An Ensemble Model to Predict the Recurrence of Differentiated Thyroid Cancer: Development and Evaluation

Swarnali Dasgupta

2025-01-02

Contents

Project Overview

The data being studied is the **'Differentiated Thyroid Cancer Recurrence'** dataset from the UC Irvine Machine Learning Repository. This dataset contains **13 clinicopathological features** for **383 patients** aimed at predicting the recurrence of differentiated thyroid cancer as 'yes' or 'no'.

The dataset was collected over a duration of 15 years, with each patient being followed for at least 10 years. Each row in the data represents a patient, while each column represents a predictive feature.

Machine learning uses algorithms that learn from data to predict something. These predictions can be generated through supervised learning, where algorithms learn patterns from existing data, or unsupervised learning, where they discover general patterns in data. Machine learning models can predict numerical values based on historical data, categorize events as true or false, and cluster data points based on commonalities. We will be mainly using logistic regression, random forest and kNN models since these models do particularly well with classification tasks. One hot encoding will be used to ensure kNN can handle the data.

Objectives This project aims to:
1. Train machine learning models to predict the likelihood of thyroid cancer recurrence.
2. Build the following models:
- Logistic Regression Model
- Random Forest Model
- k-Nearest Neighbors (kNN) Model
- Ensemble Model
3. Compare all models to determine the most efficacious model.

Workflow

1. **Data Acquisition**
2. **Data Exploration**
3. **Data Cleaning and Shaping**
4. **Logistic Regression + Performance Analysis**
5. **Random Forest Model + Performance Analysis**

6. **kNN Model + Performance Analysis**

7. **Model Comparison**

8. **Ensemble Model + Performance Analysis**

1. Data Acquisition

This step involves downloading the data from the repository and reading it into memory. Before this, all prerequisite packages must be downloaded and loaded.

```
# Read in the dataset
url <- ("https://archive.ics.uci.edu/static/public/915/differentiated+thyroid+cancer+recurrence.zip")
all_data <- read_csv(archive_read(url), show_col_types = FALSE)
data <- all_data
```

