CYO Project - Disease Prediction (Capstone)

Juan Canon

2023-04-10

Executive Summary

In this report, we present a comprehensive analysis of a dataset containing symptoms and diagnoses for 42 diseases. The dataset, sourced from two CSV files for training and testing purposes, comprises 133 columns. The first 132 columns correspond to symptoms experienced by patients, while the final column represents the prognosis. This dataset was obtained from Kaggle, a popular online platform that hosts data science competitions and provides a vast repository of datasets and tools for machine learning and analytics.

Given the increasing need for efficient and accurate disease diagnosis, the primary goal of this project is to develop a model that predicts prognosis based on a patient's reported symptoms. To achieve this, we first conducted a thorough exploration of the dataset, utilizing various visualization techniques to gain a deeper understanding of the relationships between variables. Subsequently, we evaluated four distinct machine learning models: Decision Tree, Random Forest, k-Nearest Neighbors (kNN), and Naive Bayes.

A reliable disease prediction model can offer several advantages to healthcare professionals, such as:

Enhancing the speed and accuracy of diagnosis, reducing misdiagnoses and the need for unnecessary testing. Facilitating early detection and intervention, which can improve patient outcomes and potentially lower treatment costs. Assisting in decision-making by providing an additional source of information to support clinical judgment. Helping to prioritize patients based on the severity of their conditions, enabling more efficient allocation of medical resources. Our analysis culminated in the selection of a model that demonstrated optimal performance in predicting the prognosis of the 42 diseases under consideration. The findings of this report highlight the potential of machine learning algorithms in revolutionizing the field of medical diagnostics, ultimately leading to better patient care and healthcare system efficiency.

Analysis

First Glance at the Data

Before diving into the modeling phase, it is essential to gain a deeper understanding of the dataset's characteristics and quality. In this section, we will explore the dataset by examining its structure, variable types, and any potential issues such as missing values. This initial exploration will provide valuable insights into the relationships between variables and help guide the selection of appropriate machine learning models for the disease prediction task.

```r
# Load the CSV files into data frames
training <- read_csv("Training.csv")
```

```
## New names:
## Rows: 4920 Columns: 134
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): prognosis dbl (132): itching, skin_rash, nodal_skin_eruptions,
## continuous_sneezing, sh... lgl (1): ...134
## i Use ‘spec()‘ to retrieve the full column specification for this data. i
## Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
## * ‘fluid_overload‘ -> ‘fluid_overload...46‘
## * ‘fluid_overload‘ -> ‘fluid_overload...118‘
## * ‘‘ -> ‘...134‘
```

```
testing <- read_csv("Testing.csv")
```

```
## New names:
## Rows: 42 Columns: 133
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): prognosis dbl (132): itching, skin_rash, nodal_skin_eruptions,
## continuous_sneezing, sh...
## i Use ‘spec()‘ to retrieve the full column specification for this data. i
## Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
## * ‘fluid_overload‘ -> ‘fluid_overload...46‘
## * ‘fluid_overload‘ -> ‘fluid_overload...118‘
```

```
#have a first look into the data.
head(training)
```

```
## # A tibble: 6 x 134
##   itching skin_~1 nodal~2 conti~3 shive~4 chills joint~5 stoma~6 acidity ulcer~7
##     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1       1       1       1       0       0      0       0       0       0       0
## 2       0       1       1       0       0      0       0       0       0       0
## 3       1       0       1       0       0      0       0       0       0       0
## 4       1       1       0       0       0      0       0       0       0       0
## 5       1       1       1       0       0      0       0       0       0       0
## 6       0       1       1       0       0      0       0       0       0       0
## # ... with 124 more variables: muscle_wasting <dbl>, vomiting <dbl>,
## #   burning_micturition <dbl>, ‘spotting_ urination‘ <dbl>, fatigue <dbl>,
## #   weight_gain <dbl>, anxiety <dbl>, cold_hands_and_feets <dbl>,
## #   mood_swings <dbl>, weight_loss <dbl>, restlessness <dbl>, lethargy <dbl>,
## #   patches_in_throat <dbl>, irregular_sugar_level <dbl>, cough <dbl>,
## #   high_fever <dbl>, sunken_eyes <dbl>, breathlessness <dbl>, sweating <dbl>,
## #   dehydration <dbl>, indigestion <dbl>, headache <dbl>, ...
```

```
# First 5 column names
colnames(training)[1:5]
```

```
## [1] "itching"              "skin_rash"             "nodal_skin_eruptions"
## [4] "continuous_sneezing"  "shivering"
```

```
# Structure of the first 5 columns
str(training[, 1:5])
```

```
## tibble [4,920 x 5] (S3: tbl_df/tbl/data.frame)
##  $ itching             : num [1:4920] 1 0 1 1 1 0 1 1 1 1 ...
##  $ skin_rash           : num [1:4920] 1 1 0 1 1 1 0 1 1 1 ...
##  $ nodal_skin_eruptions: num [1:4920] 1 1 1 0 1 1 1 0 1 1 ...
##  $ continuous_sneezing : num [1:4920] 0 0 0 0 0 0 0 0 0 0 ...
##  $ shivering           : num [1:4920] 0 0 0 0 0 0 0 0 0 0 ...
```

```
# Summary statistics of the first 5 variables
summary(training[, 1:5])
```

```
##     itching          skin_rash      nodal_skin_eruptions continuous_sneezing
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000      Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000      1st Qu.:0.00000
##  Median :0.0000   Median :0.0000   Median :0.00000      Median :0.00000
##  Mean   :0.1378   Mean   :0.1598   Mean   :0.02195      Mean   :0.04512
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000      3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000      Max.   :1.00000
##    shivering
##  Min.   :0.00000
##  1st Qu.:0.00000
##  Median :0.00000
##  Mean   :0.02195
##  3rd Qu.:0.00000
##  Max.   :1.00000
```

```
# Check for missing values
cat("Sum of Missing values in each column:\n")
```

```
## Sum of Missing values in each column:
```

```
sum(colSums(is.na(training)))
```

```
## [1] 4920
```

```
training[, 134]
```

```
## # A tibble: 4,920 x 1
##    ...134
##    <lgl>
##  1 NA
##  2 NA
##  3 NA
##  4 NA
##  5 NA
##  6 NA
##  7 NA
##  8 NA
##  9 NA
## 10 NA
## # ... with 4,910 more rows
```

```r
training <- training %>% select(-134)

# Select first 300 values of prognosis column and create frequency table
table(training$prognosis[1:300])
```

```
##
##                             AIDS          Alcoholic hepatitis
##                               10                           10
##                          Allergy             Bronchial Asthma
##                               10                           10
##             Cervical spondylosis                  Chicken pox
##                               10                           10
##              Chronic cholestasis                  Common Cold
##                               10                           10
##                           Dengue                     Diabetes
##                               10                           10
## Dimorphic hemmorhoids(piles)                    Drug Reaction
##                               10                           10
##                 Fungal infection             Gastroenteritis
##                               10                           10
##                             GERD                 Heart attack
##                               10                           10
##                      hepatitis A                  Hepatitis B
##                               10                           10
##                      Hepatitis C                  Hepatitis D
##                               10                           10
##                      Hepatitis E                 Hypertension
##                               10                           10
##                         Jaundice                      Malaria
##                               10                           10
##                         Migraine Paralysis (brain hemorrhage)
##                               10                           10
##              Peptic ulcer diseae                    Pneumonia
##                               10                           10
##                     Tuberculosis                      Typhoid
##                               10                           10
```

```r
length(unique(training$prognosis))
```

```
## [1] 41
```

