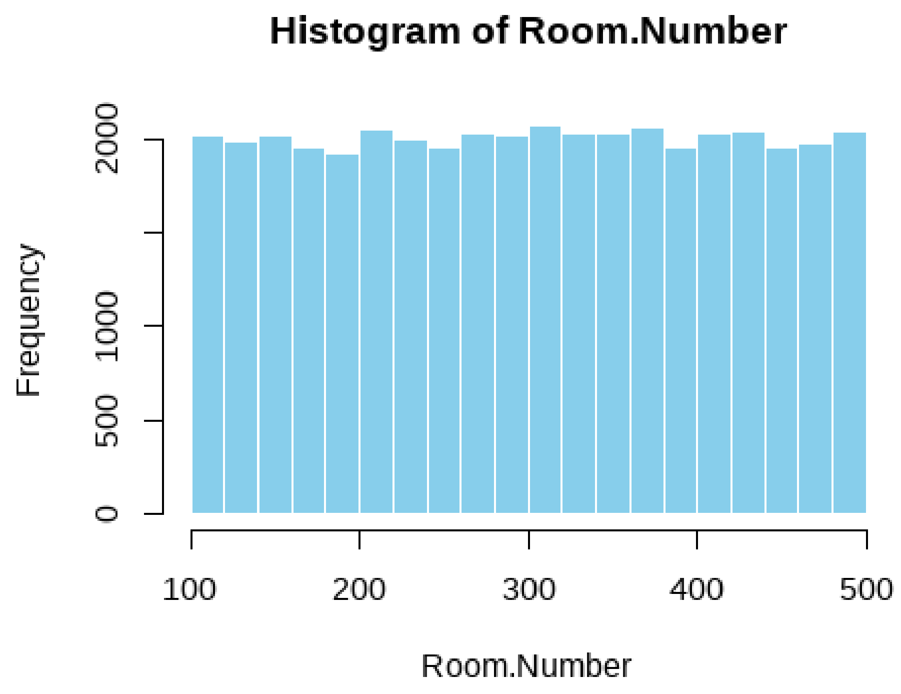
We plot histograms for continuous numeric variables to get an idea of how the variable is distributed.

```
for (var in colnames(health)){
  # Check if the variable is numeric
  if (is.numeric(health[[var]])){
  # Create histogram for each numeric variable
  hist(health[[var]], main = paste("Histogram of", var), xlab = var, col =
 "skyblue", border = "white")
} else {
    # If column is not numeric, then says its not applicable
    print(paste("Histogram is not applicable for", var))
  }
}
```



```
## [1] "Histogram is not applicable for Gender"
## [1] "Histogram is not applicable for Blood.Type"
## [1] "Histogram is not applicable for Medical.Condition"
## [1] "Histogram is not applicable for Insurance.Provider"
```





```
## [1] "Histogram is not applicable for Admission.Type"
## [1] "Histogram is not applicable for Medication"
## [1] "Histogram is not applicable for Test.Results"
```

Analysis: - Age: Looks to be negatively skewed - Billing amount: Looks to be positively skewed - Room number: Has the form of a uniform distribution

Appendix 3.5: EDA - Pie Charts For All Variables

We use pie charts to see the percentage of each category in the variable to better understand how the data of each variable is proportioned.

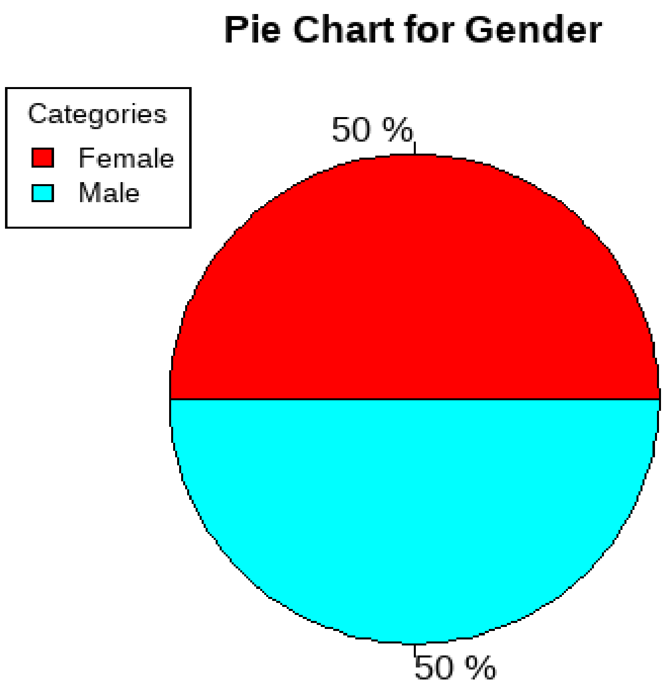
Appendix 3.5.1: Creating Function For Making Pie Charts

Here, we are going to create a unique function to quickly make pie charts. Then, we will loop our unique functions in Section 3.5.2 to each of the variables so that it will create the pie charts easily.

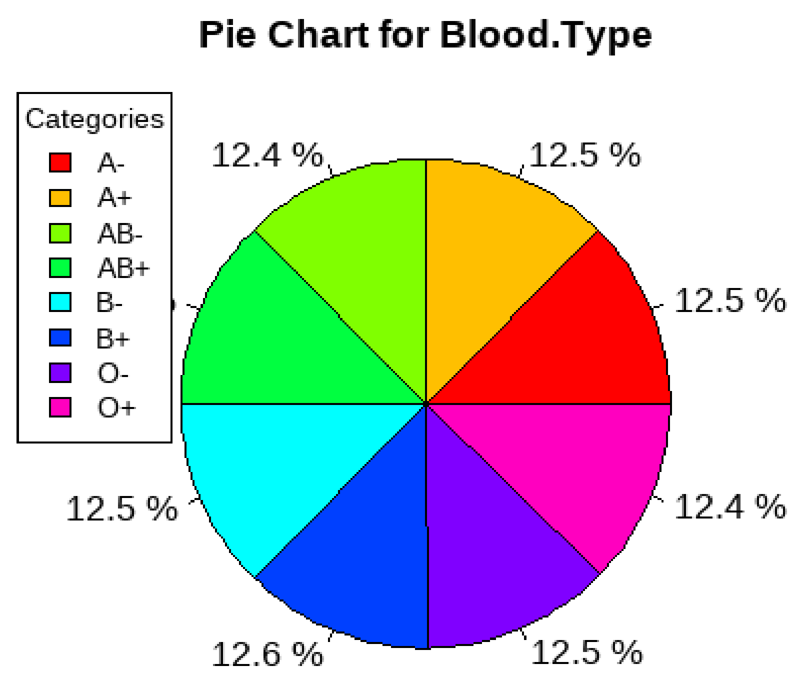
```r
pie_chart <- function(data, var) {
  # Select the variable form the dataset
  selected_var <- data[[var]]

  # If NA exists, convert it to be an "Unknown" so that it can be classed
as a category as well
  selected_var[is.na(selected_var)] <- "Unknown"

  # Count the number/frequency
  count <- table(selected_var)

  # Calculate percentages to 1 decimal place
  percentages <- round(100*count/sum(count), 1)

  # Adjust the plot margins for the pie chart
  par(mar=c(1,3,3,1))

  # Create the pie chart
  draw_chart <- pie(count,
                   col=rainbow(length(count)), # Auto-assign colours to ea
ch of the proportions
                   main=paste("Pie Chart for", var), # Name of pie chart
                   labels=paste(percentages, "%"), # Show it as percentage
                   radius=0.85,
                   cex=1.1, # Adjust the size of the chart
                   xpd=TRUE) # Allows legend to be drawn outside the plot
region

  # Add legend to see what colour means
  legend("topleft",
         legend=names(count),
         fill=rainbow(length(count)),
         title="Categories",
         cex=0.9, # Adjust the size of the text
         xpd=TRUE)
}
```

Appendix 3.5.2: Making Pie Charts

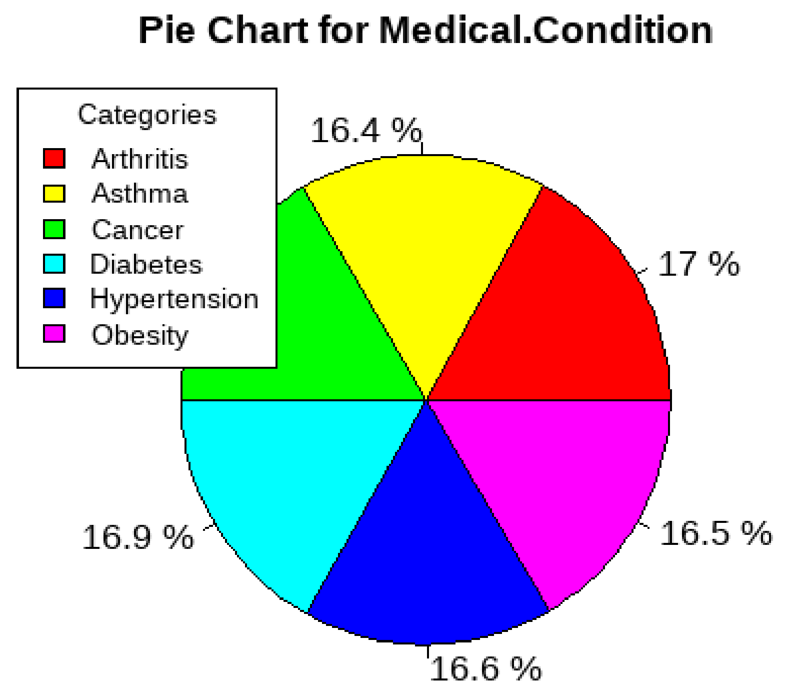
```r
# Here, we ignore "Doctor" and "Hospital" column as ther have too many fac
tor levels
for (var in colnames(health)) {
    if (is.factor(health[[var]])) {
        pie_chart(health, var)
    } else {
    # If column is not a factor, then says its not applicable
    print(paste("Pie chart is not applicable for", var))
    }
}

## [1] "Pie chart is not applicable for Age"

## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



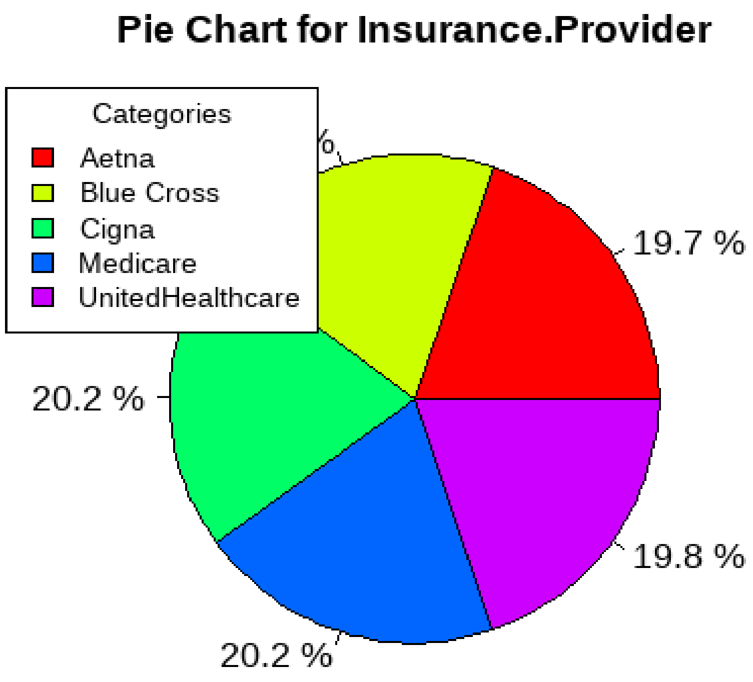
```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