2. Data Exploration

This step involves exploring the data by inspecting the structure and summary of the data, inspecting for missing values and constructing bar plots to visualise the relationship between some of the categorical predictor features and the target variable. We also look for class imbalance, outlier detection & rectification and correlation testing using chi-squared test method.

```
#Explore the data
str(data)
```

```
## spc_tbl_ [383 x 17] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Age                 : num [1:383] 27 34 30 62 62 52 41 46 51 40 ...
##  $ Gender              : chr [1:383] "F" "F" "F" "F" ...
##  $ Smoking             : chr [1:383] "No" "No" "No" "No" ...
##  $ Hx Smoking          : chr [1:383] "No" "Yes" "No" "No" ...
##  $ Hx Radiothreapy     : chr [1:383] "No" "No" "No" "No" ...
##  $ Thyroid Function    : chr [1:383] "Euthyroid" "Euthyroid" "Euthyroid" "Euthyroid" ...
##  $ Physical Examination: chr [1:383] "Single nodular goiter-left" "Multinodular goiter" "Single nodul
##  $ Adenopathy          : chr [1:383] "No" "No" "No" "No" ...
##  $ Pathology           : chr [1:383] "Micropapillary" "Micropapillary" "Micropapillary" "Micropapilla
##  $ Focality            : chr [1:383] "Uni-Focal" "Uni-Focal" "Uni-Focal" "Uni-Focal" ...
##  $ Risk                : chr [1:383] "Low" "Low" "Low" "Low" ...
##  $ T                   : chr [1:383] "T1a" "T1a" "T1a" "T1a" ...
##  $ N                   : chr [1:383] "N0" "N0" "N0" "N0" ...
##  $ M                   : chr [1:383] "M0" "M0" "M0" "M0" ...
##  $ Stage               : chr [1:383] "I" "I" "I" "I" ...
##  $ Response            : chr [1:383] "Indeterminate" "Excellent" "Excellent" "Excellent" ...
##  $ Recurred            : chr [1:383] "No" "No" "No" "No" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Age = col_double(),
##   ..   Gender = col_character(),
##   ..   Smoking = col_character(),
##   ..   `Hx Smoking` = col_character(),
##   ..   `Hx Radiothreapy` = col_character(),
##   ..   `Thyroid Function` = col_character(),
##   ..   ‘Physical Examination‘ = col_character(),
##   ..   Adenopathy = col_character(),
##   ..   Pathology = col_character(),
##   ..   Focality = col_character(),
##   ..   Risk = col_character(),
##   ..   T = col_character(),
##   ..   N = col_character(),
##   ..   M = col_character(),
##   ..   Stage = col_character(),
##   ..   Response = col_character(),
##   ..   Recurred = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
#dim(data)
summary(data)
```

```
##       Age           Gender            Smoking           Hx Smoking
##  Min.   :15.00   Length:383         Length:383         Length:383
##  1st Qu.:29.00   Class :character   Class :character   Class :character
##  Median :37.00   Mode  :character   Mode  :character   Mode  :character
##  Mean   :40.87
##  3rd Qu.:51.00
##  Max.   :82.00
##  Hx Radiothreapy    Thyroid Function   Physical Examination  Adenopathy
##  Length:383         Length:383         Length:383            Length:383
##  Class :character   Class :character   Class :character      Class :character
##  Mode  :character   Mode  :character   Mode  :character      Mode  :character
##
##
##
##   Pathology           Focality             Risk                 T
##  Length:383         Length:383         Length:383         Length:383
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##       N                  M                 Stage             Response
##  Length:383         Length:383         Length:383         Length:383
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##    Recurred
##  Length:383
##  Class :character
##  Mode  :character
##
##
##
```

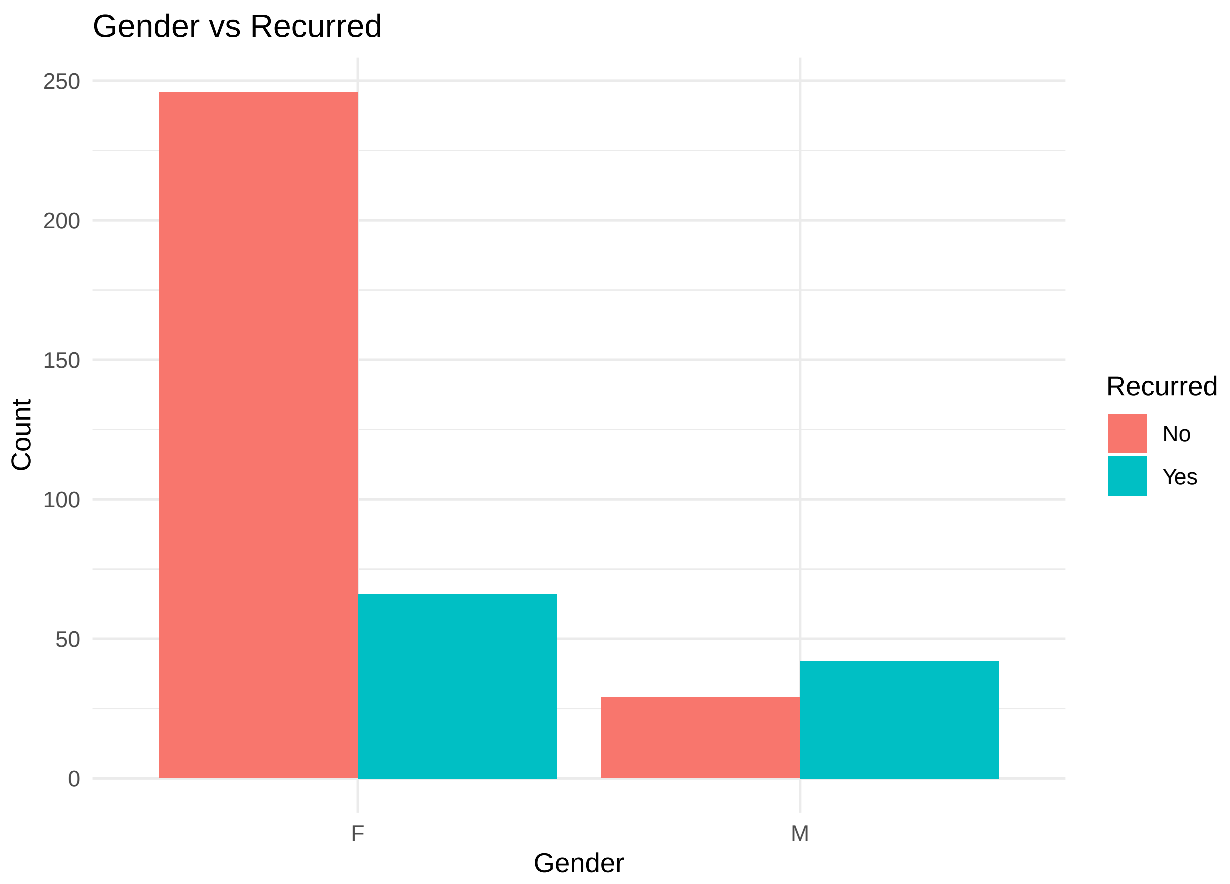
```
ifelse(any(is.na(data)), "NA values present", "No NA values")
```

```
## [1] "No NA values"
```