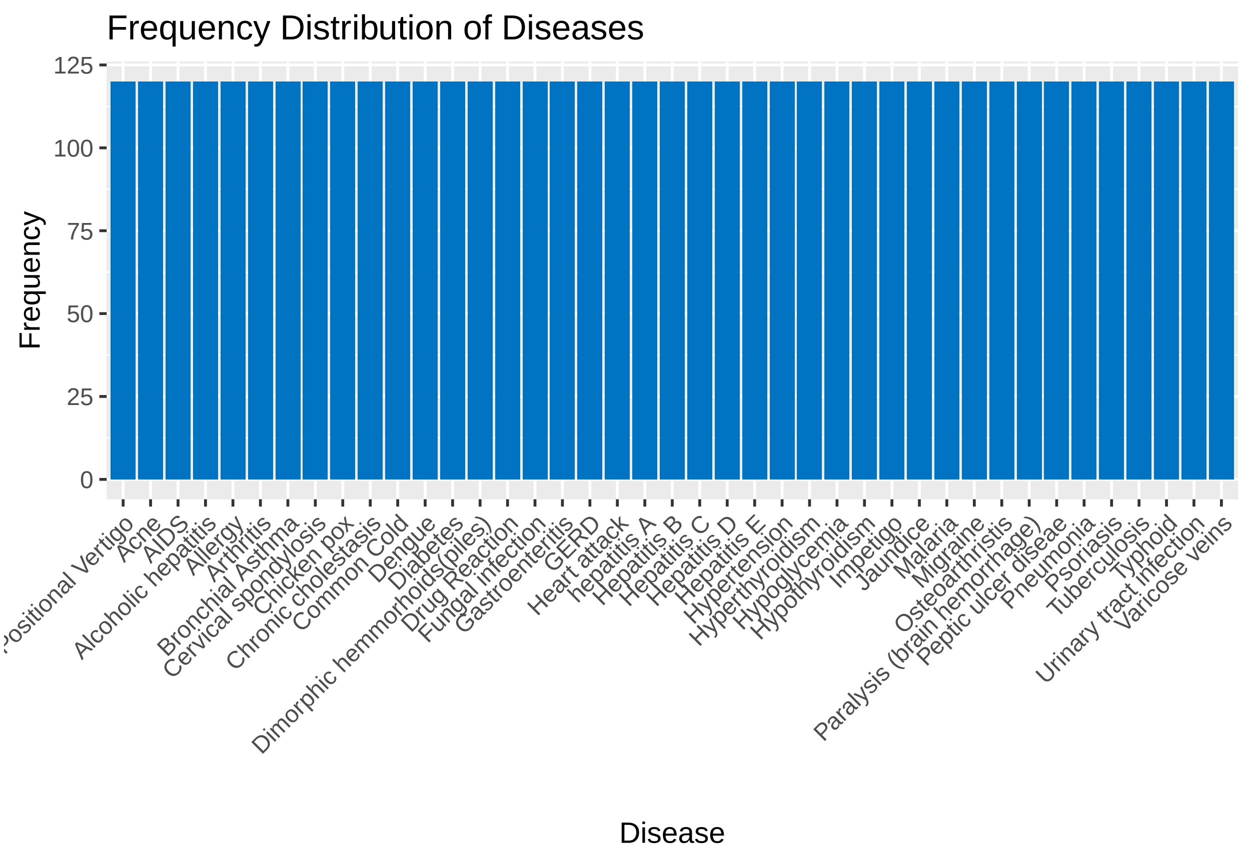
Our preliminary data analysis revealed that the dataset is well-structured and primarily comprises binary variables representing the presence or absence of symptoms. We discovered that the last column of the dataset contained only missing values (NA), which we subsequently removed from our analysis. The remaining variables were found to be reasonably well-distributed, with 41 unique disease diagnoses present in the training dataset.

Notably, the dataset appears to be well-balanced, with each disease having an equal number of observations. This suggests that the creators of the dataset made a deliberate effort to maintain a balanced representation of each disease. This initial exploration of the data confirms that it is suitable for further analysis and application of machine learning models for predicting disease prognosis.

Data Inspection and Visualization

We start by displaying the frequency distribution of diseases in the dataset, allowing us to quickly identify the most and least common diseases, providing insight into the relative prevalence of each condition.

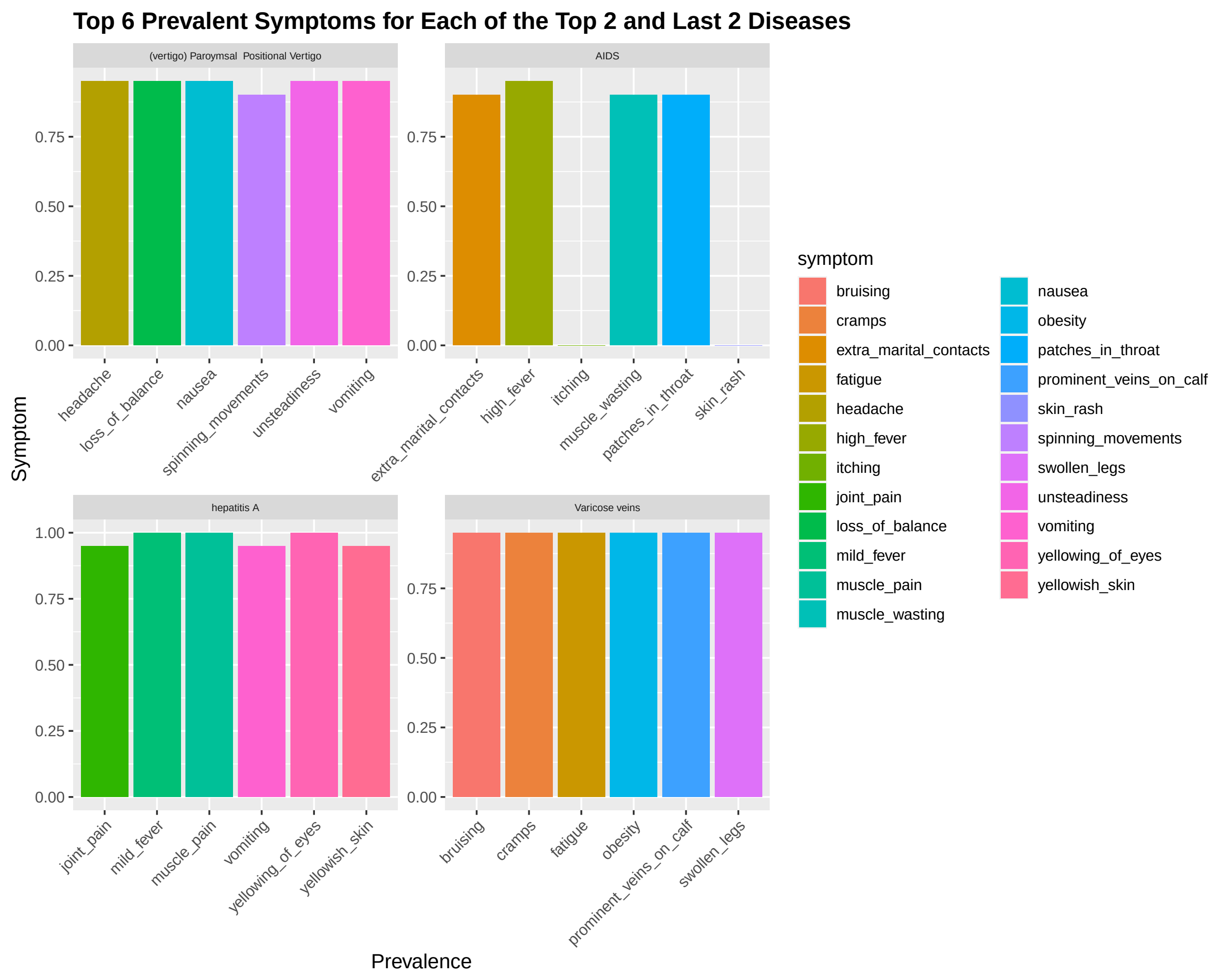
```r
#Visualization-----------------------------------------------------------

# Visualize the frequency distribution of diseases
#Note: Axis angle is adjusted to make it easier to read.
ggplot(training, aes(x=prognosis)) +
  geom_bar(fill = "#0073C2") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title="Frequency Distribution of Diseases", x="Disease", y="Frequency")
```



As suspected, the dataset has been created in a way that each prognosis has an equal amount of observations. This is a beneficial characteristic for model creation, as it helps to ensure that each disease category is equally represented in the training data. This can help to prevent the model from being biased towards more frequently occurring diseases, and can increase the model's accuracy and generalizability to new data.

Next we showcase the top 6 prevalent symptoms for each of the top 2 and last 2 diseases, offering insights into their clinical presentations and enabling us to better understand their characteristic symptoms.

```r
#Most prevalent symptoms for some diseases----------------------------------

# Get top 2 and last 2 diseases
top_diseases <- training %>%
  count(prognosis, sort = TRUE) %>%
  slice(c(head(row_number(), 2), tail(row_number(), 2))) %>%
  pull(prognosis)

# Filter the tibble for the top 2 and last 2 diseases
filtered_tibble <- training %>%
  filter(prognosis %in% top_diseases)

# Calculate the prevalence of each symptom for each disease
symptom_prevalence <- filtered_tibble %>%
  group_by(prognosis) %>%
  summarise(across(itching:blackheads, \(x) mean(x, na.rm = TRUE))) %>%
  pivot_longer(cols = -prognosis, names_to = "symptom", values_to = "prevalence")

# Get top 6 prevalent symptoms for each disease
top_symptoms_per_disease <- symptom_prevalence %>%
  group_by(prognosis) %>%
  arrange(prognosis, desc(prevalence)) %>%
  slice_head(n = 6)

# Visualize the top 6 prevalent symptoms
#for each of the top 2 and last 2 diseases using facet_wrap
top_symptoms_per_disease %>%
  ggplot(aes(x = symptom, y = prevalence, fill = symptom)) +
  geom_col() +
  facet_wrap(~prognosis, scales = "free", ncol = 2) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        plot.title = element_text(size = 14, face = "bold"),
        axis.title.x = element_text(size = 12),
        axis.title.y = element_text(size = 12), strip.text = element_text(size = 6)) +
  labs(title = "Top 6 Prevalent Symptoms for Each of the Top 2 and Last 2 Diseases",
       x = "Prevalence",
       y = "Symptom")
```