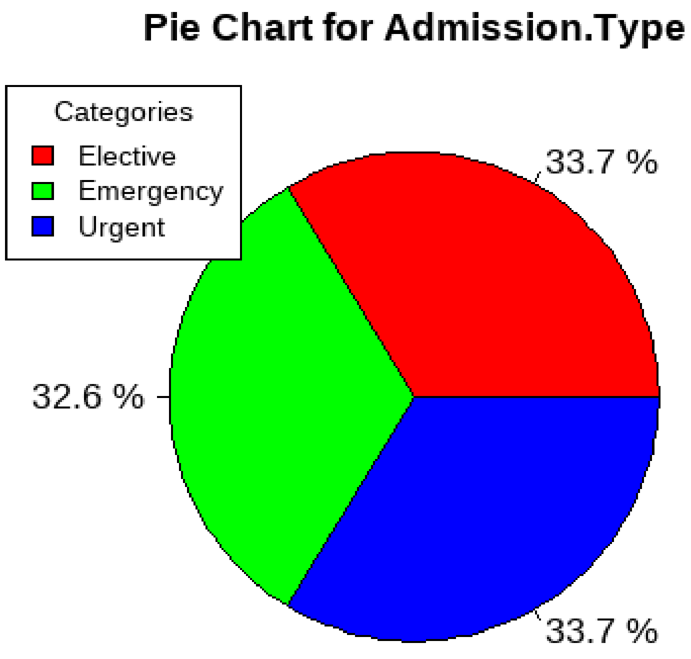
```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



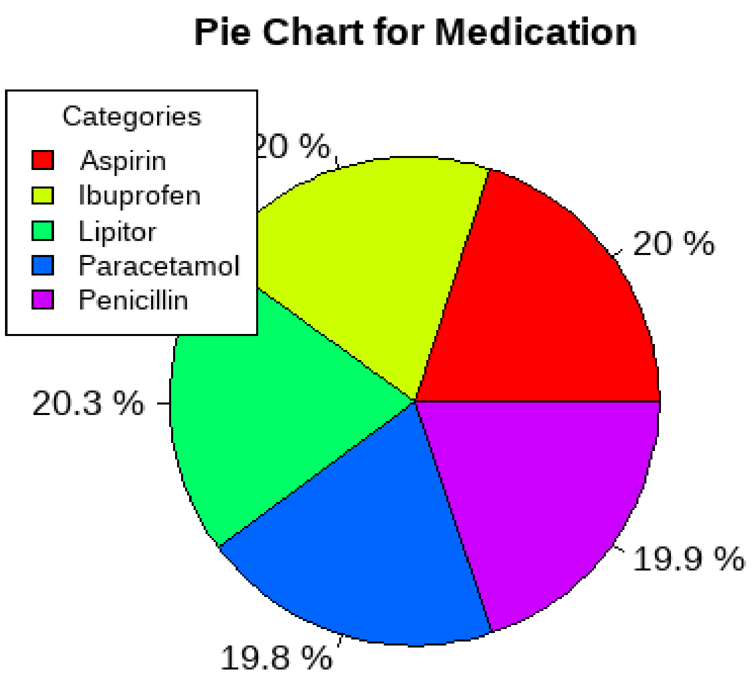
```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



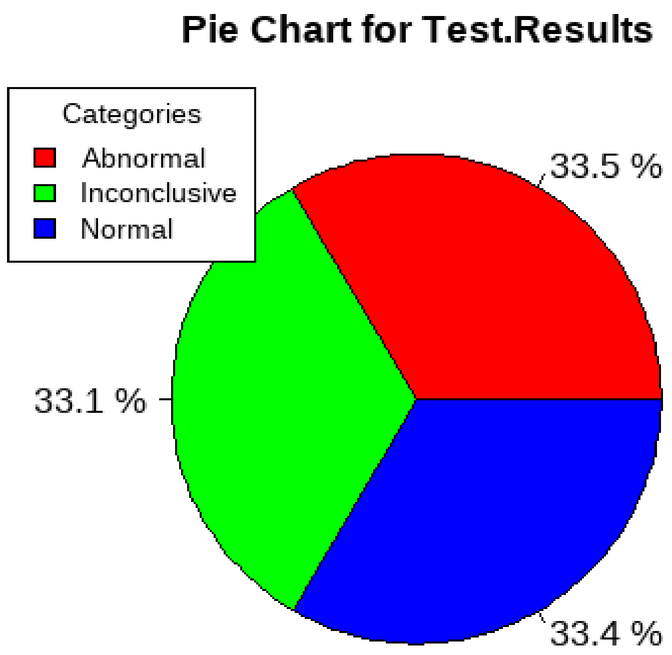
```
## [1] "Pie chart is not applicable for Billing.Amount"
## [1] "Pie chart is not applicable for Room.Number"

## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



Analysis: - Gender: Split evenly between male and female - Blood type, medical condition, insurance provider, admission type, medication, test results: Almost split evenly in its own variables

Appendix 3.6: EDA - Bar Charts For All Variables

We use bar charts to see the count of each category in the variable to better understand how the data of each variable is spread.

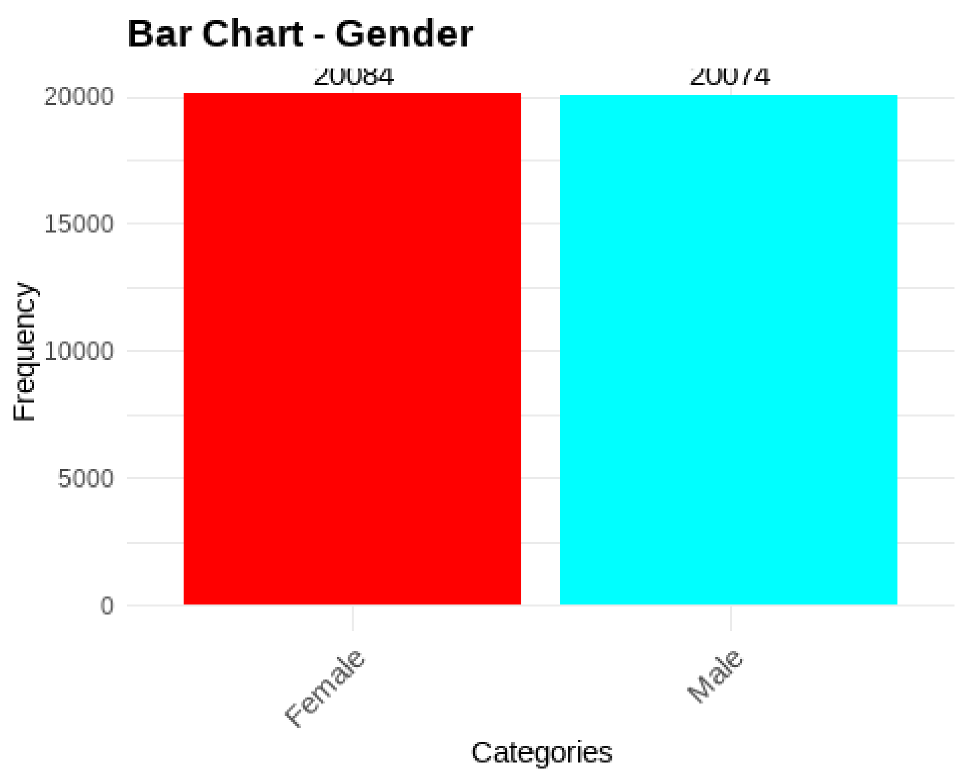
Appendix 3.6.1: Creating Function For Making Bar Chart

Here, we are going to create a unique function to quickly make the bar charts. Then, we will loop our unique functions in Section 3.6.2 to each of the variables so that it will create the bar charts easily.

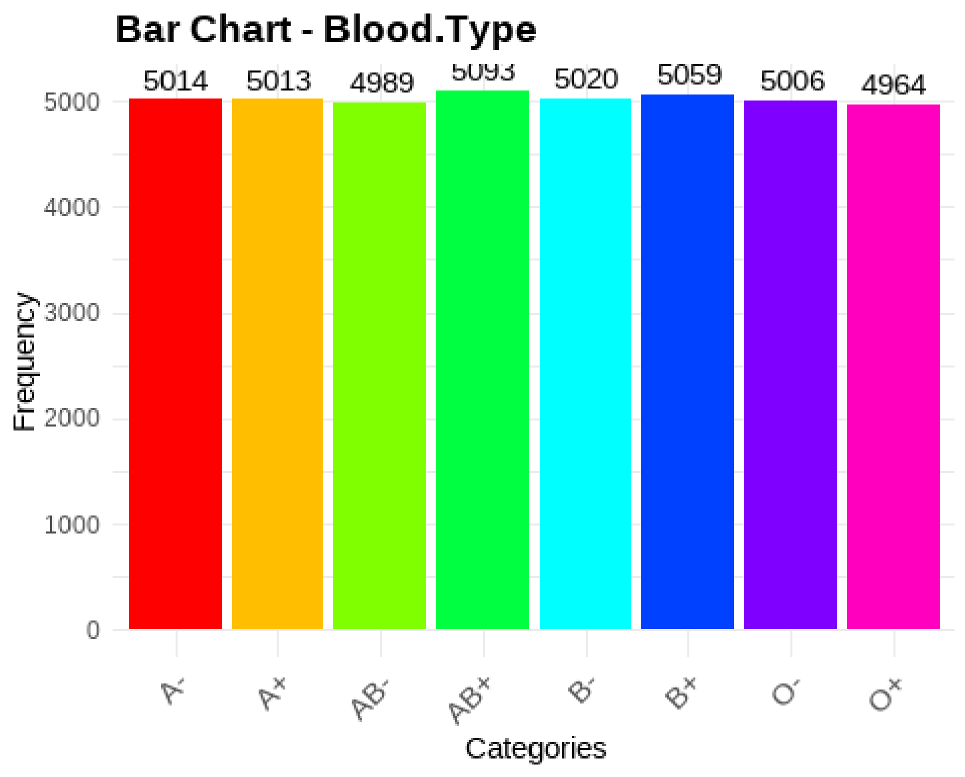
```r
bar_chart <- function(data, var) {
  # Select the variable form the dataset
  selected_var <- data[[var]]

  # If NA exists, convert it to be an "Unknown" so that it can be classed
as a category as well
  selected_var[is.na(selected_var)] <- "Unknown"

  # Count the number/frequency
  count <- as.data.frame(table(selected_var))

  # Rename the columns for ggplot2 compatibility
  colnames(count) <- c("category", "count")

  # Create bar chart
  bar <- ggplot(count, aes(x = category, y = count, fill = category)) +
    geom_bar(stat = "identity") +  # Use actual counts
    geom_text(aes(label = count), vjust = -0.5, size = 4) +  # Add count l
abels above bars
    labs(title = paste("Bar Chart -", var), x = "Categories", y = "Frequen
cy") +  # Labels
    scale_fill_manual(values = rainbow(length(count$category))) +  # Manua
l color palette
    theme_minimal() +  # Clean theme
    theme(axis.text.x=element_text(angle = 45, hjust = 1)) + # Rotate x-ax
is labels by 45 degrees
    theme(plot.title=element_text(size=14, face="bold"),  # Adjust plot ti
tle size,
          axis.text.x = element_text(size = 11),    # Adjust x-axis label
size,
          legend.position="none")

  # Print the bar chart
  print(bar)
}
```

Appendix 3.6.2: Making Bar Charts

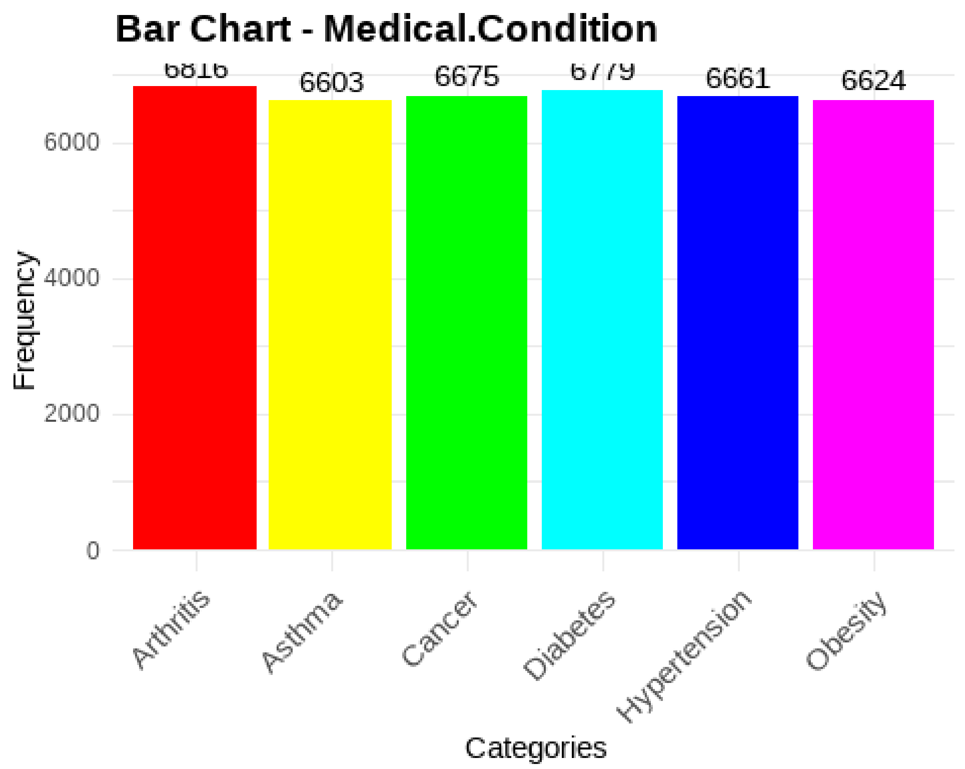
```
# Here, we there have too many factor levels
for (var in colnames(health)) {
    if (is.factor(health[[var]])) {
        bar_chart(health, var)
    } else {
    # If column is not a factor, then says its not applicable
    print(paste("Bar chart is not applicable for", var))
    }
}