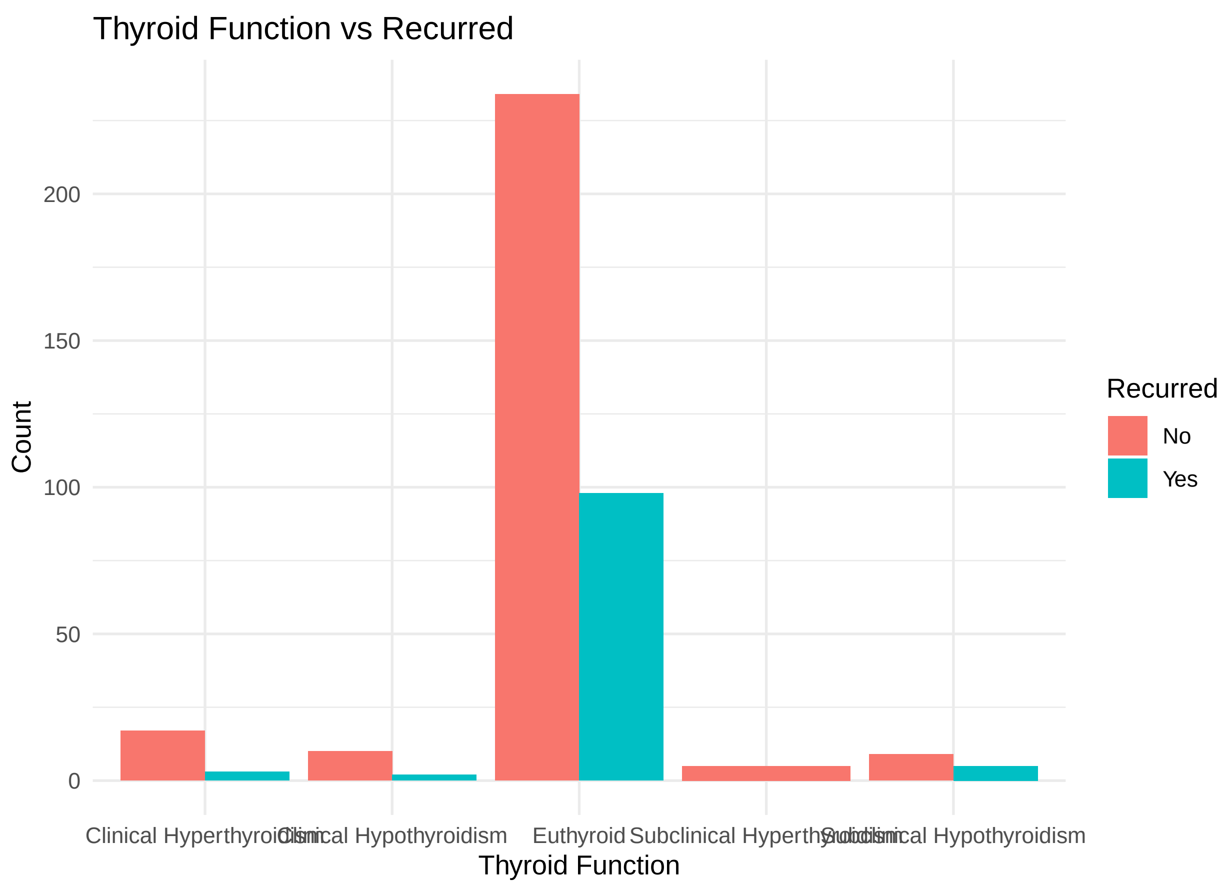
There are 383 patients with 17 clinico-pathological features.