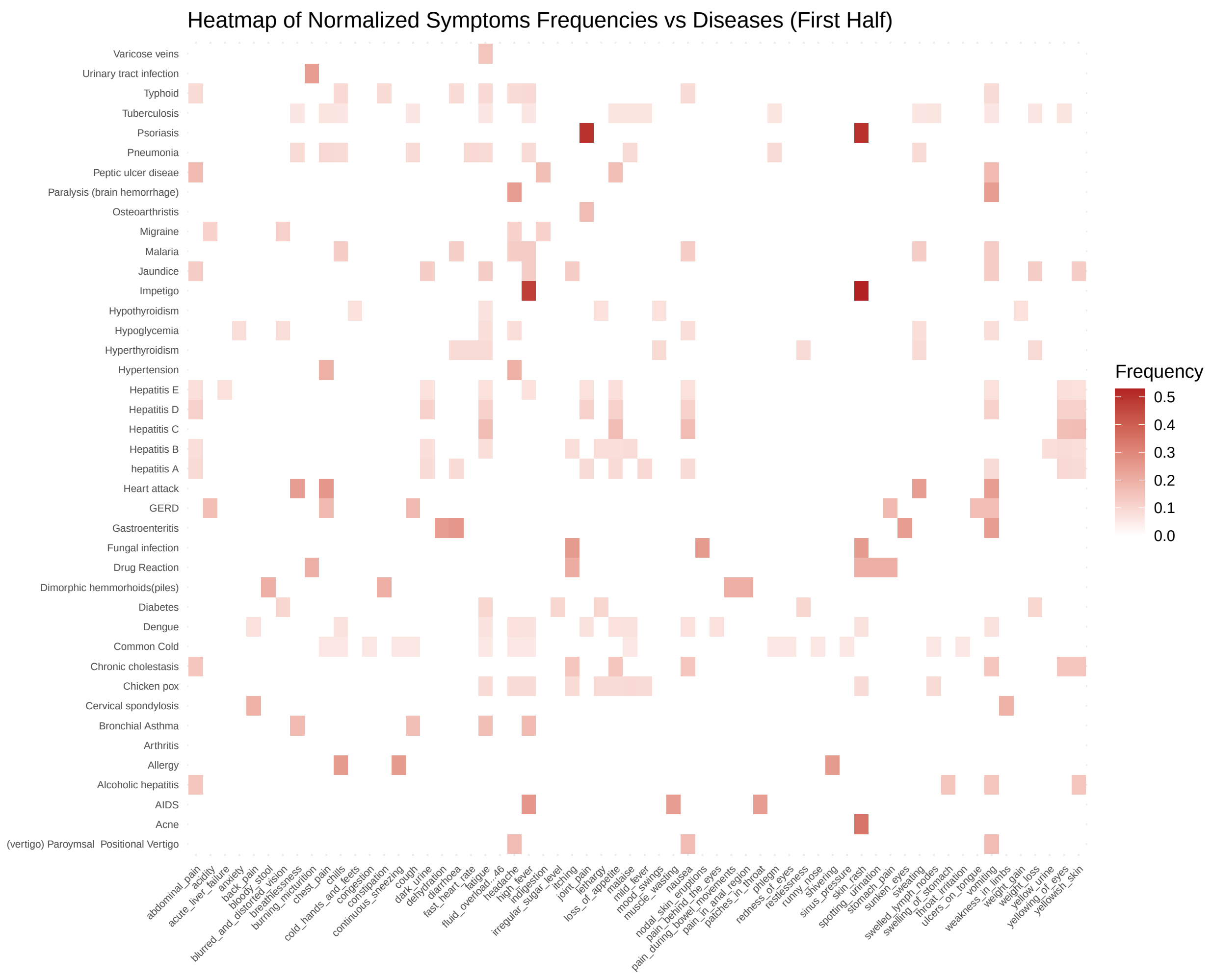
In general, we observe that the dataset exhibits a clear distinction between the presence or absence of symptoms for each disease. This suggests that there is little ambiguity in the data, which could potentially facilitate the development of a predictive model. The distinct symptom profiles for each disease may allow the model to accurately distinguish between various conditions, resulting in a more reliable disease prediction.

Heatmap of Symptoms by Disease

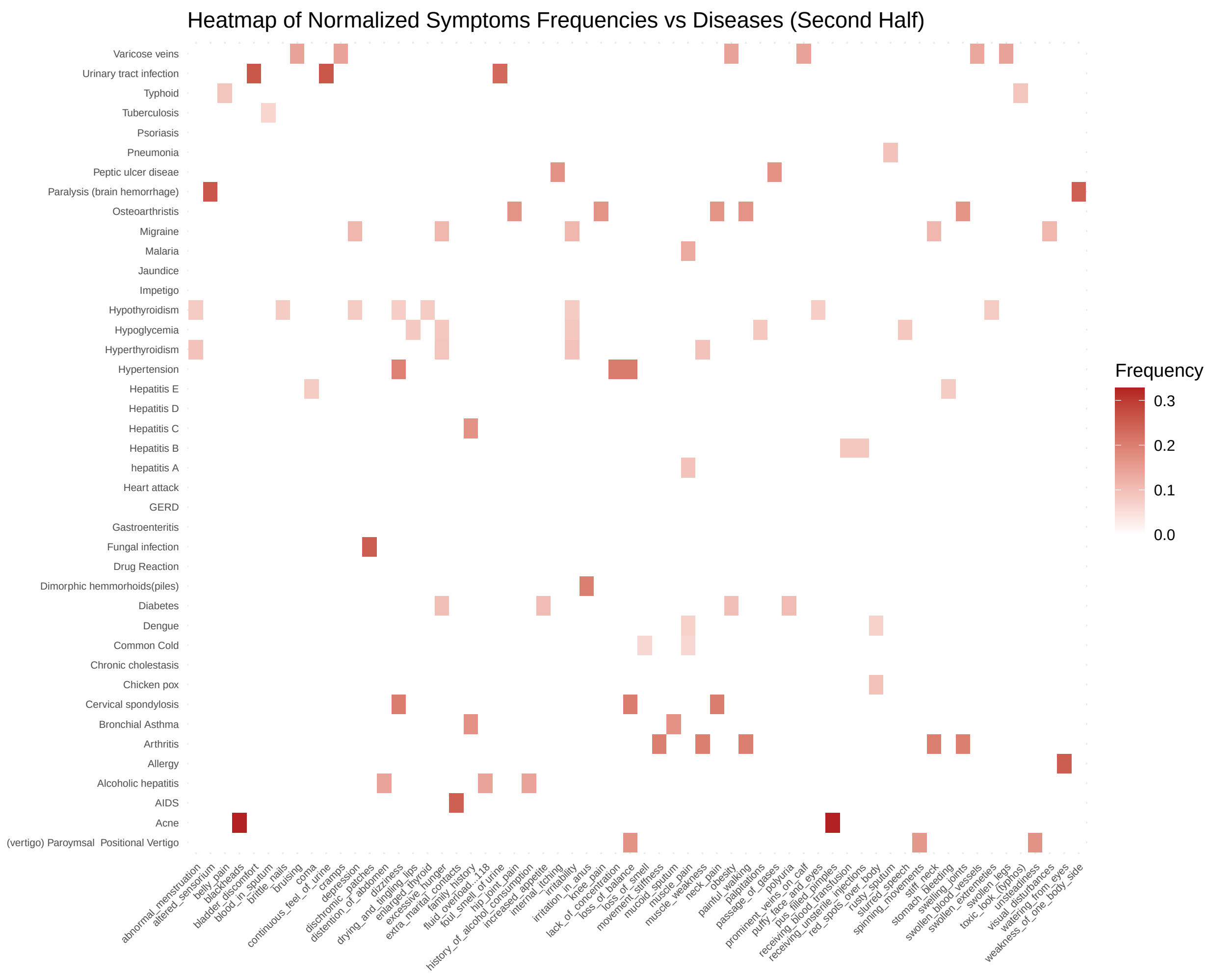
In order to create the heatmaps, we first aggregate the data to calculate the frequency of each symptom for each disease (using mainly “dplyr” tools). Then, we normalize the frequency within each disease to account for varying disease prevalence in the dataset. To visualize the symptoms more effectively, we split the symptoms into two halves and filter the data accordingly, creating two separate heatmaps that represent the normalized symptom frequencies for each disease.

```r
# Aggregate the data to calculate the frequency of each symptom for each disease
symptoms_vs_diseases <- training %>%
  group_by(prognosis) %>%
  summarise(across(itching:blackheads, sum)) %>%
  pivot_longer(cols = -prognosis, names_to = "Symptom", values_to = "Frequency") %>%
  group_by(prognosis) %>%
  mutate(Frequency = Frequency/sum(Frequency))
```

```r
# Split the symptoms into two halves
symptoms <- unique(symptoms_vs_diseases$Symptom)
half <- floor(length(symptoms) / 2)
first_half_symptoms <- symptoms[1:half]
second_half_symptoms <- symptoms[(half + 1):length(symptoms)]

# Filter the data for the first half of the symptoms
first_half_data <- symptoms_vs_diseases %>% filter(Symptom %in% first_half_symptoms)

# Create the heatmap for the first half of the symptoms
heatmap_firsHalf <- ggplot(first_half_data, aes(x = Symptom,
                                    y = prognosis, fill = Frequency)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "firebrick") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 6),
        axis.text.y = element_text(size = 6),
        axis.title.x = element_blank(),
        axis.title.y = element_blank()) +
  labs(title = "Heatmap of Normalized Symptoms Frequencies vs Diseases (First Half)",
       fill = "Frequency")
heatmap_firsHalf
```



```r
# Filter the data for the second half of the symptoms
second_half_data <- symptoms_vs_diseases %>% filter(Symptom %in% second_half_symptoms)

# Create the heatmap for the second half of the symptoms
heatmap_secondHalf <- ggplot(second_half_data, aes(x = Symptom,
                                                   y = prognosis,
                                                   fill = Frequency)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "firebrick") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 6),
        axis.text.y = element_text(size = 6),
        axis.title.x = element_blank(),
        axis.title.y = element_blank()) +
  labs(title = "Heatmap of Normalized Symptoms Frequencies vs Diseases (Second Half)",
       fill = "Frequency")

heatmap_secondHalf
```