## [1] "Bar chart is not applicable for Age"

## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



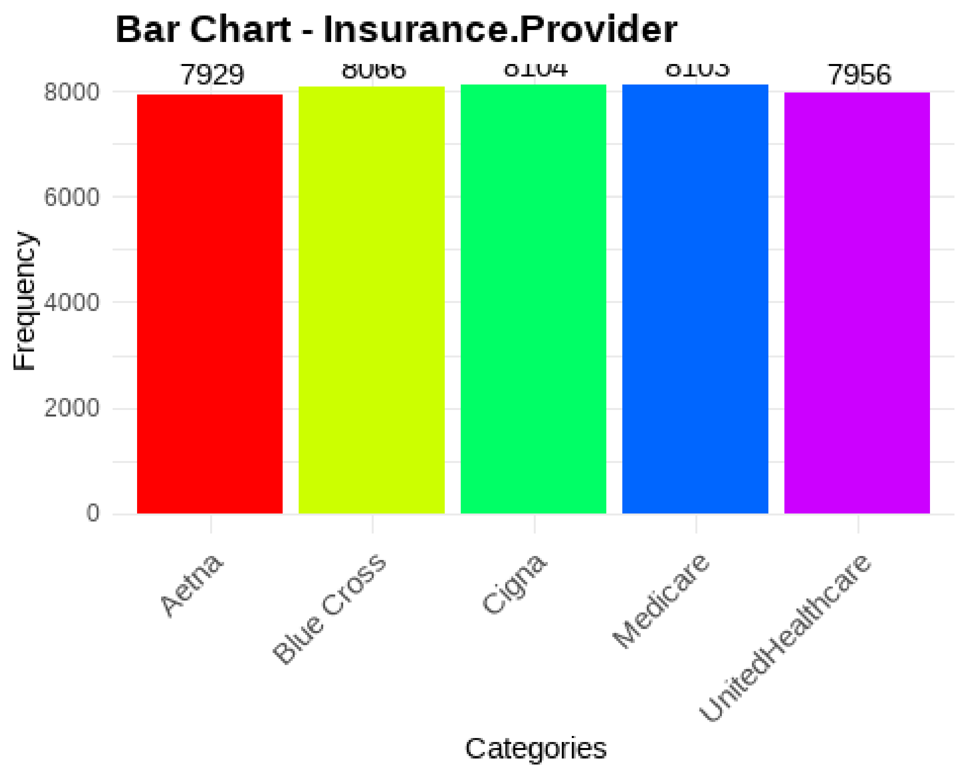
```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



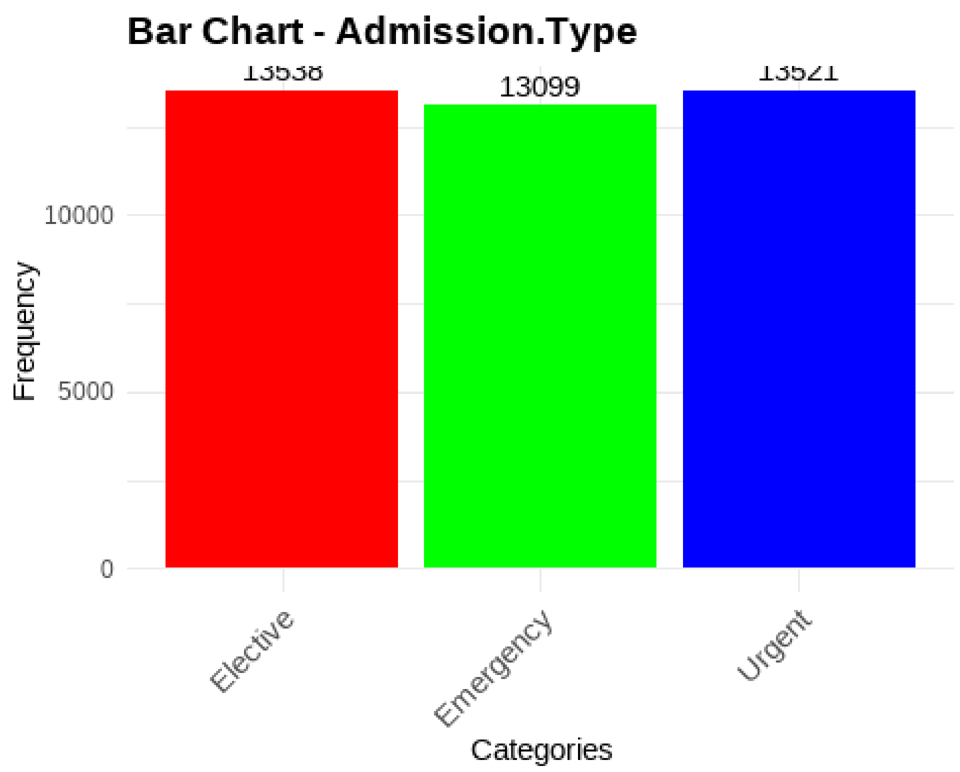
```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



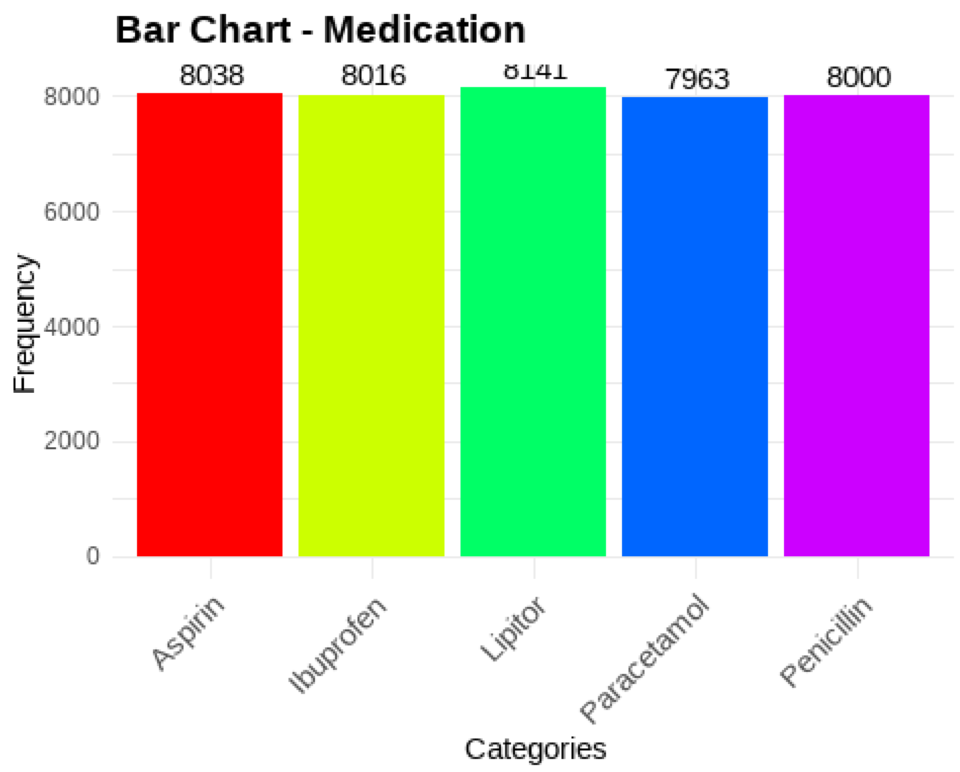
```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



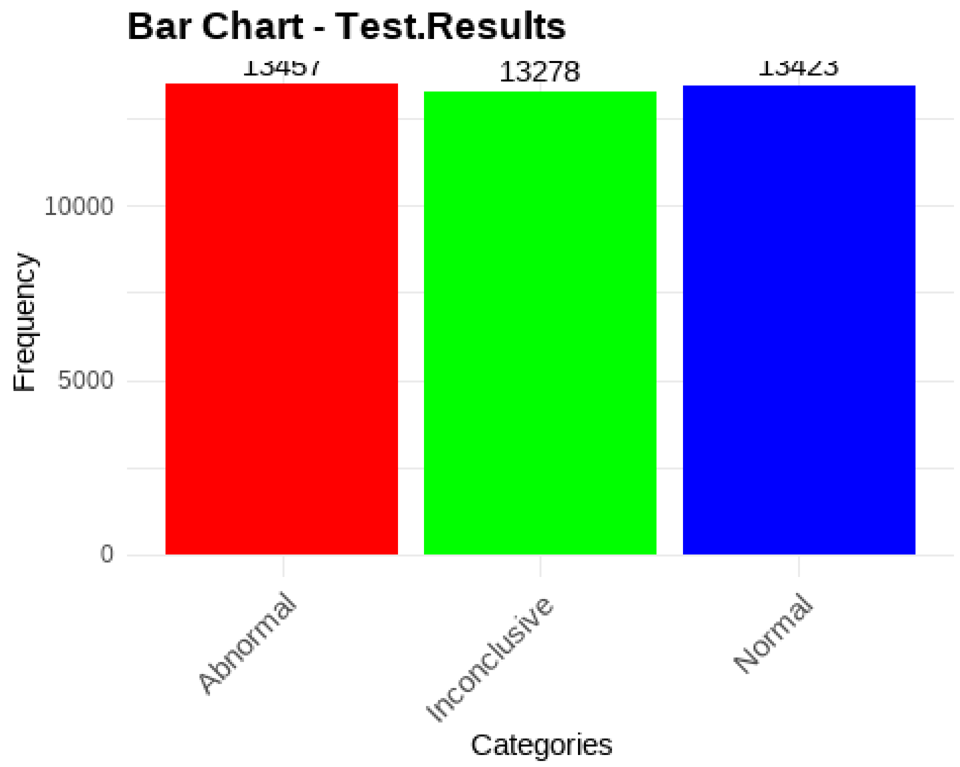
```
## [1] "Bar chart is not applicable for Billing.Amount"
## [1] "Bar chart is not applicable for Room.Number"
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



```
## Warning in `[<-.factor`(`*tmp*`, is.na(selected_var), value = "Unknown"
):
## invalid factor level, NA generated
```



Analysis: -

Verifies pie chart analysis as the output shows similar results

Appendix 4: Variable Selection And Regression

As we can see from the EDA above the response variable (the outcome), admission type, is categorical data. Therefore, since the response variable only has 3 outcomes even though it has multiple independent variables (a factor that can cause changes in the response variable), so we are going to use mutliclass logistic regression.