Construct plots to explore relationships: We are constructing barplots to explore relationships between some of the predictor variables and the target variable.

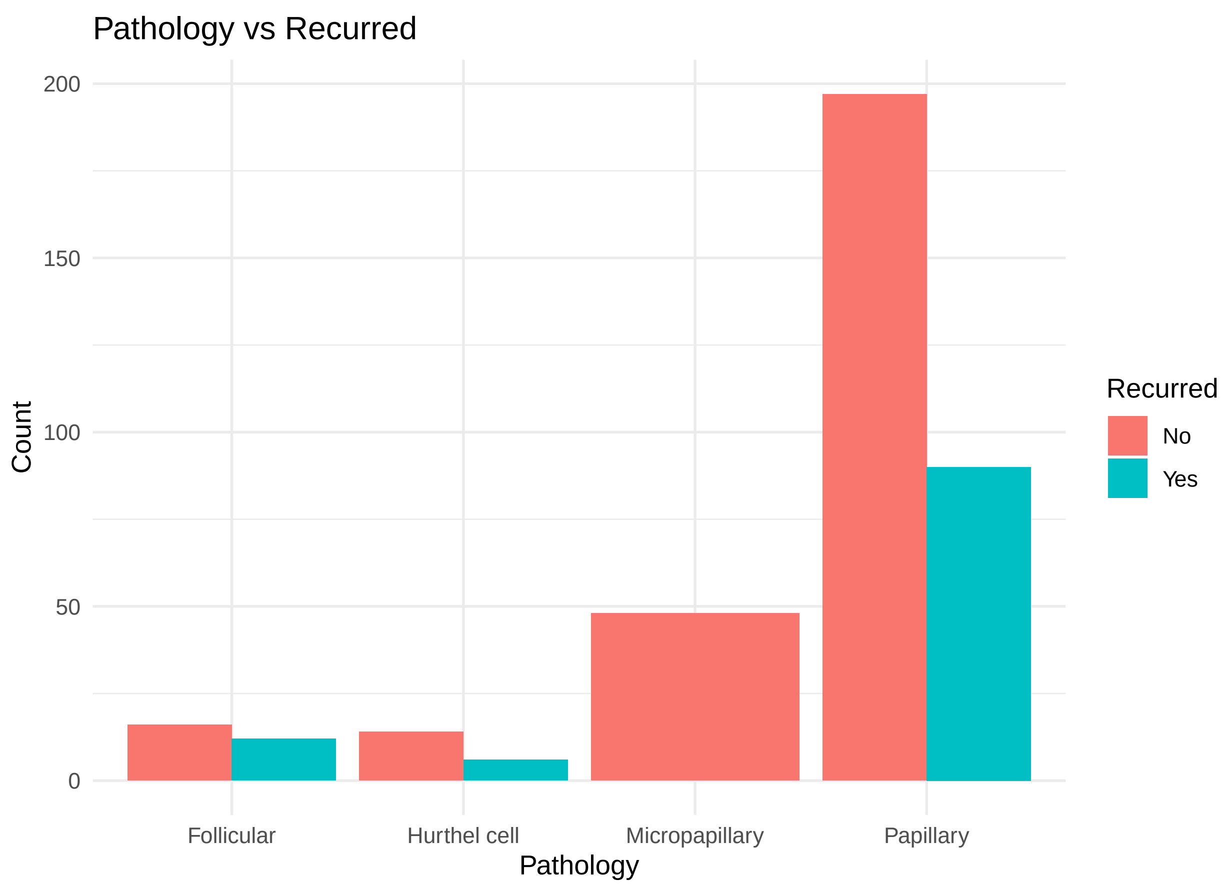
```
#Barplot for Gender vs Recurred (Grouped)
ggplot(data, aes(x = Gender, fill = Recurred)) +
  geom_bar(position = "dodge") +  # Use "dodge" for grouped bars
  labs(title = "Gender vs Recurred",
       x = "Gender",
       y = "Count") +
  theme_minimal()
```



```
#Barplot for Thyroid Function vs Recurred (Grouped)
ggplot(data, aes(x = `Thyroid Function`, fill = Recurred)) +
  geom_bar(position = "dodge") +
  labs(title = "Thyroid Function vs Recurred",
       x = "Thyroid Function",
       y = "Count") +
  theme_minimal()
```



```r
#Barplot for Pathology vs Recurred (Grouped)
ggplot(data, aes(x = Pathology, fill = Recurred)) +
  geom_bar(position = "dodge") +
  labs(title = "Pathology vs Recurred",
       x = "Pathology",
       y = "Count") +
  theme_minimal()
```