Upon examining the heatmaps, we notice that most diseases are associated with a relatively small number of symptoms, which suggests that the dataset is well-defined and specific. There are some exceptions, such as the common cold, hepatitis, and tuberculosis, which exhibit a higher number of associated symptoms. These diseases may require further analysis to understand the underlying reasons for their increased symptom diversity and the potential implications on diagnostic accuracy.

Multicollinearity Heatmap of Symptoms

Creating a collinearity heatmap of symptoms helps us identify any potential redundancies or strong correlations between symptoms, which can have implications for the accuracy and efficiency of our diagnostic model. By calculating the pairwise correlations between symptoms and visualizing them as a heatmap, we can quickly assess the relationships among symptoms and identify areas of high collinearity.

Please visit a short guided cited at the end of the report, which explains how to build such graphs/analysis.

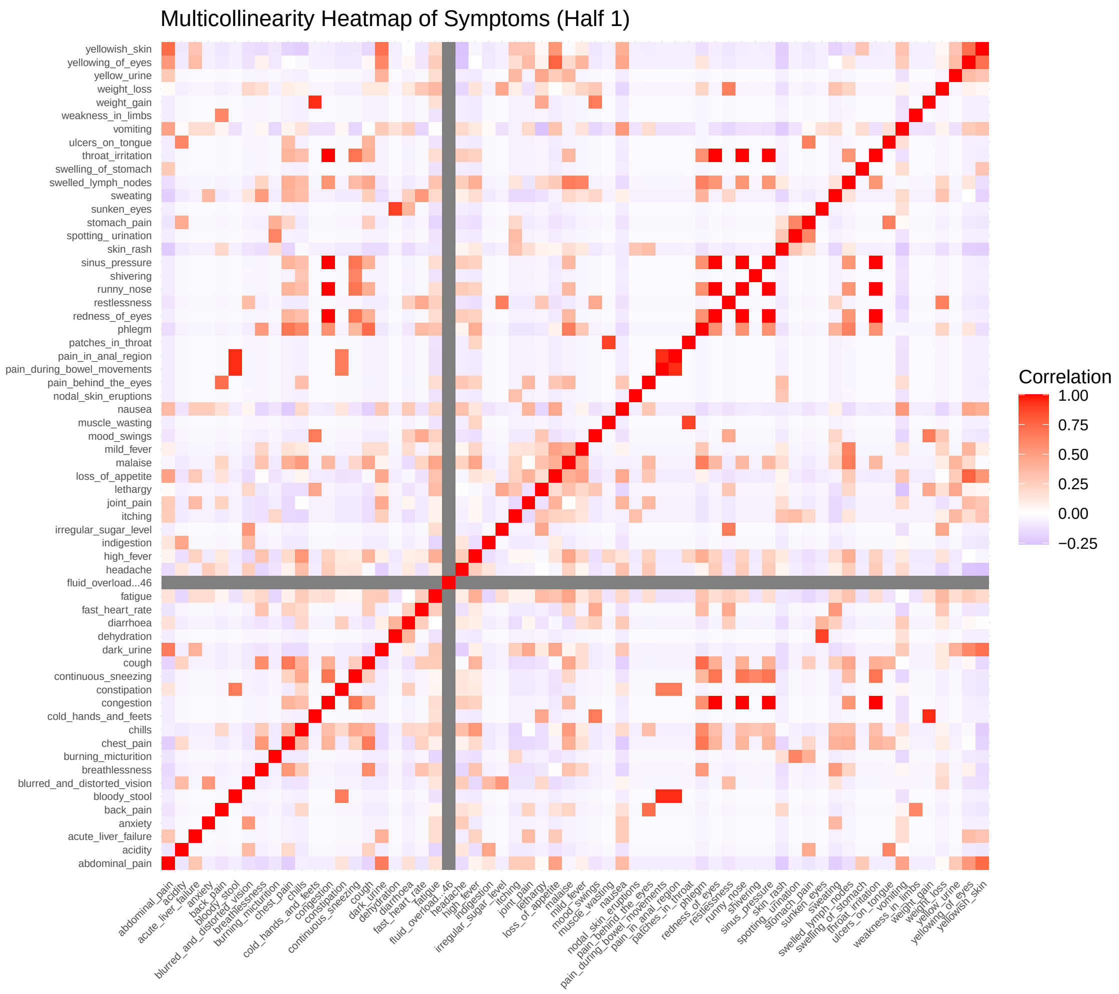
```r
#Correlation plot of the symptoms---------------------------------------------

# Select only the symptoms columns
symptoms_data <- training %>%
  select(itching:blackheads)

# Calculate the midpoint for splitting the symptoms data
```

```r
midpoint <- ceiling(ncol(symptoms_data) / 2)

# Split the data into two halves
symptoms_data_half1 <- symptoms_data[, 1:midpoint]
symptoms_data_half2 <- symptoms_data[, (midpoint + 1):ncol(symptoms_data)]

# Calculate the correlation matrices for each half.
#We use the cor() function from the
correlation_matrix_half1 <- cor(symptoms_data_half1)
```

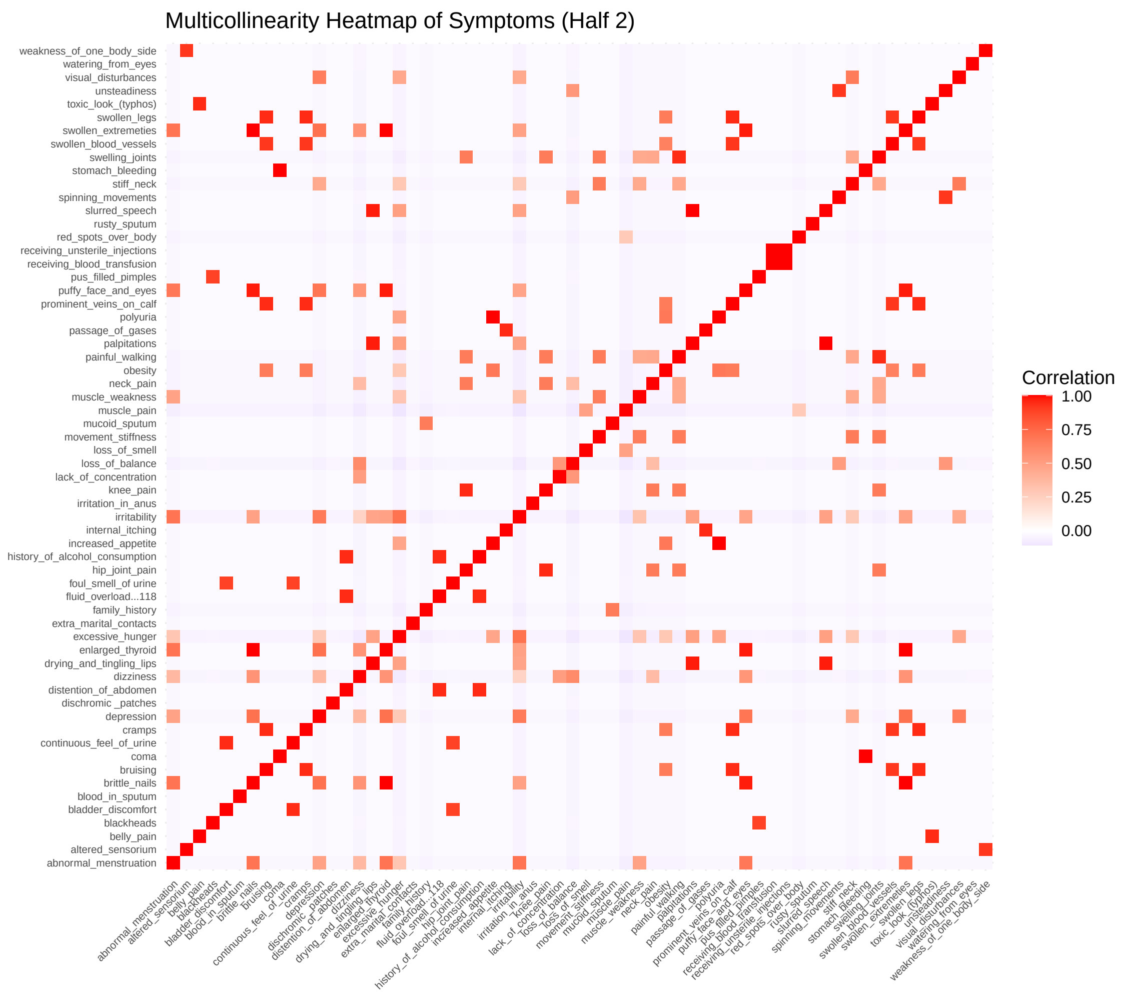
```
## Warning in cor(symptoms_data_half1): the standard deviation is zero
```

```r
correlation_matrix_half2 <- cor(symptoms_data_half2)

# Convert the correlation matrices to long format data frames
correlation_df_half1 <- correlation_matrix_half1 %>%
  as.data.frame() %>%
  rownames_to_column(var = "Variable1") %>%
  pivot_longer(cols = -Variable1, names_to = "Variable2",
               values_to = "Correlation")

correlation_df_half2 <- correlation_matrix_half2 %>%
  as.data.frame() %>%
  rownames_to_column(var = "Variable1") %>%
  pivot_longer(cols = -Variable1, names_to = "Variable2",
               values_to = "Correlation")

# Create heatmaps for each half
heatmap_plot_half1 <- ggplot(correlation_df_half1, aes(x = Variable1,
                                                       y = Variable2, fill = Correlation)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red", midpoint = 0) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 6),
        axis.text.y = element_text(size = 6),
        axis.title.x = element_blank(),
        axis.title.y = element_blank()) +
  coord_fixed(ratio = 1) +
  labs(title = "Multicollinearity Heatmap of Symptoms (Half 1)",
       fill = "Correlation")

heatmap_plot_half2 <- ggplot(correlation_df_half2, aes(x = Variable1,
                                                       y = Variable2, fill = Correlation)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", mid = "white", high = "red", midpoint = 0) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 6),
        axis.text.y = element_text(size = 6),
        axis.title.x = element_blank(),
        axis.title.y = element_blank()) +
  coord_fixed(ratio = 1) +
  labs(title = "Multicollinearity Heatmap of Symptoms (Half 2)",
       fill = "Correlation")
```

```
# Display the heatmaps
heatmap_plot_half1
```



```
heatmap_plot_half2
```