Appendix 4.1: Loading Necessary Packages

```
library(MASS)   # Used for advanced statistical modelling and variable sel
ection

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

library(glmnet) # Used to fit the model

## Loading required package: Matrix

##
## Attaching package: 'Matrix'
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-10

library(Matrix) # For building Matrices
library(nnet)   # Used to fit the model
library(caret)  # Used for cross-validation, classification, building, tun
ing, and evaluating the model

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

library(hnp)    # Used to test for goodness of fit after variable selectio
n
```

Appendix 4.2: Variable Selection

Appendix 4.2.1: Designing The Data For Model Fitting

```
# First, we make the predictors into a matrix and outcome as the factor ou
tcome for admission type
x <- model.matrix(Admission.Type ~ ., data = health)[, -1]
y <- health$Admission.Type
```

Appendix 4.2.2: Select Coefficients

```
## Pure LASSO
fit_lasso <- cv.glmnet(
  x, y,
  family = "multinomial",
  alpha = 1 # LASSO
  )

# Best shrinkage parameter
fit_lasso$lambda.min

## [1] 0.006732435

# Selected coefficients
coef(fit_lasso, s = "lambda.min")

## $Elective
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                                      lambda.min
## (Intercept)                          0.01140706
## Age                                  .
## GenderMale                           .
## Blood.TypeA+                         .
## Blood.TypeAB-                        .
## Blood.TypeAB+                        .
## Blood.TypeB-                      .
## Blood.TypeB+                      .
## Blood.TypeO-                      .
## Blood.TypeO+                      .
## Medical.ConditionAsthma           .
## Medical.ConditionCancer           .
## Medical.ConditionDiabetes         .
## Medical.ConditionHypertension     .
## Medical.ConditionObesity          .
## Insurance.ProviderBlue Cross      .
## Insurance.ProviderCigna           .
## Insurance.ProviderMedicare        .
## Insurance.ProviderUnitedHealthcare .
## Billing.Amount                    .
## Room.Number                       .
## MedicationIbuprofen               .
## MedicationLipitor                 .
## MedicationParacetamol             .
## MedicationPenicillin              .
## Test.ResultsInconclusive          .
## Test.ResultsNormal                .
## 
## $Emergency
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                                    lambda.min
## (Intercept)                        -0.0215576
## Age                                 .
## GenderMale                          .
## Blood.TypeA+                        .
## Blood.TypeAB-                       .
## Blood.TypeAB+                       .
## Blood.TypeB-                        .
## Blood.TypeB+                        .
## Blood.TypeO-                        .
## Blood.TypeO+                        .
## Medical.ConditionAsthma             .
## Medical.ConditionCancer             .
## Medical.ConditionDiabetes           .
## Medical.ConditionHypertension       .
## Medical.ConditionObesity            .
## Insurance.ProviderBlue Cross        .
## Insurance.ProviderCigna             .
## Insurance.ProviderMedicare          .
## Insurance.ProviderUnitedHealthcare  .
## Billing.Amount                      .
## Room.Number                         .
## MedicationIbuprofen                 .
## MedicationLipitor                   .
## MedicationParacetamol               .
## MedicationPenicillin                .
## Test.ResultsInconclusive            .
## Test.ResultsNormal                  .
## 
## $Urgent
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                                      lambda.min
## (Intercept)                          0.01015054
## Age                                  .
## GenderMale                           .
## Blood.TypeA+                         .
## Blood.TypeAB-                        .
## Blood.TypeAB+                        .
## Blood.TypeB-                         .
## Blood.TypeB+                         .
## Blood.TypeO-                         .
## Blood.TypeO+                         .
## Medical.ConditionAsthma              .
## Medical.ConditionCancer              .
## Medical.ConditionDiabetes            .
## Medical.ConditionHypertension        .
## Medical.ConditionObesity             .
## Insurance.ProviderBlue Cross         .
## Insurance.ProviderCigna              .
## Insurance.ProviderMedicare           .
## Insurance.ProviderUnitedHealthcare .
## Billing.Amount                       .
## Room.Number                          .
## MedicationIbuprofen                  .
## MedicationLipitor                    .
## MedicationParacetamol                .
## MedicationPenicillin                 .
## Test.ResultsInconclusive             .
## Test.ResultsNormal                   .

# Given that there are only values in the intercept, none of the variables
 were selected in this case
# But, since LASSO is quite an aggressive form of variable selection, we c
an try to use more ridge by reducing the alpha

## A Balanced Elastic Net
fit_lasso <- cv.glmnet(
  x, y,
  family = "multinomial",
  alpha = 0.5
  )

# Best shrinkage parameter
fit_lasso$lambda.min

## [1] 0.01346487

# Selected coefficients
coef(fit_lasso, s = "lambda.min")

## $Elective
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                                      lambda.min
## (Intercept)                          0.01140706
## Age                                  .
## GenderMale                         .
## Blood.TypeA+                       .
## Blood.TypeAB-                      .
## Blood.TypeAB+                      .
## Blood.TypeB-                       .
## Blood.TypeB+                       .
## Blood.TypeO-                       .
## Blood.TypeO+                       .
## Medical.ConditionAsthma            .
## Medical.ConditionCancer            .
## Medical.ConditionDiabetes          .
## Medical.ConditionHypertension      .
## Medical.ConditionObesity           .
## Insurance.ProviderBlue Cross       .
## Insurance.ProviderCigna            .
## Insurance.ProviderMedicare         .
## Insurance.ProviderUnitedHealthcare .
## Billing.Amount                     .
## Room.Number                        .
## MedicationIbuprofen                .
## MedicationLipitor                  .
## MedicationParacetamol              .
## MedicationPenicillin               .
## Test.ResultsInconclusive           .
## Test.ResultsNormal                 .
##
## $Emergency
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                                    lambda.min
## (Intercept)                        -0.0215576
## Age                                 .
## GenderMale                          .
## Blood.TypeA+                        .
## Blood.TypeAB-                       .
## Blood.TypeAB+                       .
## Blood.TypeB-                        .
## Blood.TypeB+                        .
## Blood.TypeO-                        .
## Blood.TypeO+                        .
## Medical.ConditionAsthma             .
## Medical.ConditionCancer             .
## Medical.ConditionDiabetes           .
## Medical.ConditionHypertension       .
## Medical.ConditionObesity            .
## Insurance.ProviderBlue Cross        .
## Insurance.ProviderCigna             .
## Insurance.ProviderMedicare          .
## Insurance.ProviderUnitedHealthcare  .
## Billing.Amount                      .
## Room.Number                         .
## MedicationIbuprofen                 .
## MedicationLipitor                   .
## MedicationParacetamol               .
## MedicationPenicillin                .
## Test.ResultsInconclusive            .
## Test.ResultsNormal                  .
##
## $Urgent
## 27 x 1 sparse Matrix of class "dgCMatrix"
##                                    lambda.min
## (Intercept)                        0.01015054
## Age                                .
## GenderMale                         .
## Blood.TypeA+                       .
## Blood.TypeAB-                      .
## Blood.TypeAB+                      .
## Blood.TypeB-                       .
## Blood.TypeB+                       .
## Blood.TypeO-                       .
## Blood.TypeO+                       .
## Medical.ConditionAsthma            .
## Medical.ConditionCancer            .
## Medical.ConditionDiabetes          .
## Medical.ConditionHypertension      .
## Medical.ConditionObesity           .
## Insurance.ProviderBlue Cross       .
## Insurance.ProviderCigna            .
## Insurance.ProviderMedicare         .
## Insurance.ProviderUnitedHealthcare .
## Billing.Amount                     .
## Room.Number                        .
## MedicationIbuprofen                .
## MedicationLipitor                  .
## MedicationParacetamol              .
## MedicationPenicillin               .
## Test.ResultsInconclusive           .
## Test.ResultsNormal                 .

## We can see that there are selected variables now, but they are weak. So
, we will not consider them when doing the predictions
## However, we can still use it for further interpretation later on. Hence
, we will get the names of the selected variables for later

# Get exact names of the variables selected
selected_vars <- unique(unlist(lapply(
  coef(fit_lasso, s = "lambda.min"),
  function(m) rownames(m)[as.numeric(m) != 0]
)))
selected_vars <- setdiff(selected_vars, "(Intercept)")
selected_vars

## character(0)
```

Appendix 4.3: Prediction

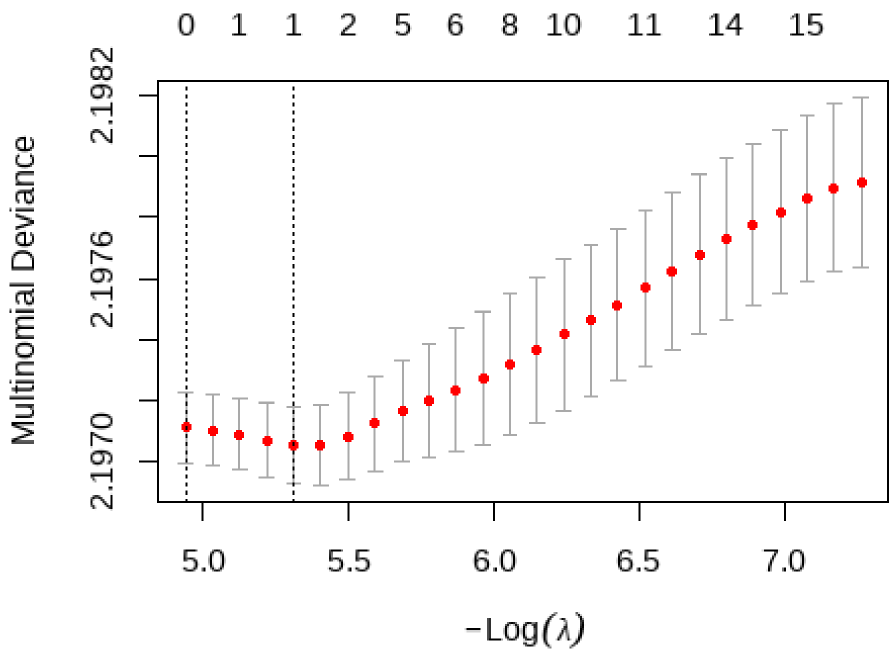
Appendix 4.3.1: Splitting The Data

```
set.seed(100)# For reproducibility
```

```r
# Randomly select 80% of the row as the split
split <- sample(1:nrow(health), 0.8 * nrow(health))
train_data <- health[split, ]  # Use the 80% selected for training
test_data <- health[-split, ]  # Use the remaining for testing

# Build matrices for training data
x_train <- sparse.model.matrix(Admission.Type ~ ., data = train_data)[,-1]
y_train <- train_data$Admission.Type

# Build matrices for test data
x_test <- sparse.model.matrix(Admission.Type ~ ., data = test_data)[,-1]
y_test <- test_data$Admission.Type
```

Appendix 4.3.2: Fit the LASSO On Training Data And Test Data

```r
## Training Data
fit_lasso_train <- cv.glmnet(
  x_train, y_train,
  family = "multinomial",
  alpha = 1
)

## Test Data
fit_lasso_test <- cv.glmnet(
  x_test, y_test,
  family = "multi",
  alpha = 1
)

# Plotting cross-validation curve
plot(fit_lasso_train)
```



Analysis: - Shows that the lambda.min (best fit model) and the lambda.1se (simpler model) will give similar outcomes. - So, we will just use the best fit model (which uses all the independent variables) going forward when we do predictions, seeing that there are not much benefits using a simpler model - However, we can still use the variable selection to see what other/further effects it might give to the model

Appendix 4.3.3: Making Predictions Using The Full Model

```
## Training data
pred_probs_train <- predict(fit_lasso_train, newx = x_train, s = "lambda.m
in", type = "response")

## Test Data
pred_probs_test <- predict(fit_lasso_test, newx = x_test, s = "lambda.min"
, type = "response")
```

Appendix 4.3.4: Prediction Accuracy

```
pred <- predict(fit_lasso_train, newx = x_test, s = "lambda.min", type = "
class")
mean(pred == y_test)

## [1] 0.3367779
```

Analysis: Model is about 34% accurate, which is low. Though this was expected as all the variables were fairly evenly spread out, so we already expected that the predictions may not be very accurate