Check for class imbalance: Visualize class imbalance in a table

```
##
##  No Yes
## 275 108
```

There appears to be some class imbalance, which we will handle individually for the models by assigning weights or other techniques.

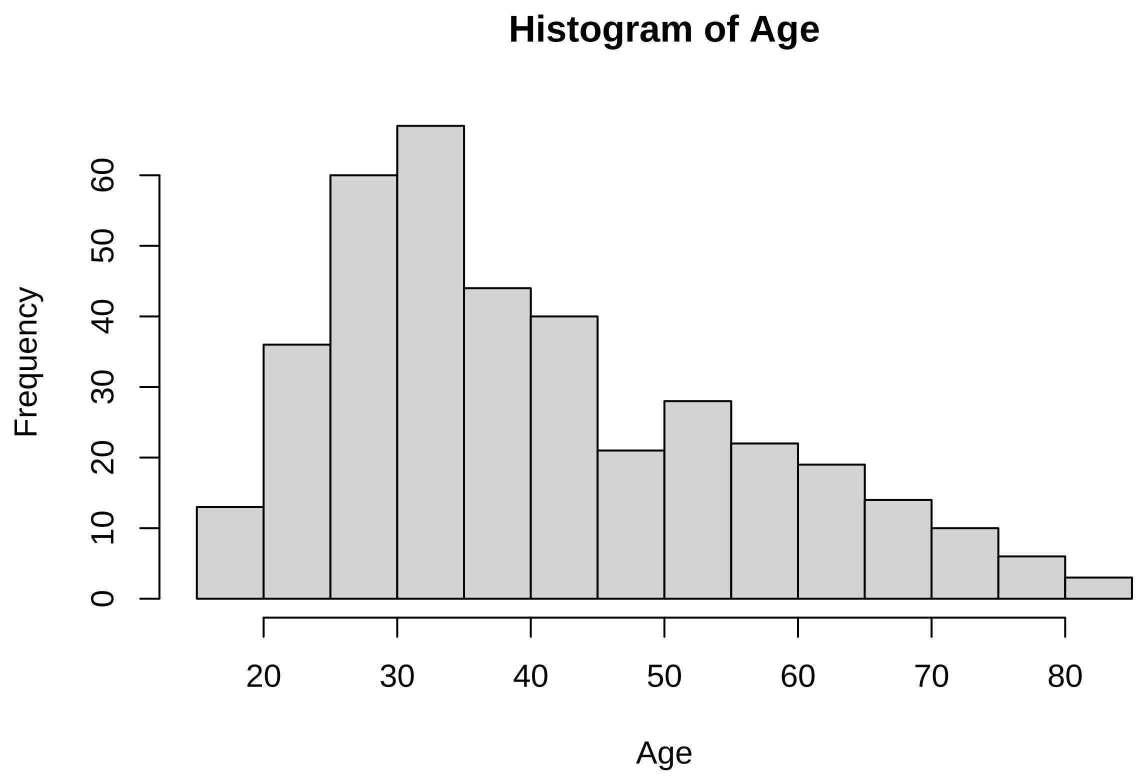
Check for outliers in Age column: Perform outlier detection and rectification

```
## Number of outliers:  0
```