Upon examining the collinearity heatmap, we observe that several symptoms appear to be somewhat redundant, suggesting that a more efficient symptom definition might be possible. For example, we find that the "fluid_overload" symptom is always 0, which provides no additional information, and we have consequently removed it from the training dataset. Another example is the close relationship between symptoms like runny nose, congestion, and sinus pressure, which may collectively provide overlapping information.

For the moment, we will leave the predictor variables as they are and proceed to evaluate the performance of our models. Identifying and addressing these redundancies can help improve the efficiency and analysis of our diagnostic model in later stages of analysis.

```r
# Remove the fluid_overload column from the training dataset
training <- training[, -46]
```

Possible Variables simplification

Leveraging the data processing conducted on the correlation matrix, we aim to simplify the variables, provided it proves beneficial. We opted to filter pairs of variables with a correlation higher than 0.9 to retain valuable information.

We primarily employed functions from the "dplyr" package for filtering and rearranging, along with for loops and conditional functions.

```r
#New train data simplifying symptoms
#combining symptoms with more than 0.9 of correlation


symptoms_correlation <- cor(symptoms_data, use = "pairwise.complete.obs")
```

```
## Warning in cor(symptoms_data, use = "pairwise.complete.obs"): the standard
## deviation is zero
```

```r
# Calculate the correlation matrix for the entire symptoms_data
correlation_matrix <- cor(symptoms_data)
```

```
## Warning in cor(symptoms_data): the standard deviation is zero
```

```r
# Find pairs of symptoms with a correlation higher than 0.9
high_correlation_pairs <- which(correlation_matrix > 0.9 & correlation_matrix < 1, arr.ind = TRUE)
high_correlation_pairs <- unique(t(apply(high_correlation_pairs, 1, sort)))

# Create an empty data frame to store the new combined features
new_features <- data.frame(matrix(ncol = 0, nrow = nrow(symptoms_data)))

# Perform PCA for each group of highly correlated symptoms and create new features
for (i in 1:nrow(high_correlation_pairs)) {
  symptoms_pair <- colnames(symptoms_data)[high_correlation_pairs[i, ]]
  symptoms_subset <- symptoms_data[, symptoms_pair]

  # Combine the correlated symptoms with an OR operation (presence of either symptom)
  combined_symptom <- symptoms_subset[, 1] | symptoms_subset[, 2]
  combined_symptom <- as.integer(combined_symptom)

  # Create new feature name based on the combined symptoms
  new_feature_name <- paste(symptoms_pair, collapse = "_")

  new_features <- cbind(new_features, setNames(list(combined_symptom), new_feature_name))
}

# Remove the original symptoms from the dataset
reduced_symptoms_data <- symptoms_data %>%
  select(-colnames(symptoms_data)[unique(c(high_correlation_pairs))])

# Add the new combined features to the dataset
reduced_symptoms_data <- cbind(reduced_symptoms_data, new_features)

# Add the prognosis column to the reduced_symptoms_data
reduced_symptoms_data$prognosis <- training$prognosis

ncol(reduced_symptoms_data)
```

```
## [1] 130
```

Upon examining the simplification, we noticed that it removed only about 4 variables, which doesn't significantly impact the model creation process. Consequently, we decided to proceed with building the models using the original dataset.

Machine Learning - Models Generation

We conducted a literature review to understand which models are commonly applied to this type of data and, in general, to disease prediction. Our investigation revealed that several models utilized during the data science program, such as ***k-nearest neighbors (kNN), decision trees, and random forests***, are applicable to this situation. These models have been extensively used and validated in the field of medical diagnosis and disease prediction. It is worth noting that our focus is not on delving into the mathematical theory behind these models; instead, we aim to apply well-established methodologies to interesting applications, leveraging their potential to gain valuable insights and make accurate predictions.

Another model, provided also within the train() function of the caret package is the ***Naive Bayes*** algorithm. The algorithm is based on the Bayes' theorem, which computes the probability of an event occurring, given certain evidence. In the context of disease prediction, this translates to calculating the probability of a particular disease given the observed symptoms.

```r
set.seed(1)
# Split the data into training (80%) and testing (20%) sets
train_indices <- createDataPartition(training$prognosis, p = 0.8,
                                     list = FALSE)
train_data <- training[train_indices, ]
test_data <- training[-train_indices, ]

# Split the data into training (80%) and testing (20%) sets
train_indices_reduced <- createDataPartition(reduced_symptoms_data$prognosis, p = 0.8,
                                             list = FALSE)
train_data_reduced <- reduced_symptoms_data[train_indices_reduced, ]
test_data_reduced <- reduced_symptoms_data[-train_indices_reduced, ]

# Convert prognosis to factor
train_data$prognosis <- factor(train_data$prognosis)
test_data$prognosis <- factor(test_data$prognosis, levels = levels(train_data$prognosis))

# Fix variable names
names(train_data) <- make.names(names(train_data), unique = TRUE)
names(test_data) <- make.names(names(test_data), unique = TRUE)
```

We start by splitting the training data into separate sets for model training and validation. Before proceeding with model creation, we need to convert the prognosis variable to a factor, ensuring that the levels in the test data match those in the train data. Additionally, we fix variable names using the make.names function to avoid any potential issues during the modeling process. All the models are created using the train() function from the caret package, which streamlines the process and allows us to efficiently compare their performance.