Appendix 4.4: Variable Selection Interpretation

Appendix 4.4.1: Summary of Variable Selection Fit

```
# Refit the model with the selected variables
fit_refit <- multinom(
  Admission.Type ~ Age +
                   Blood.Type +
                   Medical.Condition +
                   Gender +
                   Insurance.Provider +
                   Medication,
  data = health
)

## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44088.289404
## iter  20 value 44087.563627
## iter  30 value 44085.872675
## iter  40 value 44084.599046
## final  value 44084.511396
## converged

summary(fit_refit) # To see key outcomes of the model

## Call:
## multinom(formula = Admission.Type ~ Age + Blood.Type + Medical.Conditio
n +
##     Gender + Insurance.Provider + Medication, data = health)
##
## Coefficients:
##             (Intercept)          Age Blood.TypeA+ Blood.TypeAB- Blood.Typ
eAB+
## Emergency  -0.1188004 0.0010224649  0.003702696  -0.008871348    0.0310
7263
## Urgent     -0.1681841 0.0008012514 -0.008373136  -0.037173542    0.0483
9050
##            Blood.TypeB- Blood.TypeB+ Blood.TypeO- Blood.TypeO+
## Emergency -0.008126562   0.06930849  -0.05008149  -0.02489878
## Urgent     0.010135709   0.04945616  -0.03906435  -0.05885322
##            Medical.ConditionAsthma Medical.ConditionCancer
## Emergency             0.0009387206            -0.032504809
## Urgent                0.0027392519             0.008587153
##            Medical.ConditionDiabetes Medical.ConditionHypertension
## Emergency                -0.00606062                   -0.05548912
## Urgent                    0.06541303                   -0.02156108
##            Medical.ConditionObesity  GenderMale Insurance.ProviderBlue C
ross
## Emergency                0.07211601 0.004905793                  0.00997
1167
## Urgent                   0.02947333 0.048218580                  0.05507
0123
##            Insurance.ProviderCigna Insurance.ProviderMedicare
## Emergency               0.02483461                 0.02725148
## Urgent                  0.11904184                 0.10303487
##            Insurance.ProviderUnitedHealthcare MedicationIbuprofen
## Emergency                      -0.0008272783          0.02125015
## Urgent                          0.0527720631          0.03707748
##            MedicationLipitor MedicationParacetamol MedicationPenicillin
## Emergency        0.06540553          -0.001774407          0.01635660
## Urgent           0.07000940          -0.017796399          0.03519527
##
## Std. Errors:
##            (Intercept)          Age Blood.TypeA+ Blood.TypeAB- Blood.Typ
eAB+
## Emergency  0.06581059 0.0006264306   0.04907518    0.04904322    0.0490
5811
## Urgent     0.06551693 0.0006215423   0.04869136    0.04875890    0.0485
0023
##            Blood.TypeB- Blood.TypeB+ Blood.TypeO- Blood.TypeO+
## Emergency   0.04915948   0.04907149   0.04908890   0.04904615
## Urgent      0.04859188   0.04873598   0.04856297   0.04880259
##            Medical.ConditionAsthma Medical.ConditionCancer
## Emergency              0.04231693              0.04228561
## Urgent                 0.04216518              0.04192693
##            Medical.ConditionDiabetes Medical.ConditionHypertension
## Emergency                0.04227462                    0.04224628
## Urgent                   0.04176560                    0.04192420
##            Medical.ConditionObesity GenderMale Insurance.ProviderBlue Cr
oss
## Emergency               0.04224930 0.02452960                   0.03863
408
## Urgent                  0.04233767 0.02433821                   0.03863
961
##            Insurance.ProviderCigna Insurance.ProviderMedicare
## Emergency              0.03877443                 0.03871923
## Urgent                 0.03854424                 0.03857228
##            Insurance.ProviderUnitedHealthcare MedicationIbuprofen
## Emergency                         0.03878315          0.03877667
## Urgent                            0.03874204          0.03844206
##            MedicationLipitor MedicationParacetamol MedicationPenicillin
## Emergency        0.03866484            0.03870538           0.03879883
## Urgent           0.03838771            0.03852948           0.03845511
##
## Residual Deviance: 88169.02
## AIC: 88261.02
```

Analysis: - Age: Looks that older patient are slightly more likely to have Emergency or Urgent admissions - Blood types: Admission types do vary by blood groups - Medical condition: Asthma patients are more likely to have more emergency or Urgent admissions. Diabetic patients do have more urgent admissions, though less for emergency admissions - Gender: Males shows to have slightly more likely urgent admissions compared to females - Insurance providers: Blue cross, cigna, medicare and United healthcare generally have higher number of emergency or urgent admitted patients with signa and medicare having stronger positive chance for urgent admissions - Medication: Lipitor shows some increasing effects on emergency and urgent admissions with Ibuprofen and Penicillin also having slightly lesser increasing effect on emergency and urgent admissions, but Paracetamol shows slight decreasing effect on urgent admissions

Appendix 4.4.2: Odds Ratios And Standard Errors

To see the extent of the effects of the variables and how statistically significant are the variables

```
# Using the fit_refit we got just now, we get the log-odds coefficients to
 find the odds ratios
# Log-odds ratio
coefs <- coef(fit_refit)

# Compute odds ratios
O.R. <- exp(coefs)

# Standard errors
s.e.s <- summary(fit_refit)$standard.errors
```

Appendix 4.4.3: Confidence Intervals (95%)

```
lower_CI <- exp(coefs - 1.96 * s.e.s)
upper_CI <- exp(coefs + 1.96 * s.e.s)
```

Appendix 4.4.4: Combine Into A Table

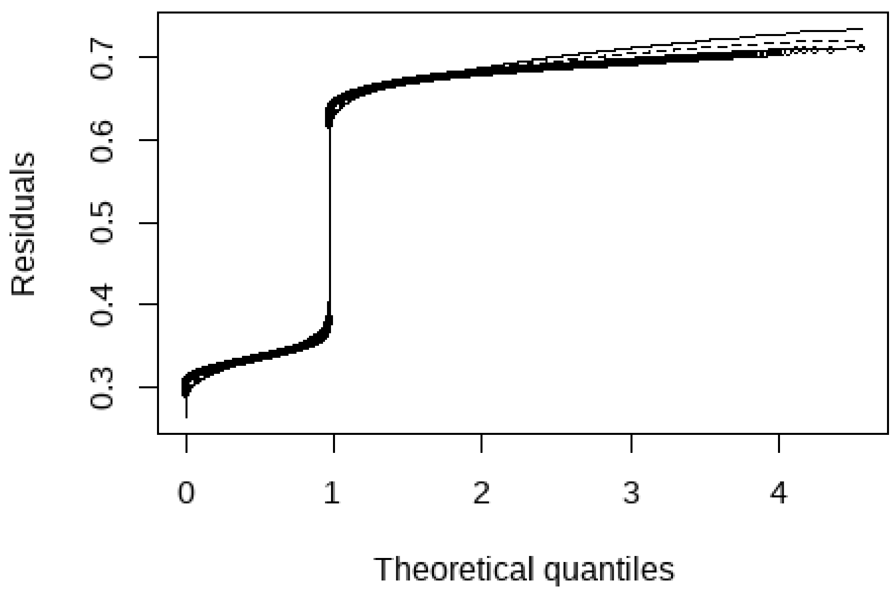
```
results <- as.data.frame(coefs) %>%
  mutate(Outcome = rownames(coefs)) %>%
  pivot_longer(-Outcome, names_to = "Variable", values_to = "Estimate") %>
%
  mutate(
    OR = exp(Estimate),
    SE = as.vector(s.e.s),
    CI_lower = exp(Estimate - 1.96 * SE),
    CI_upper = exp(Estimate + 1.96 * SE)
  )

# Show the whole table
results

## # A tibble: 46 × 7
##    Outcome   Variable                         Estimate    OR       SE CI_lower
CI_upper
##    <chr>     <chr>                               <dbl> <dbl>    <dbl>    <dbl>
   <dbl>
##  1 Emergency (Intercept)                     -0.119    0.888 0.0658      0.781
   1.01
##  2 Emergency Age                              0.00102  1.00  0.0655      0.880
   1.14
##  3 Emergency Blood.TypeA+                     0.00370  1.00  0.000626    1.00
   1.00
##  4 Emergency Blood.TypeAB-                   -0.00887  0.991 0.000622    0.990
   0.992
##  5 Emergency Blood.TypeAB+                    0.0311   1.03  0.0491      0.937
   1.14
##  6 Emergency Blood.TypeB-                    -0.00813  0.992 0.0487      0.902
   1.09
##  7 Emergency Blood.TypeB+                     0.0693   1.07  0.0490      0.974
   1.18
```

```
##  8 Emergency Blood.TypeO-            -0.0501   0.951 0.0488      0.864
   1.05
##  9 Emergency Blood.TypeO+            -0.0249   0.975 0.0491      0.886
   1.07
## 10 Emergency Medical.ConditionAsthma  0.000939 1.00  0.0485      0.910
   1.10
## # ℹ 36 more rows
```

Analysis: - Most predictors (independent variabels) shows small or little effects on admission types - Blood type generally has weak and inconsistent effects - Insurance provider, being male, obese, diabetic does show some mild positive effect on emergency or urgent admissions compared to elective admissions.