No outliers, hence no need to deal with them.

Check for data distribution: Since all but one of our features are categorical, we will be checking the data distribution for only the Age column.

```
##
##  Shapiro-Wilk normality test
##
## data:  data$Age
## W = 0.94223, p-value = 4.667e-11
```



From the results, we can conclude that the 'Age' column is not normally distributed.

Correlation analysis: We used Chi-Square testing to do correlation analysis.

```
target <- "Recurred"
categorical_columns <- setdiff(colnames(data), c("Age", "Recurred"))

#Perform Chi-Square test for each categorical column to see its correlation with the target column
chi_sq_results <- lapply(categorical_columns, function(col) {
  table <- table(data[[col]], data[[target]])
  result <- chisq.test(table)
  list(
    Feature = col,
    p_value = result$p.value,
    statistic = result$statistic
  )
})

#Convert to dataframe
chi_sq_results_df <- do.call(rbind, lapply(chi_sq_results, as.data.frame))
print(chi_sq_results_df)
```

```
##                          Feature      p_value  statistic
## X-squared                 Gender 3.458852e-10  39.396676
## X-squared1               Smoking 2.129504e-10  40.344074
## X-squared2            Hx Smoking 1.448974e-02   5.977474
## X-squared3       Hx Radiothreapy 2.795966e-03   8.936051
## X-squared4      Thyroid Function 2.723787e-01   5.148687
## X-squared5  Physical Examination 1.140166e-02  12.974379
## X-squared6            Adenopathy 4.220721e-32 157.044293
## X-squared7             Pathology 3.546645e-05  23.270435
## X-squared8              Focality 1.446380e-13  54.641643
## X-squared9                  Risk 4.507816e-46 208.826203
## X-squared10                    T 5.353537e-28 141.290246
## X-squared11                    N 5.443985e-34 153.186764
## X-squared12                    M 2.616837e-11  44.444406
## X-squared13                Stage 3.161073e-20  97.617971
## X-squared14             Response 8.863124e-67 309.472321
```

The predictor features are not independent of the target variable 'Recurred'. There is a strong association between these features and the likelihood of recurrence. Features with significant p-values are potentially valuable predictors for our models.

3. Data Cleaning and Shaping

This step involves 'wrangling' the data to fit our model's requirements. First, we impute missing values. Since our dataset does not have any missing values, we must simulate missing values and impute them using the column's mean. We will be doing this for our only numeric column, 'Age'. After this, we will be using one-hot encoding to convert categorical columns into numerical ones so that they can be read using our selected algorithms. Next, we will perform principal component analysis to capture variance, feature engineering to optimise model performance and finally we will normalise the data so that it can be uniformly read by our algorithms.