Now that the data is prepared, we proceed to create and validate the models. At first we use common or standard parameters for the models. Depending on the performance, we might want to iteratively tune them for a higher accuracy.

```r
# Decision Tree-----------------------------------------------------
dt_model <- rpart(prognosis ~ ., data = train_data)
dt_pred <- predict(dt_model, test_data, type = "class")
dt_cm <- confusionMatrix(dt_pred, test_data$prognosis)
dt_accuracy <- dt_cm$overall['Accuracy']

# Random Forest-----------------------------------------------------
rf_model <- randomForest(prognosis ~ ., data = train_data)
```

```r
rf_pred <- predict(rf_model, test_data)
rf_cm <- confusionMatrix(rf_pred, test_data$prognosis)
rf_accuracy <- rf_cm$overall['Accuracy']

#Navie bayes method
# Set up the training control
trControl <- trainControl(method = "cv", number = 10)
# Train Naive Bayes model using train() function
nb_model <- train(prognosis ~ ., data = train_data,
                  method = "naive_bayes", trControl = trControl)
# Predict on test set
nb_pred <- predict(nb_model, test_data)
# Compute confusion matrix
nb_cm <- confusionMatrix(nb_pred, test_data$prognosis)
nb_accuracy <- nb_cm$overall['Accuracy']

# kNN---------------------------------------------------------------------
k_values <- seq(1, 40, 1)
accuracy_results <- numeric(length(k_values))

for (i in seq_along(k_values)) {
  knn_model <- knn3(prognosis ~ ., data = train_data, k = k_values[i])
  knn_pred <- predict(knn_model, test_data, type = "class")
  knn_cm <- confusionMatrix(knn_pred, test_data$prognosis)
  accuracy_results[i] <- knn_cm$overall['Accuracy']
}
# Find the optimal k-value
optimal_k <- k_values[which.max(accuracy_results)]
print(paste("Optimal k-value:", optimal_k))
```

```
## [1] "Optimal k-value: 1"
```

```r
knn_pred <- predict(knn_model, test_data, type = "class")
knn_cm <- confusionMatrix(knn_pred, test_data$prognosis)
knn_accuracy <- knn_cm$overall['Accuracy']
```

```r
# Function to calculate the mean of specificity and sensitivity
mean_specificity_sensitivity <- function(cm) {
  specificity <- mean(cm$byClass[,"Specificity"], na.rm = TRUE)
  sensitivity <- mean(cm$byClass[,"Sensitivity"], na.rm = TRUE)
  return(c(specificity, sensitivity))
}

# Function to calculate the mean F1 score
mean_f1_score <- function(cm) {
  precision <- cm$byClass[,"Pos Pred Value"]
  recall <- cm$byClass[,"Sensitivity"]
  f1 <- 2 * (precision * recall) / (precision + recall)
  return(mean(f1, na.rm = TRUE))
}
# Results are stored in the "results table" by calling elements of the
# Confusion Matrix.
```

```r
results <- data.frame(
  Model = c("Decision Tree", "Random Forest", "Navie Bayies", "kNN"),
  Accuracy = c(dt_accuracy, rf_accuracy, nb_accuracy, knn_accuracy),
  F1_Score = c(
    mean_f1_score(dt_cm), mean_f1_score(rf_cm),
    mean_f1_score(nb_cm), mean_f1_score(knn_cm)
  ),
  Specificity = c(
    mean_specificity_sensitivity(dt_cm)[1],
    mean_specificity_sensitivity(rf_cm)[1],
    mean_specificity_sensitivity(nb_cm)[1],
    mean_specificity_sensitivity(knn_cm)[1]
  ),
  Sensitivity = c(
    mean_specificity_sensitivity(dt_cm)[2],
    mean_specificity_sensitivity(rf_cm)[2],
    mean_specificity_sensitivity(nb_cm)[2],
    mean_specificity_sensitivity(knn_cm)[2]
  )
)

print(results)
```

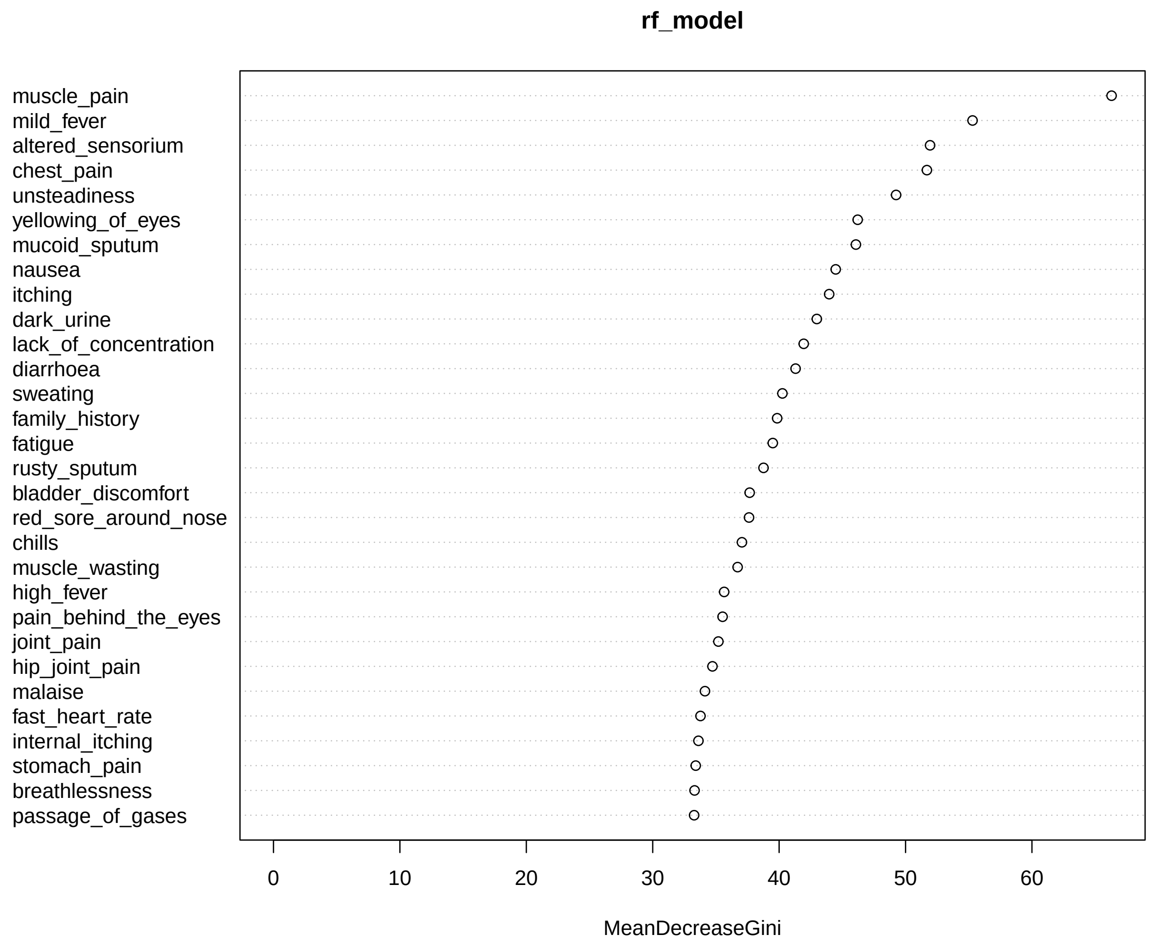
```
##           Model  Accuracy  F1_Score Specificity Sensitivity
## 1 Decision Tree 0.8445122 0.9392115   0.9961128   0.8445122
## 2 Random Forest 1.0000000 1.0000000   1.0000000   1.0000000
## 3  Navie Bayies 1.0000000 1.0000000   1.0000000   1.0000000
## 4           kNN 1.0000000 1.0000000   1.0000000   1.0000000
```

The performance of the models in this exercise was exceptional, with nearly all of them achieving perfect accuracy, except for the decision tree. This high accuracy can be attributed to the simplified nature of the dataset, which presented diseases and their corresponding symptoms as binary functions (1 or 0).

In reality, the relationship between diseases and symptoms is more complex, with symptoms often presenting in varying degrees of severity or intensity. Thus, while the models demonstrated proficiency in handling this simplified data, they may not be as effective when faced with real-world situations where the correlation between diseases and symptoms is more intricate and nuanced.

After obtaining the predictive models, we perform additional inspections to better understand their performance. One of the most insightful ways to do this is by examining the “Variable Importance” plot, which illustrates the contribution of each symptom in making accurate predictions. This plot helps identify the most significant symptoms in predicting the outcome of interest, which can aid in further refining the model and improving its accuracy.