Appendix 4.4.5: Goodness of Fit

```
# Make custom pearson residuals function
pearson_resid <- function(fit) {
  p <- fitted(fit) # Predicted probabilities
  y <- model.matrix(~ Admission.Type - 1, data = fit$model)

  res <- (y - p) / sqrt(p * (1 - p)) # Formula for residuals
  apply(res, 1, max) # Apply the maximum residual per observation
}

# Run HNP using the custom pearson residuals function and the refitted mod
el with the selected variables
hnp(fit_refit, resid.fun = pearson_resid, nsim = 99)

## Multinomial model
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44075.340925
## iter  20 value 44071.753372
## iter  30 value 44069.219867
## final  value 44068.721073
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44076.571002
## iter  20 value 44075.037201
## iter  30 value 44071.160380
## iter  40 value 44069.996397
## final  value 44069.863806
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44087.005894
## iter  20 value 44084.315668
## iter  30 value 44083.750573
## iter  40 value 44083.364132
## final  value 44083.225907
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44076.627809
## iter  20 value 44075.264697
## iter  30 value 44072.009659
## iter  40 value 44071.643064
## final  value 44071.603774
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44067.014931
## iter  20 value 44065.540644
## iter  30 value 44061.766137
## iter  40 value 44060.066780
## final  value 44059.617084
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44069.632705
## iter  20 value 44065.596578
## iter  30 value 44060.043300
## final  value 44057.837188
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44072.437779
## iter  20 value 44071.668027
## iter  30 value 44069.622809
## final  value 44068.073847
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44068.932884
## iter  20 value 44065.807030
## iter  30 value 44064.623776
## iter  40 value 44061.921070
## final  value 44061.813370
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44069.417689
## iter  20 value 44067.875016
## iter  30 value 44062.432504
## iter  40 value 44060.871238
## final  value 44060.471964
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44070.667653
## iter  20 value 44069.043138
## iter  30 value 44062.521381
## iter  30 value 44062.521163
## final  value 44062.519937
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44063.419610
## iter  20 value 44061.431162
## iter  30 value 44056.410270
## iter  40 value 44055.254126
## final  value 44055.160733
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44071.577922
## iter  20 value 44070.595757
## iter  30 value 44067.445209
## iter  40 value 44065.943522
## final  value 44065.828709
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44067.686041
## iter  20 value 44066.337577
## iter  30 value 44062.504904
## iter  40 value 44060.894509
## final  value 44060.356293
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44077.512266
## iter  20 value 44075.462154
## iter  30 value 44072.821963
## iter  40 value 44072.370502
## final  value 44072.306131
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44071.774904
## iter  20 value 44070.349910
## iter  30 value 44067.294599
## iter  40 value 44066.223775
## final  value 44065.530569
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44056.555422
## iter  20 value 44054.848188
## iter  30 value 44049.437300
## iter  40 value 44047.949280
## final  value 44047.096203
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44075.631537
## iter  20 value 44071.973453
## iter  30 value 44069.538567
## iter  40 value 44068.635311
## final  value 44068.483739
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44068.757904
## iter  20 value 44067.546772
## iter  30 value 44062.902570
## iter  40 value 44061.911919
## final  value 44061.758529
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44060.171701
## iter  20 value 44056.514251
## iter  30 value 44047.876877
## iter  40 value 44046.604448
## final  value 44046.521418
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44061.823757
## iter  20 value 44056.311417
## iter  30 value 44052.477486
## iter  40 value 44051.773640
## final  value 44051.662848
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44067.389339
## iter  20 value 44065.593918
## iter  30 value 44059.626633
## iter  40 value 44058.859107
## final  value 44058.468365
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44073.343424
## iter  20 value 44070.385618
## iter  30 value 44065.764302
## iter  40 value 44064.594415
## final  value 44064.455951
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44068.535499
## iter  20 value 44066.759198
## iter  30 value 44062.070993
## iter  40 value 44059.780700
## final  value 44059.422211
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44073.234343
## iter  20 value 44068.379968
## iter  30 value 44063.428420
## final  value 44062.922266
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44072.864321
## iter  20 value 44069.991529
## iter  30 value 44062.797188
## final  value 44061.504653
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44066.512369
## iter  20 value 44064.614015
## iter  30 value 44060.042403
## iter  40 value 44057.064353
## final  value 44056.667725
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44049.712560
## iter  20 value 44048.271933
## iter  30 value 44042.681786
## iter  40 value 44041.201765
## final  value 44041.120788
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44066.575687
## iter  20 value 44065.541726
## iter  30 value 44062.681792
## iter  40 value 44061.790749
## final  value 44061.734413
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44066.631688
## iter  20 value 44065.078746
## iter  30 value 44060.359896
## iter  40 value 44057.107185
## final  value 44056.809745
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44082.080254
## iter  20 value 44080.589386
## iter  30 value 44076.761391
## iter  40 value 44076.242485
## final  value 44076.182507
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44084.000178
## iter  20 value 44081.872036
## iter  30 value 44078.574136
## iter  40 value 44077.440425
## final  value 44077.334000
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44078.548816
## iter  20 value 44077.277790
## iter  30 value 44074.830396
## iter  40 value 44073.233230
## final  value 44073.166428
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44066.991060
## iter  20 value 44063.998185
## iter  30 value 44058.858721
## iter  40 value 44057.891634
## final  value 44057.852031
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44071.181098
## iter  20 value 44070.228483
## iter  30 value 44066.580838
## iter  40 value 44065.554677
## final  value 44065.445861
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44070.758068
## iter  20 value 44067.443849
## iter  30 value 44059.625232
## iter  40 value 44057.646969
## final  value 44057.303927
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44074.001523
## iter  20 value 44070.446099
## iter  30 value 44067.726160
## iter  40 value 44066.365880
## final  value 44066.238586
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44065.054266
## iter  20 value 44063.472424
## iter  30 value 44060.305504
## iter  40 value 44059.216010
## final  value 44059.034406
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44079.358943
## iter  20 value 44077.467468
## iter  30 value 44069.921393
## iter  40 value 44068.638661
## final  value 44068.250659
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44079.657018
## iter  20 value 44077.681879
## iter  30 value 44071.770473
## iter  40 value 44070.904441
## final  value 44070.288746
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44068.163989
## iter  20 value 44067.098990
## iter  30 value 44065.016576
## final  value 44064.317226
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44081.056844
## iter  20 value 44077.734813
## iter  30 value 44074.863752
## iter  40 value 44074.417476
## final  value 44074.286743
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44080.418453
## iter  20 value 44078.961204
## iter  30 value 44075.663620
## final  value 44073.561916
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44085.315094
## iter  20 value 44083.776273
## iter  30 value 44079.653592
## iter  40 value 44078.225125
## final  value 44078.087521
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44070.815900
## iter  20 value 44069.550461
## iter  30 value 44065.566520
## iter  40 value 44064.948042
## final  value 44064.711083
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44064.550482
## iter  20 value 44062.236585
## iter  30 value 44055.749735
## iter  40 value 44054.648465
## final  value 44054.265013
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44078.215236
## iter  20 value 44076.817989
## iter  30 value 44072.781973
## iter  40 value 44070.582814
## final  value 44070.442672
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44058.896265
## iter  20 value 44055.512049
## iter  30 value 44051.541606
## iter  40 value 44047.215682
## iter  50 value 44046.526523
## iter  50 value 44046.526282
## iter  50 value 44046.526280
## final  value 44046.526280
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44050.444315
## iter  20 value 44047.339199
## iter  30 value 44043.954353
## iter  40 value 44043.128538
## final  value 44042.963859
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44072.842578
## iter  20 value 44071.007199
## iter  30 value 44067.863960
## final  value 44067.263116
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44076.689179
## iter  20 value 44075.535957
## iter  30 value 44072.671424
## iter  40 value 44071.080655
## final  value 44070.396830
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44061.278181
## iter  20 value 44057.911302
## iter  30 value 44054.031211
## final  value 44052.596024
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44078.898473
## iter  20 value 44075.800749
## iter  30 value 44073.973093
## iter  40 value 44073.576875
## iter  40 value 44073.576710
## iter  40 value 44073.576708
## final  value 44073.576708
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44069.076976
## iter  20 value 44066.726965
## iter  30 value 44058.813616
## final  value 44057.411685
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44072.656339
## iter  20 value 44068.848647
## iter  30 value 44064.975411
## iter  40 value 44063.237496
## final  value 44063.122627
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44064.964896
## iter  20 value 44063.213598
## iter  30 value 44059.343359
## iter  30 value 44059.343269
## final  value 44058.629583
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44061.733613
## iter  20 value 44059.657841
## iter  30 value 44052.917954
## final  value 44051.003443
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44068.171663
## iter  20 value 44064.473629
## iter  30 value 44060.519440
## iter  40 value 44059.160630
## final  value 44059.107793
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44075.155221
## iter  20 value 44074.298026
## iter  30 value 44071.711654
## final  value 44071.023013
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44057.384515
## iter  20 value 44055.481782
## iter  30 value 44051.080136
## iter  40 value 44049.559464
## final  value 44049.339890
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44061.082043
## iter  20 value 44060.144349
## iter  30 value 44057.463552
## final  value 44056.670878
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44077.146362
## iter  20 value 44076.051422
## iter  30 value 44072.197089
## iter  40 value 44071.417172
## final  value 44071.118751
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44070.921554
## iter  20 value 44070.049918
## iter  30 value 44067.019365
## iter  40 value 44065.359104
## final  value 44065.196246
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44069.542899
## iter  20 value 44067.324847
## iter  30 value 44062.354447
## iter  40 value 44061.546456
## final  value 44061.395451
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44077.590018
## iter  20 value 44074.201367
## iter  30 value 44071.151460
## iter  40 value 44070.378815
## final  value 44070.225482
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44068.519232
## iter  20 value 44066.341881
## iter  30 value 44063.971351
## iter  40 value 44062.990519
## final  value 44062.849420
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44067.743396
## iter  20 value 44065.921025
## iter  30 value 44060.427568
## iter  40 value 44059.474461
## final  value 44059.165831
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44077.504031
## iter  20 value 44076.200983
## iter  30 value 44074.824056
## iter  40 value 44074.657879
## final  value 44074.470199
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44075.038364
## iter  20 value 44072.514498
## iter  30 value 44070.684630
## iter  40 value 44069.485823
## final  value 44069.417410
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44063.963832
## iter  20 value 44062.723061
## iter  30 value 44057.920657
## iter  40 value 44057.211343
## final  value 44056.944763
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44053.698289
## iter  20 value 44050.184289
## iter  30 value 44040.938256
## iter  40 value 44039.863165
## final  value 44039.675985
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44060.483880
## iter  20 value 44057.888836
## iter  30 value 44053.780136
## iter  40 value 44052.709883
## final  value 44052.564184
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44067.321494
## iter  20 value 44063.493851
## iter  30 value 44062.182694
## final  value 44061.654625
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44080.741294
## iter  20 value 44078.049540
## iter  30 value 44076.462595
## final  value 44075.852037
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44065.141622
## iter  20 value 44062.977424
## iter  30 value 44055.339005
## final  value 44054.807333
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44055.169798
## iter  20 value 44052.868116
## iter  30 value 44048.585627
## iter  40 value 44047.553983
## final  value 44047.533450
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44065.435604
## iter  20 value 44060.925202
## iter  30 value 44050.133971
## iter  40 value 44047.483379
## final  value 44047.321891
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44081.186634
## iter  20 value 44080.112917
## iter  30 value 44076.892506
## iter  40 value 44076.153427
## final  value 44075.844043
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44072.705572
## iter  20 value 44069.222560
## iter  30 value 44064.468477
## iter  40 value 44062.936966
## final  value 44062.805570
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44064.566755
## iter  20 value 44062.186559
## iter  30 value 44055.348274
## iter  30 value 44055.347885
## iter  40 value 44054.364095
## iter  50 value 44054.220968
## final  value 44054.107057
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44056.780219
## iter  20 value 44054.761082
## iter  30 value 44046.837431
## final  value 44044.712288
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44057.984806
## iter  20 value 44056.710655
## iter  30 value 44051.422328
## iter  40 value 44050.233596
## final  value 44049.159781
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44064.366007
## iter  20 value 44062.799838
## iter  30 value 44057.935209
## final  value 44056.457547
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44062.162730
## iter  20 value 44060.477520
## iter  30 value 44055.365679
## iter  40 value 44054.398827
## final  value 44054.310018
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44081.816096
## iter  20 value 44078.645834
## iter  30 value 44075.412779
## iter  40 value 44073.874449
## final  value 44073.517687
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44076.744257
## iter  20 value 44075.347639
## iter  30 value 44071.313075
## iter  30 value 44071.312776
## iter  40 value 44070.919943
## iter  50 value 44070.879240
## final  value 44070.770461
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44061.225709
## iter  20 value 44056.976510
## iter  30 value 44054.345050
## iter  40 value 44053.289266
## final  value 44053.200305
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44062.746619
## iter  20 value 44060.029300
## iter  30 value 44056.402843
## iter  40 value 44055.659915
## final  value 44055.564488
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44066.040484
## iter  20 value 44064.542117
## iter  30 value 44060.886335
## iter  40 value 44058.786071
## final  value 44058.505066
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44070.823482
## iter  20 value 44068.172809
## iter  30 value 44062.668863
## iter  40 value 44061.761779
## final  value 44061.703977
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44070.197435
## iter  20 value 44064.602396
## iter  30 value 44059.162391
## iter  40 value 44057.781658
## final  value 44057.568548
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44062.324105
## iter  20 value 44061.058670
## iter  30 value 44058.170871
## iter  40 value 44057.426514
## final  value 44057.377398
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44085.808409
## iter  20 value 44084.961051
## iter  30 value 44082.082812
## iter  40 value 44081.725660
## final  value 44081.673952
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44053.087323
## iter  20 value 44051.936328
## iter  30 value 44047.972105
## final  value 44046.453936
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44053.997616
## iter  20 value 44052.391576
## iter  30 value 44047.395972
## iter  40 value 44045.955236
## final  value 44045.655580
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44059.149409
## iter  20 value 44058.234107
## iter  30 value 44056.250331
## final  value 44054.230510
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44066.631834
## iter  20 value 44064.879630
## iter  30 value 44059.747550
## iter  40 value 44058.441388
## final  value 44058.371771
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44077.325481
## iter  20 value 44075.965047
## iter  30 value 44072.199063
## iter  40 value 44071.127712
## final  value 44071.004935
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44064.109153
## iter  20 value 44058.781300
## iter  30 value 44048.464344
## iter  40 value 44047.361413
## final  value 44046.970481
## converged
## # weights:  72 (46 variable)
## initial  value 44118.072288
## iter  10 value 44074.030371
## iter  20 value 44072.917034
## iter  30 value 44070.423234
## iter  40 value 44069.878783
## final  value 44069.763830
## converged
```



Analysis: -
Given that this is fitted using a multinomial logistic regression model, the vertical line in the output is expected as the response variable is categorical and the residuals can only take a few values - Though with the smooth curve, it can be seen that the largest residuals are slightly outside the envelope which indicates there are missing predictors - This indicates that the fitted model after variable selection is a moderate-to-poor fit, which justifies the previous decision of using all the variables for the model instead of the model after variable selection