Identification of missing values: Remove some values and simulate identification and imputation of missing values

```r
set.seed(123)

#Introduce missing values in column Age
missing_indices <- sample(1:nrow(data), size = floor(0.1 * nrow(data)))
data$Age[missing_indices] <- NA
#colSums(is.na(data))

#Impute the missing values using mean
data$Age[is.na(data$Age)] <- mean(data$Age, na.rm = TRUE)
#colSums(is.na(data))
```

Missing values were added and successfully imputed using the column mean. The column selected for this exercise is 'Age', since it is the only numerical column in this dataset.

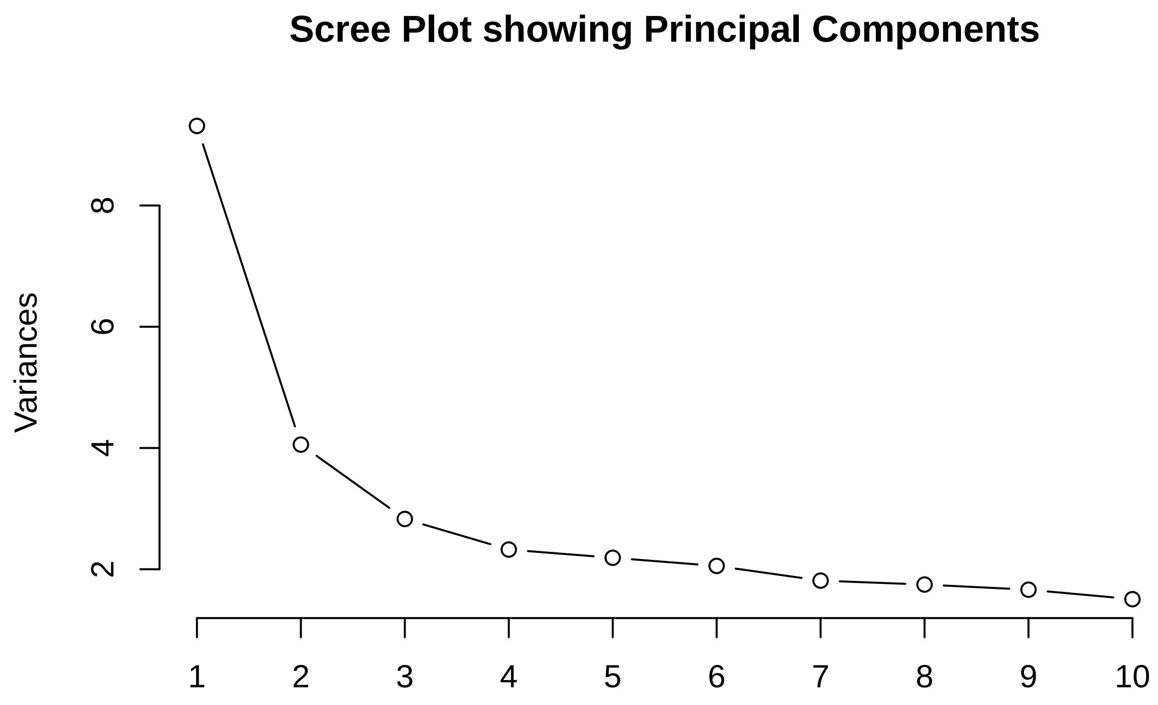
One-hot encoding to make the columns readable- for Yes and No columns: We encode the 'Yes' and 'No' columns.

```r
#Create a mapping function to automate
mappings <- list(
  Gender = c("M" = 1, "F" = 0),
  Smoking = c("Yes" = 1, "No" = 0),
  'Hx Smoking' = c("Yes" = 1, "No" = 0),
  'Hx Radiothreapy' = c("Yes" = 1, "No" = 0),
  Recurred = c("Yes" = 1, "No" = 0)
)

binary_encoding <- function(data, col.name, mappings) {
  if (col.name %in% names(mappings)) {
    data[[col.name]] <- mappings[[col.name]][data[[col.name]]]
  }
  return(data)
}

#Apply the mapping function
for (colname in colnames(data)) {
  data <- binary_encoding(data, colname, mappings)
}
```