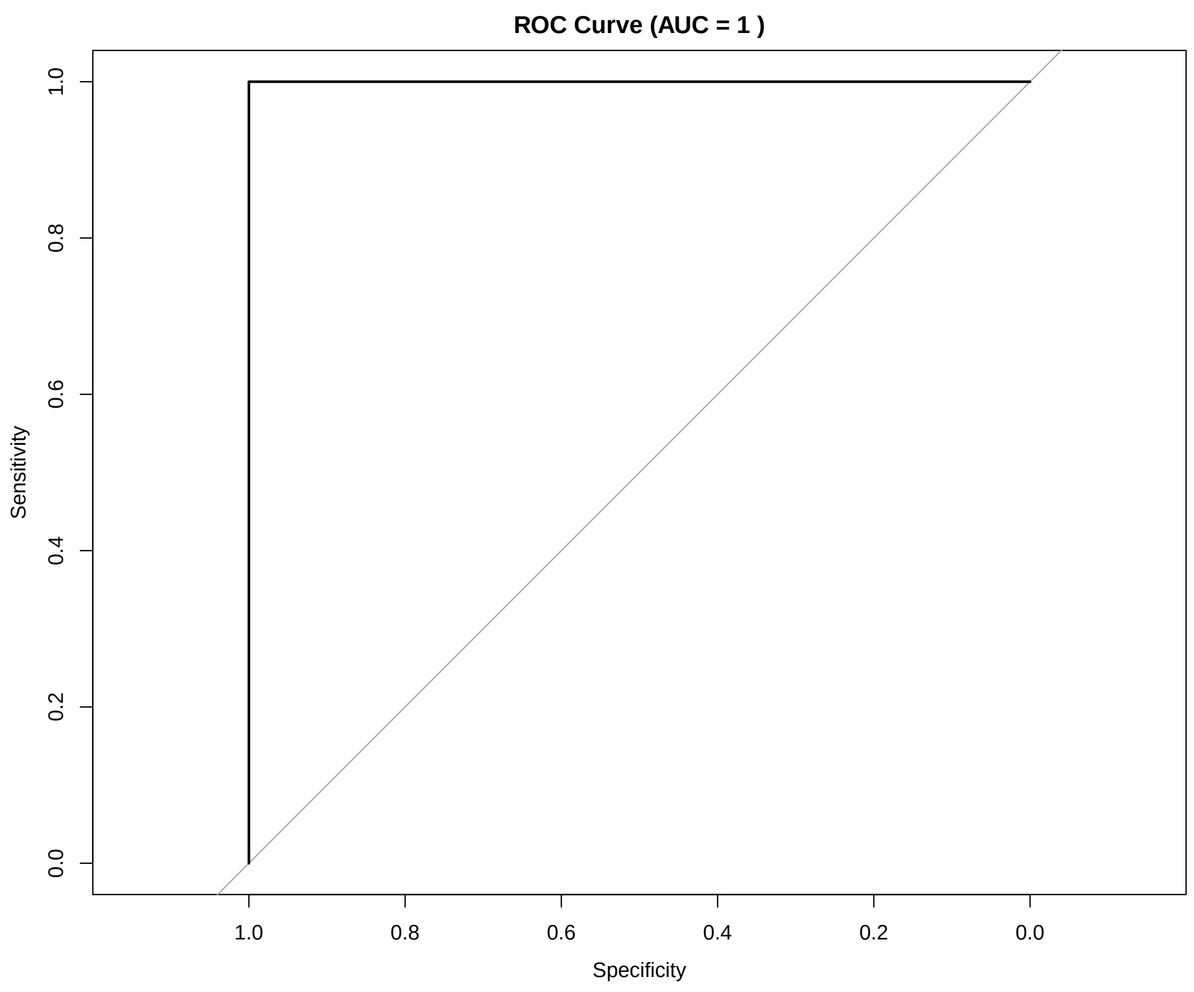
```r
# Visualize Random Forest (Variable Importance)
varImpPlot(rf_model)
```



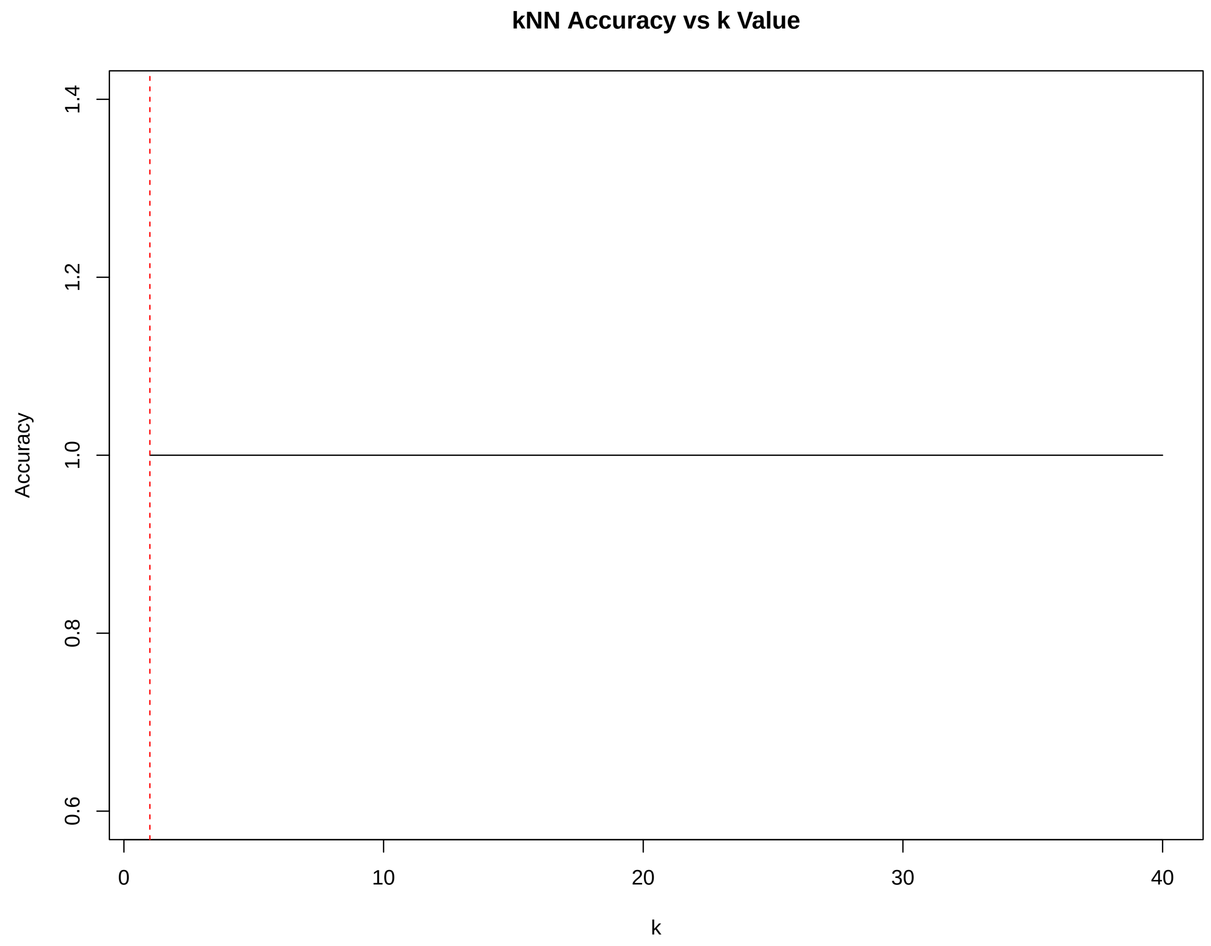
```
#ROC Plot for the rf model

# Predict probabilities for the test data
pred_probs <- predict(rf_model, newdata = test_data, type = "prob")

# Extract the probabilities for the positive class
positive_probs <- pred_probs[, 2]

# Calculate the ROC curve and AUC
roc_obj <- roc(test_data$prognosis, positive_probs)
```

```
## Setting levels: control = (vertigo) Paroymsal  Positional Vertigo, case = Acne

## Setting direction: controls < cases
```

```
auc <- auc(roc_obj)

# Plot the ROC curve
ROC_rf <- plot(roc_obj, main = paste("ROC Curve (AUC =", round(auc, 2), ")"))
```



```r
#knn visualization
knn_values_accuracy <- plot(k_values, accuracy_results, type = "l",
                            xlab = "k", ylab = "Accuracy", main = "kNN Accuracy vs k Value")
abline(v = optimal_k, col = "red", lty = 2)
```



Interestingly, some prevalent symptoms in the predictive model are also quite common, possibly due to their association with various diseases, making them useful health predictors. These symptoms may also serve as early disease indicators, enhancing the model's prediction accuracy.

For example, eye yellowing could indicate liver disease, while muscle pain and mild fever might suggest a viral infection, narrowing down potential outcomes and improving the model's predictive ability.

However, plots like the "ROC Plot" or "KNN vs k-value" offer limited information due to the model's 100% accuracy. This suggests that all k-values for the KNN model perform similarly, and the ROC plot's sensitivity and specificity measures aren't compromised.