Appendix 4.5: Cross-Validation

Appendix 4.5.1: K-Fold Cross Validation

```
set.seed(100) # For reproducibility

## 5-fold cross-validation
# Define cross-validation settings
cv_control_5 <-trainControl(
  method = "cv",
  number = 5,
)

# Train multinomial logistic regression with CV
model_cv5 <- train(
  Admission.Type ~ .,
  data = health,
  method = "multinom",   # Tells that its a multinomial logistic regressio
n
  trControl = cv_control_5,
  trace = FALSE
)
```

```
# Print CV results
print(model_cv5)

## Penalized Multinomial Regression
##
## 40158 samples
##     9 predictor
##     3 classes: 'Elective', 'Emergency', 'Urgent'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 32126, 32127, 32127, 32127, 32125
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.3351761  -0.0003591819
##   1e-04  0.3351761  -0.0003595632
##   1e-01  0.3352259  -0.0002655195
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.

# Access resampling results, e.g., accuracy
model_cv5$resample$Accuracy

## [1] 0.3310508 0.3379405 0.3360727 0.3368605 0.3342050

mean(model_cv5$resample$Accuracy)

## [1] 0.3352259

## Next, we try 10-fold cross-validation to see if there are any improveme
nts
set.seed(100) # For reproducibility

## 10-fold cross-validation
# Define cross-validation settings
cv_control_10 <-trainControl(
  method = "cv",
  number = 10,
)

# Train multinomial logistic regression with CV
model_cv10 <- train(
  Admission.Type ~ .,
  data = health,
  method = "multinom",   # Tells that its a multinomial logistic regressio
n
  trControl = cv_control_10,
  trace = FALSE
)

# Print CV results
print(model_cv10)
```

```
## Penalized Multinomial Regression
##
## 40158 samples
##     9 predictor
##     3 classes: 'Elective', 'Emergency', 'Urgent'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 36142, 36142, 36142, 36141, 36143, 36144, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.3407289  0.007869052
##   1e-04  0.3407538  0.007905297
##   1e-01  0.3408534  0.008054431
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.
```

```
# Access resampling results, e.g., accuracy
model_cv10$resample$Accuracy
```

```
##  [1] 0.3249502 0.3453685 0.3452825 0.3336653 0.3338316 0.3484433 0.3361
554
##  [8] 0.3505976 0.3446215 0.3456175
```

```
mean(model_cv10$resample$Accuracy)
```

```
## [1] 0.3408534
```

Analysis 5-fold CV: - Average accuracy is about 33.5% which is very low - Kappa is negative, so the model is no better than pure guessing using 5-fold CV - Shows that model has very weak predictive power

Analysis 10-fold CV: - Average accuracy is about 33.8% which is only slightly better than 5-fold CV - Kappa is positive now, but (~0.004) still less than 0.05, so model is not much better than pure guessing - Still shows that model has very weak predictive power

Appendix 4.5.2: Confusion Matrix

```
confusionMatrix(factor(pred, levels = levels(y_test)), y_test)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Elective Emergency Urgent
##   Elective      2237      2194   2302
##   Emergency        0         0      0
##   Urgent         427       404    468
##
## Overall Statistics
##
##                Accuracy : 0.3368
##                  95% CI : (0.3264, 0.3472)
##     No Information Rate : 0.3449
##     P-Value [Acc > NIR] : 0.9381
##
##                   Kappa : 0.0045
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: Elective Class: Emergency Class: Urgent
## Sensitivity                   0.8397           0.0000       0.16895
## Specificity                   0.1624           1.0000       0.84208
## Pos Pred Value                0.3322              NaN       0.36028
## Neg Pred Value                0.6713           0.6765       0.65810
## Prevalence                    0.3317           0.3235       0.34487
## Detection Rate                0.2785           0.0000       0.05827
## Detection Prevalence          0.8383           0.0000       0.16173
## Balanced Accuracy             0.5011           0.5000       0.50551
```

Analysis: - About 33.7% is correct, which is what we have expected from the previous results - Kappa is 0.0045, which confirms the k-fold cross-validation - Though we can note here, that the model completely fails to predict emergency cases, but the model still performs at chance level for every class of admission type - Results indicate that the available predictors are not suitable or insufficient to distinguish admission types, perticularly emergency admissions

Appendix 5: Classification

Appendix 5.1: Loading Necessary Packages

```
library(e1071)        # For data mining and classification

##
## Attaching package: 'e1071'

## The following objects are masked from 'package:dlookr':
##
##     kurtosis, skewness

## The following object is masked from 'package:ggplot2':
##
##     element

library(randomForest) # For random forests

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(class)         # For KNN
library(kernlab)       # To back-end for SVM (Radial)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:scales':
##
##     alpha

## The following object is masked from 'package:purrr':
##
##     cross

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

Appendix 5.2: Resplit The Data

```r
set.seed(100)# For reproducibility

# Randomly select 80% of the row as the split
split <- sample(1:nrow(health), 0.8 * nrow(health))
train_data <- health[split, ]  # Use the 80% selected for training
test_data <- health[-split, ]  # Use the remaining for testing
```

Appendix 5.3: Reprocessing And Train Control

```r
# Set the instructions on how to train and evaluate the model
train_control <- trainControl(
  method = "cv",
  number = 5,
  classProbs = TRUE,
  summaryFunction = multiClassSummary # Multiclass performance matrix
)
```

Appendix 5.4: Random Forest

Appendix 5.4.1: Random Forest Model

```r
# Train the model
rf_model <- train(
  Admission.Type ~ .,
  data = train_data, # Use the training data set
  method = "ranger", # Fits an efficient RF, supports large datasets and m
ulticlass problems efficiently
  trControl = train_control, # Applies the previously set instructions
  tuneLength = 5 # Automatically tries different mtry values and selects t
he best one
)
```

```
## Growing trees.. Progress: 66%. Estimated remaining time: 15 seconds.
## Growing trees.. Progress: 64%. Estimated remaining time: 17 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 1 seconds.
## Growing trees.. Progress: 76%. Estimated remaining time: 10 seconds.
## Growing trees.. Progress: 81%. Estimated remaining time: 7 seconds.
## Growing trees.. Progress: 94%. Estimated remaining time: 2 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 8 seconds.
## Growing trees.. Progress: 74%. Estimated remaining time: 11 seconds.
## Growing trees.. Progress: 82%. Estimated remaining time: 6 seconds.
## Growing trees.. Progress: 83%. Estimated remaining time: 6 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 8 seconds.
## Growing trees.. Progress: 89%. Estimated remaining time: 3 seconds.
## Growing trees.. Progress: 84%. Estimated remaining time: 6 seconds.
## Growing trees.. Progress: 100%. Estimated remaining time: 0 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 11 seconds.
```

Appendix 5.4.2: Predictions and Evaluation

```
rf_pred <- predict(rf_model, newdata = test_data) # Predict
rf_acc <- mean(rf_pred == test_data$Admission.Type) # Accuracy
rf_cm <- confusionMatrix(rf_pred, test_data$Admission.Type) # Overall mode
l evaluation

print("Random Forest Accuracy:")

## [1] "Random Forest Accuracy:"

print(rf_acc)

## [1] 0.3306773

print("Random Forest Confusion Matrix:")

## [1] "Random Forest Confusion Matrix:"

print(rf_cm)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Elective Emergency Urgent
##   Elective      1201      1237   1276
##   Emergency      503       484    523
##   Urgent         960       877    971
##
## Overall Statistics
##
##                Accuracy : 0.3307
##                  95% CI : (0.3204, 0.3411)
##     No Information Rate : 0.3449
##     P-Value [Acc > NIR] : 0.9965
##
##                   Kappa : -0.0061
##
##  Mcnemar's Test P-Value : <2e-16
##
```

```
## Statistics by Class:
##
##                      Class: Elective Class: Emergency Class: Urgent
## Sensitivity                   0.4508          0.18630        0.3505
## Specificity                   0.5319          0.81119        0.6509
## Pos Pred Value                0.3234          0.32053        0.3458
## Neg Pred Value                0.6612          0.67587        0.6556
## Prevalence                    0.3317          0.32346        0.3449
## Detection Rate                0.1495          0.06026        0.1209
## Detection Prevalence          0.4624          0.18800        0.3496
## Balanced Accuracy             0.4913          0.49874        0.5007
```

Analysis: - Accuracy is still about 33.1%, which is almost the same as the regression model, no improvements - Though random forest does predict all 3 classes of admission types (unlike the multinomial model), but the predictions are almost uniformly spread - Kappa is negative here, which indicates that its worse than random guessing an ocnfirms that the predictions are essentially random

Appendix 5.5: K-Nearest Neighbors (KNN)

Appendix 5.5.1: KNN Model

```
knn_model <- train(
  Admission.Type ~ .,
  data = train_data, # Use training data
  method = "knn", # Use K-Nearest Neighbors classification
  preProcess = c("center", "scale"), # Standardise predictors
  trControl = train_control,# Applies the previously set instructions
  tuneLength = 10 # Automatically tries multiple k values and select the b
est one
)
```

Appendix 5.5.2: Predictions and Evaluation

```
knn_pred <- predict(knn_model, newdata = test_data) # Predict
knn_acc <- mean(knn_pred == test_data$Admission.Type) # Accuracy
knn_cm <- confusionMatrix(knn_pred, test_data$Admission.Type) # Overall mo
del evaluation

print("k-NN Accuracy:")

## [1] "k-NN Accuracy:"

print(knn_acc)

## [1] 0.3290588

print("k-NN Confusion Matrix:")

## [1] "k-NN Confusion Matrix:"

print(knn_cm)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Elective Emergency Urgent
##   Elective       938       927    993
##   Emergency      839       809    881
##   Urgent         887       862    896
##
## Overall Statistics
##
##                Accuracy : 0.3291
##                  95% CI : (0.3188, 0.3395)
##     No Information Rate : 0.3449
##     P-Value [Acc > NIR] : 0.9987
##
##                   Kappa : -0.0066
##
##  Mcnemar's Test P-Value : 0.0143
##
## Statistics by Class:
##
##                      Class: Elective Class: Emergency Class: Urgent
## Sensitivity                   0.3521           0.3114        0.3235
## Specificity                   0.6423           0.6835        0.6676
## Pos Pred Value                0.3282           0.3199        0.3388
## Neg Pred Value                0.6664           0.6749        0.6521
## Prevalence                    0.3317           0.3235        0.3449
## Detection Rate                0.1168           0.1007        0.1116
## Detection Prevalence          0.3558           0.3149        0.3293
## Balanced Accuracy             0.4972           0.4974        0.4955
```

Analysis: - Accuracy is about 32.9%, which is no better than random forests - Again, most predictions are almost evenly spread across all admission types - Again, kappa is negative, so model is no better than random guessing

Appendix 5.6: Linear Discriminant Analysis (LDA)

Appendix 5.6.1: LDA Model

```
lda_model <- train(
  Admission.Type ~ .,
  data = train_data, # Use training data
  method = "lda", # Use LDA classification
  trControl = train_control # Applies the previously set instructions
)
```

Appendix 5.6.4: Predictions And Evaluation

```
lda_pred <- predict(lda_model, newdata = test_data) # Predict
lda_acc <- mean(lda_pred == test_data$Admission.Type) # Accuracy
lda_cm <- confusionMatrix(lda_pred, test_data$Admission.Type) # Overall mo
del evaluation

print("LDA Accuracy:")

## [1] "LDA Accuracy:"

print(lda_acc)

## [1] 0.3359064
```

```
print("LDA Confusion Matrix:")

## [1] "LDA Confusion Matrix:"

print(lda_cm)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Elective Emergency Urgent
##   Elective      1298      1284   1343
##   Emergency      371       335    362
##   Urgent         995       979   1065
##
## Overall Statistics
##
##                Accuracy : 0.3359
##                  95% CI : (0.3256, 0.3464)
##     No Information Rate : 0.3449
##     P-Value [Acc > NIR] : 0.9558
##
##                   Kappa : 5e-04
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: Elective Class: Emergency Class: Urgent
## Sensitivity                   0.4872          0.12895        0.3845
## Specificity                   0.5106          0.86511        0.6249
## Pos Pred Value                0.3307          0.31367        0.3504
## Neg Pred Value                0.6674          0.67504        0.6585
## Prevalence                    0.3317          0.32346        0.3449
## Detection Rate                0.1616          0.04171        0.1326
## Detection Prevalence          0.4887          0.13297        0.3784
## Balanced Accuracy             0.4989          0.49703        0.5047
```

Analysis: - Accuracy is about 33.6%, similar to previous results - Again, most predictions are almost evenly spread across all admission types - Kappa is positive here, but (0.0003) still less than 0.05, effectively no better than random guessing

- Overall, every model gives a chance performance, so the problem is not the model choice.
- It may be that the predictors do not contain much usable information to distinguish admission types

Appendix 6: Deep Learning

Appendix 6.1: Loading Necessary Packages

```
library(keras3) # Used for deep learning
library(tensorflow) # Allows to run custom deep-learning computations

##
## Attaching package: 'tensorflow'
## The following objects are masked from 'package:keras3':
##
##     set_random_seed, shape

## The following object is masked from 'package:caret':
##
##     train
```

Appendix 6.1: Resplit The Data

```
set.seed(100)# For reproducibility

# Randomly select 80% of the row as the split
split <- sample(1:nrow(health), 0.8 * nrow(health))
train_data <- health[split, ]  # Use the 80% selected for training
test_data <- health[-split, ]  # Use the remaining for testing
```

Appendix 6.2: Reprocessing Data

```
# Separate predictors and response variables
# Using training data
x_train <- model.matrix(Admission.Type ~ ., data = train_data)[, -1] # Rem
ove the intercept

# Using test data
x_test <- model.matrix(Admission.Type ~ ., data = test_data)[, -1] # Remov
e the intercept
```

Appendix 6.3: Scale Predictors

```
# Standardising independent variables x
x_train <- scale(x_train)
x_test  <- scale(
  x_test,
  center = attr(x_train, "scaled:center"), # Mean of each column
  scale  = attr(x_train, "scaled:scale") # Standard deviation of each colu
mn
)

# Defining admission type as the response variable y
y_train <- train_data$Admission.Type
y_test  <- test_data$Admission.Type

# Converting training labels to categorical
y_train_cat <- tf$keras$utils$to_categorical(as.integer(y_train) - 1) # -1
 to ensure classes starts at 0

# Converting test labels to categorical
y_test_cat <- tf$keras$utils$to_categorical(as.integer(y_test) - 1) # -1 t
o ensure classes starts at 0

num_classes <- ncol(y_train_cat) # Gives the number of unique classes
```

Appendix 6.4: Define Neural Network

```
model <- keras_model_sequential() %>%
  # Add first layer
  layer_dense(units = 32, activation = "relu") %>%

  # Dropout after the first layer to prevent overfitting (drop 30% of neur
ons)
  layer_dropout(rate = 0.3) %>%

  # Add second layer
  layer_dense(units = 16, activation = "relu") %>%

  # Dropout after second layer to prevent overfitting in deeper layers (dr
op 30% of neurons)
  layer_dropout(rate = 0.3) %>%

  # Output layer (final classification layer)
  layer_dense(units = num_classes, activation = "softmax")
```

Appendix 6.7: Compile And Train Model

```
# Compile model
model %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001), # Use Adam optimiser
to adapt learning rates automatically for faster and stable training
  loss = "categorical_crossentropy", # Measures the difference between pre
dicted probabilities and true labels
  metrics = "accuracy" # Tracks accuracy
)

# Train model
history <- model %>% fit(
  x_train, y_train_cat,
  validation_split = 0.2, # Reserve 20% of training data for validation to
 monitor overfitting
  epochs = 50, # Number of full passes throung the training data
  batch_size = 32 # Number of samples processed before updating the weight
s
)

## Epoch 1/50
## 804/804 - 2s - 3ms/step - accuracy: 0.3332 - loss: 1.1181 - val_accurac
y: 0.3340 - val_loss: 1.0986
## Epoch 2/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3460 - loss: 1.0994 - val_accurac
y: 0.3372 - val_loss: 1.0985
## Epoch 3/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3362 - loss: 1.0992 - val_accurac
y: 0.3368 - val_loss: 1.0985
## Epoch 4/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3416 - loss: 1.0986 - val_accurac
y: 0.3414 - val_loss: 1.0985
## Epoch 5/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3396 - loss: 1.0985 - val_accurac
```

```
y: 0.3341 - val_loss: 1.0987
## Epoch 6/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3374 - loss: 1.0986 - val_accurac
y: 0.3301 - val_loss: 1.0988
## Epoch 7/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3438 - loss: 1.0983 - val_accurac
y: 0.3383 - val_loss: 1.0987
## Epoch 8/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3444 - loss: 1.0979 - val_accurac
y: 0.3360 - val_loss: 1.0986
## Epoch 9/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3446 - loss: 1.0977 - val_accurac
y: 0.3357 - val_loss: 1.0986
## Epoch 10/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3439 - loss: 1.0979 - val_accurac
y: 0.3385 - val_loss: 1.0985
## Epoch 11/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3495 - loss: 1.0976 - val_accurac
y: 0.3410 - val_loss: 1.0986
## Epoch 12/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3502 - loss: 1.0971 - val_accurac
y: 0.3403 - val_loss: 1.0991
## Epoch 13/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3518 - loss: 1.0972 - val_accurac
y: 0.3371 - val_loss: 1.0987
## Epoch 14/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3538 - loss: 1.0968 - val_accurac
y: 0.3363 - val_loss: 1.0988
## Epoch 15/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3534 - loss: 1.0966 - val_accurac
y: 0.3371 - val_loss: 1.0987
## Epoch 16/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3546 - loss: 1.0967 - val_accurac
y: 0.3382 - val_loss: 1.0989
## Epoch 17/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3573 - loss: 1.0970 - val_accurac
y: 0.3430 - val_loss: 1.0987
## Epoch 18/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3556 - loss: 1.0964 - val_accurac
y: 0.3394 - val_loss: 1.0987
## Epoch 19/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3529 - loss: 1.0963 - val_accurac
y: 0.3413 - val_loss: 1.0988
## Epoch 20/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3527 - loss: 1.0959 - val_accurac
y: 0.3427 - val_loss: 1.0989
## Epoch 21/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3526 - loss: 1.0959 - val_accurac
y: 0.3428 - val_loss: 1.0990
## Epoch 22/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3563 - loss: 1.0951 - val_accurac
y: 0.3397 - val_loss: 1.0995
## Epoch 23/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3596 - loss: 1.0947 - val_accurac
```

```
y: 0.3385 - val_loss: 1.0994
## Epoch 24/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3606 - loss: 1.0952 - val_accurac
y: 0.3341 - val_loss: 1.0994
## Epoch 25/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3560 - loss: 1.0951 - val_accurac
y: 0.3394 - val_loss: 1.1000
## Epoch 26/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3608 - loss: 1.0950 - val_accurac
y: 0.3357 - val_loss: 1.0994
## Epoch 27/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3617 - loss: 1.0952 - val_accurac
y: 0.3378 - val_loss: 1.0995
## Epoch 28/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3605 - loss: 1.0943 - val_accurac
y: 0.3455 - val_loss: 1.0997
## Epoch 29/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3563 - loss: 1.0953 - val_accurac
y: 0.3396 - val_loss: 1.0996
## Epoch 30/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3633 - loss: 1.0949 - val_accurac
y: 0.3361 - val_loss: 1.1000
## Epoch 31/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3624 - loss: 1.0943 - val_accurac
y: 0.3405 - val_loss: 1.1002
## Epoch 32/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3605 - loss: 1.0939 - val_accurac
y: 0.3461 - val_loss: 1.1002
## Epoch 33/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3620 - loss: 1.0944 - val_accurac
y: 0.3378 - val_loss: 1.1004
## Epoch 34/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3597 - loss: 1.0945 - val_accurac
y: 0.3386 - val_loss: 1.1005
## Epoch 35/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3658 - loss: 1.0935 - val_accurac
y: 0.3419 - val_loss: 1.1005
## Epoch 36/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3662 - loss: 1.0935 - val_accurac
y: 0.3427 - val_loss: 1.1011
## Epoch 37/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3584 - loss: 1.0949 - val_accurac
y: 0.3364 - val_loss: 1.1010
## Epoch 38/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3630 - loss: 1.0933 - val_accurac
y: 0.3382 - val_loss: 1.1014
## Epoch 39/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3679 - loss: 1.0938 - val_accurac
y: 0.3344 - val_loss: 1.1001
## Epoch 40/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3654 - loss: 1.0933 - val_accurac
y: 0.3294 - val_loss: 1.1010
## Epoch 41/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3641 - loss: 1.0931 - val_accurac
```

```
y: 0.3282 - val_loss: 1.1010
## Epoch 42/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3618 - loss: 1.0932 - val_accurac
y: 0.3350 - val_loss: 1.1013
## Epoch 43/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3649 - loss: 1.0939 - val_accurac
y: 0.3330 - val_loss: 1.1013
## Epoch 44/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3671 - loss: 1.0929 - val_accurac
y: 0.3396 - val_loss: 1.1015
## Epoch 45/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3602 - loss: 1.0944 - val_accurac
y: 0.3388 - val_loss: 1.1008
## Epoch 46/50
## 804/804 - 1s - 2ms/step - accuracy: 0.3656 - loss: 1.0933 - val_accurac
y: 0.3389 - val_loss: 1.1007
## Epoch 47/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3675 - loss: 1.0930 - val_accurac
y: 0.3343 - val_loss: 1.1010
## Epoch 48/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3661 - loss: 1.0930 - val_accurac
y: 0.3349 - val_loss: 1.1012
## Epoch 49/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3653 - loss: 1.0934 - val_accurac
y: 0.3310 - val_loss: 1.1014
## Epoch 50/50
## 804/804 - 1s - 1ms/step - accuracy: 0.3662 - loss: 1.0927 - val_accurac
y: 0.3333 - val_loss: 1.1011
```

Analysis (Accuracy plot - Top): - Training accuracy hovers around 33% to 35% - Validation accuracy is similar to training accuracy but is slightly lower with both curves being mostly flat and only have a slight upward drift - Hence, model is not overfitting as trainig and validation accuracy are very close - Shows that the model is not much better than random guessing (3 classes of admission types, so about 33.3% if randomly guess) - So, neural network is not learning meaningful patterns

Analysis (Loss plot - Bottom): - Training loss drops sharply initially then slowly decreases - Validation loss stays almost the same at around 1.1 - Other than the initial gap, most of the gap between the training and validation loss is small - This indicates that initial learning does happen, but the model quickly levels - Though since there is no divergence between the training and validation loss, there unlikely to have any overfitting

Appendix 6.8: Evaluating Deep Learning Model On Test Data

```
model %>% evaluate(x_test, y_test_cat)

## 251/251 - 0s - 1ms/step - accuracy: 0.3375 - loss: 1.0998

## $accuracy
## [1] 0.3375249
##
## $loss
## [1] 1.099841
```

```
log(3)

## [1] 1.098612
```

Analysis: - Neural network shows an accuracy of about 33.5%, which means it correctly predicts about 33.5% of cases in the test set (close to random guessing) - For the 3 classes, a completely random guess would give loss of 1.099, which is about log(3). This indicates that the model is not learning useful patterns from the data

Appendix 5.6.9: Prediction And Evaluation

```
# Predict probabilities
pred_prob <- model %>% predict(x_test)

## 251/251 - 0s - 840us/step

# Convert probabilities into class indices starting at 0
pred_class <- apply(pred_prob, 1, which.max) - 1

# Convert indices to factors
pred_class <- factor(
  levels(y_test)[pred_class + 1],
  levels = levels(y_test)
)

# Overall evaluation
confusionMatrix(pred_class, y_test)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Elective Emergency Urgent
##   Elective       724       710    767
##   Emergency      324       284    300
##   Urgent        1616      1604   1703
##
## Overall Statistics
##
##                Accuracy : 0.3375
##                  95% CI : (0.3272, 0.348)
##     No Information Rate : 0.3449
##     P-Value [Acc > NIR] : 0.9189
##
##                   Kappa : -0.002
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: Elective Class: Emergency Class: Urgent
## Sensitivity                  0.27177          0.10931        0.6148
## Specificity                  0.72485          0.88517        0.3881
## Pos Pred Value               0.32894          0.31278        0.3459
## Neg Pred Value               0.66730          0.67518        0.6568
## Prevalence                     0.33167          0.32346         0.3449
## Detection Rate                 0.09014          0.03536         0.2120
## Detection Prevalence           0.27403          0.11305         0.6129
## Balanced Accuracy              0.49831          0.49724         0.5014
```

Analysis: - Overall accuracy is about 33.5% - Kappa gives 0.0092, means that neural network is not any better than random guessing - Though model does predict emergency more often as it is more sensitive, but precision of guessing is low, which may lead to missclassifications