Results

Once the models are generated, we utilize them to construct a results table based on the testing dataset. The testing data, which was provided by the Kaggle contributor, includes observations from 42 patients, each with a specific prognosis. Using this dataset, we can evaluate the performance of the generated models and determine their accuracy in predicting patient outcomes.

```
# Predict on the testing dataset

# Convert prognosis to factor
```

```r
testing$prognosis <- factor(testing$prognosis)
testing$prognosis <- factor(testing$prognosis, levels = levels(testing$prognosis))

# Fix variable names
names(testing) <- make.names(names(testing), unique = TRUE)
names(testing) <- make.names(names(testing), unique = TRUE)

dt_pred_testing <- predict(dt_model, testing, type = "class")
dt_cm_testing <- confusionMatrix(dt_pred_testing, testing$prognosis)
dt_accuracy_testing <- dt_cm_testing$overall['Accuracy']

rf_pred_testing <- predict(rf_model, testing)
rf_cm_testing <- confusionMatrix(rf_pred_testing, testing$prognosis)
rf_accuracy_testing <- rf_cm_testing$overall['Accuracy']

nb_pred_testing <- predict(nb_model, testing)
nb_cm_testing <- confusionMatrix(nb_pred_testing, testing$prognosis)
nb_accuracy_testing <- nb_cm_testing$overall['Accuracy']

knn_pred_testing <- predict(knn_model, testing, type = "class")
knn_cm_testing <- confusionMatrix(knn_pred_testing, testing$prognosis)
knn_accuracy_testing <- knn_cm_testing$overall['Accuracy']

# Create the results table for the testing dataset
results_testing <- data.frame(
  Model = c("Decision Tree", "Random Forest", "Naive Bayes", "kNN"),
  Accuracy = c(
    dt_accuracy_testing, rf_accuracy_testing,
    nb_accuracy_testing, knn_accuracy_testing
  ),
  F1_Score = c(
    mean_f1_score(dt_cm_testing), mean_f1_score(rf_cm_testing),
    mean_f1_score(nb_cm_testing), mean_f1_score(knn_cm_testing)
  ),
  Specificity = c(
    mean_specificity_sensitivity(dt_cm_testing)[1],
    mean_specificity_sensitivity(rf_cm_testing)[1],
    mean_specificity_sensitivity(nb_cm_testing)[1],
    mean_specificity_sensitivity(knn_cm_testing)[1]
  ),
  Sensitivity = c(
    mean_specificity_sensitivity(dt_cm_testing)[2],
    mean_specificity_sensitivity(rf_cm_testing)[2],
    mean_specificity_sensitivity(nb_cm_testing)[2],
    mean_specificity_sensitivity(knn_cm_testing)[2]
  )
)
print(results_testing)
```

```
##           Model  Accuracy  F1_Score Specificity Sensitivity
## 1 Decision Tree 0.8809524 0.9729730   0.9970256   0.9024390
## 2 Random Forest 0.9761905 0.9837398   0.9994051   0.9878049
## 3   Naive Bayes 1.0000000 1.0000000   1.0000000   1.0000000
## 4           kNN 1.0000000 1.0000000   1.0000000   1.0000000
```

The results indicate that the Random Forest model's performance slightly decreased on the validation data compared to the training data. While accuracy and F1-score remain high, specificity and sensitivity measures have dropped somewhat. This could be due to over fitting the training data or an insufficiently sized validation dataset. It's crucial to evaluate model performance on multiple datasets to ensure effective generalization to new data.

Taking into account that in a real-world application, specialists would likely prefer to be presented with multiple potential diagnoses, we conducted an additional analysis. By utilizing the Naive Bayes model, we generated the top 5 most probable diseases for each observation, offering a broader range of options for further patient testing and evaluation. This approach ensures that healthcare professionals have a more comprehensive understanding of possible conditions, ultimately leading to better-informed decisions in patient care.

```r
#top 5 diseases per observation using the nb_______________________________________

# Create an empty list with the same length as rand_obs to store the results
random_samples <- 5

# Randomly select observations from the training data
set.seed(123)
rand_obs <- sample(1:nrow(testing), random_samples)

# Create an empty list with the same length as random_samples to store the results
results <- vector("list", length(rand_obs))

# Loop through each random observation
for (i in 1:random_samples) {

  # Predict the probabilities of each disease given the symptoms using the Naive Bayes model
  probs <- predict(nb_model, newdata = testing[rand_obs[i],], type = "prob")

  # Sort the probabilities in descending order and select the top 5 diseases
  top_diseases <- names(sort(probs, decreasing = TRUE))[1:5]
  top_probs <- sort(probs, decreasing = TRUE)[1:5]

  # Store the results in a list
  results[[i]] <- list(top_diseases = top_diseases, top_probs = top_probs)
}

# Print the results
for (i in 1:length(results)) {
  cat("\nObservation", rand_obs[i], ":\n")
  cat("Top diseases:", results[[i]]$top_diseases, "\n")
  cat("Top probabilities:", paste(round(results[[i]]$top_probs, 14), collapse = ", "), "\n")
}
```

```
##
## Observation 31 :
## Top diseases: Varicose veins Allergy Fungal infection Paralysis (brain hemorrhage) AIDS
## Top probabilities: 1, 0, 0, 0, 0
##
## Observation 15 :
```

```
## Top diseases: Jaundice Chronic cholestasis Fungal infection AIDS Paralysis (brain hemorrhage)
## Top probabilities: 1, 0, 0, 0, 0
##
## Observation 14 :
## Top diseases: Paralysis (brain hemorrhage) Heart attack Gastroenteritis Hypertension Allergy
## Top probabilities: 1, 0, 0, 0, 0
##
## Observation 3 :
## Top diseases: GERD Heart attack Paralysis (brain hemorrhage) Gastroenteritis Drug Reaction
## Top probabilities: 1, 0, 0, 0, 0
##
## Observation 37 :
## Top diseases: (vertigo) Paroymsal  Positional Vertigo Paralysis (brain hemorrhage) Hypertension Hear
## Top probabilities: 1, 0, 0, 0, 0
```

Even though the probabilities associated with the different observations/patients may over fit for certain prognoses, there are intriguing relationships between the groups of diseases suggested.

For instance, let's have a look into the **observation 15** prediction.

- **Jaundice**
- **Hepatitis D**
- **Chronic cholestasis**
- **Alcoholic Hepatitis**
- **Hepatitis E**

These diseases are primarily liver-related conditions. Jaundice is a symptom of liver dysfunction, which can be caused by various conditions, including different types of hepatitis (Hepatitis D and E) and alcoholic hepatitis. Chronic cholestasis is also a liver condition characterized by a decrease in bile flow, which can lead to jaundice. It makes sense for the model to suggest these conditions as possible prognoses since they share a common organ system and similar symptoms.

Alternatively, for the **observation 3**, we can observe the following diseases:

- **GERD (Gastroesophageal reflux disease)**
- **Heart Attack**
- **Paralysis (brain hemorrhage)**
- **Gastroenteritis**
- **Drug Reaction**

In this case, GERD and Gastroenteritis are both gastrointestinal conditions, while Heart attack and Paralysis (brain hemorrhage) are cardiovascular and neurological conditions, respectively. A drug reaction can cause various symptoms that may mimic other conditions, making it a reasonable suggestion.

In conclusion, the diseases suggested by the model for each observation are not always directly related, but they often share overlapping symptoms, risk factors, or involve the same organ systems. This can make the model's suggestions reasonable for further consideration in the diagnostic process.

Conclusion

In conclusion, this study aimed to build and evaluate machine learning models to predict patient diagnoses based on their symptoms. The data was first prepared by splitting it into training and validation sets, converting variables, and fixing variable names. Four models were then created: Decision Tree, Random Forest, Naive Bayes, and kNN. Their performance was assessed using accuracy, F1-score, specificity, and sensitivity.

The results demonstrated exceptionally high accuracy for all models, with the Decision Tree being the only exception. This high performance can be attributed to the dataset's simplified nature, which presents diseases and their corresponding symptoms as binary functions. While these models showed proficiency in handling simplified data, their effectiveness in real-world situations where the correlation between diseases and symptoms is more complex might be reduced.

Variable Importance plots were used to identify the most significant symptoms for accurate predictions. Common symptoms that are associated with various diseases were found to be useful health predictors, enhancing the model's predictive ability.

Lastly, an additional analysis was performed using the Naive Bayes model to generate the top 5 most probable diseases for each observation, providing a broader range of options for further patient testing and evaluation. Although the probabilities associated with the different observations/patients may overfit for certain prognoses, the suggested groups of diseases often share overlapping symptoms, risk factors, or involve the same organ systems, making the models' suggestions reasonable for further consideration in the diagnostic process.

References

- *Disease Prediction using Machine Learning - Kaggle Datasets. Kaushil Ruparelia. Accessed April 5, 2023. https://www.kaggle.com/kaushil268/disease-prediction-using-machine-learning.*
- *Kaggle. Accessed April 5, 2023. https://www.kaggle.com/.*
- *ggplot2 Quick correlation matrix heatmap - R software and data visualization." STHDA. Accessed April 6, 2023. http://www.sthda.com/english/wiki/ggplot2-quick-correlation-matrix-heatmap-r-software-and-data-visualization.*
- *Arora, S., Vohra, R., & Bhardwaj, A. (2022). An interpretable machine learning approach for heart disease prediction using clinical and demographic features. Applied Sciences, 12(15), 7449. https://www.mdpi.com/2076-3417/12/15/7449*
- *Binary classification heart disease prediction-7 ideas how to start and improve your model. Microsoft Azure AI Gallery. Accessed April 10, 2023. https://gallery.azure.ai/Experiment/Binary-classification-heart-disease-prediction-7-ideas-how-to-start-and-improve-your-model-1.*
- *Vyas, R., & Patel, R. (2021). Machine learning approaches for binary classification to discover liver diseases using clinical data. ResearchGate. https://www.researchgate.net/publication/351105092_Machine_Learning_Approaches_for_Binary_Classification_to_Discover_Liver_Diseases_using_Clinical_Data.*
- *Smart Health Disease Prediction using Naive Bayes. NevonProjects. Accessed April 10, 2023. https://nevonprojects.com/smart-health-disease-prediction-using-naive-bayes/.*
- *Kononenko, I. (2001). Machine learning for medical diagnosis: history, state of the art and perspective. Artificial Intelligence in medicine, 23(1), 89-109. DOI: 10.1016/S0933-3657(01)00077-X*