One-hot encoding to make columns readable- for categorical columns: We encode the categorical columns.

```r
#Handle the dataset to isolate the categorical columns
categorical_columns <- c("Adenopathy", "Thyroid Function", "Physical Examination", "Pathology", "Focali
categorical_df <- data[, categorical_columns]
data <- data[, !colnames(data) %in% categorical_columns]

#Use caret package to create one hot encoding
dummies <- dummyVars(" ~ .", data = categorical_df)
one_hot_encoded <- predict(dummies, newdata = categorical_df)
one_hot_encoded <- as.data.frame(one_hot_encoded)

#cbind to original dataframe
data <- cbind(data, one_hot_encoded)

#Make syntactically valid column names
colnames(data) <- make.names(colnames(data))
```

```r
#### Principal component analysis
pca_result <- prcomp(data, center = TRUE, scale. = TRUE)
#summary(pca_result)
plot(pca_result, type = "l", main = "Scree Plot showing Principal Components")
```



Clean up column names to make processing easier: Here we reassign column names.

```r
#Remove spaces in the column names using gsub
colnames(data) <- gsub("^X.", "", colnames(data))
#colnames

#colnames(train.data) <- gsub(" ", ".", colnames(train.data))
#colnames(train.data) <- gsub("`", "", colnames(train.data))
#colnames(train.data) <- gsub("-", ".", colnames(train.data))
```

Feature engineering by combining T, N and M scores to give a combined risk score: ‘ Since T, N, and M are part of the widely accepted TNM staging system used in oncology, they are naturally correlated and can be analyzed together by combining them into ‘Severity’ score.

```r
#Calculate severity for T
data$T_severity <- (1 * data$TT1a) + (2 * data$TT1b) + (3 * data$TT2) +
                   (4 * data$TT3a) + (5 * data$TT3b) + (6 * data$TT4a) + (7 * data$TT4b)
#Calculate severity for N
data$N_severity <- (0 * data$NN0) + (1 * data$NN1a) + (2 * data$NN1b)
#Calculate severity for M
data$M_severity <- (0 * data$MM0) + (1 * data$MM1)

#Combine into a single Severity score
data$Severity <- data$T_severity + data$N_severity + data$M_severity
summary(data$Severity)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   3.000   3.796   5.000  10.000
```

```r
table(data$Severity)
```

```
##
##   1   2   3   4   5   6   7   8   9  10
##  47  39 119  68  36  36  15  12   7   4
```

Normalize the age and severity columns using min-max scaling: Normalization is an important step to ensure the model treats the features equally.

```r
#Normalize Age
data$Scaled.Age <- (data$Age - min(data$Age))/(max(data$Age) - min(data$Age))
data <- data[, !colnames(data) %in% "Age"]

#Normalize Severity
data$Severity <- (data$Severity - min(data$Severity))/(max(data$Severity) - min(data$Severity))
backup_data <- data
```

All of our data is now cleaned and shaped. Further shaping is to be done for individual models.

4. Logistic Regression + Performance Analysis

A logistic regression model is a supervised machine learning algorithm that performs classification tasks by predicting the probability of an outcome. Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Here, we will also use bagging with homogenous learners to boost the model's performance. Once the model is built, we will be using metrics like accuracy, sensitivity, precision, ROC curve and AUC values to gauge the model's performance.

Model building: This step involves the crux of this model- building it. We tidy the data more for this algorithm and set up the design.

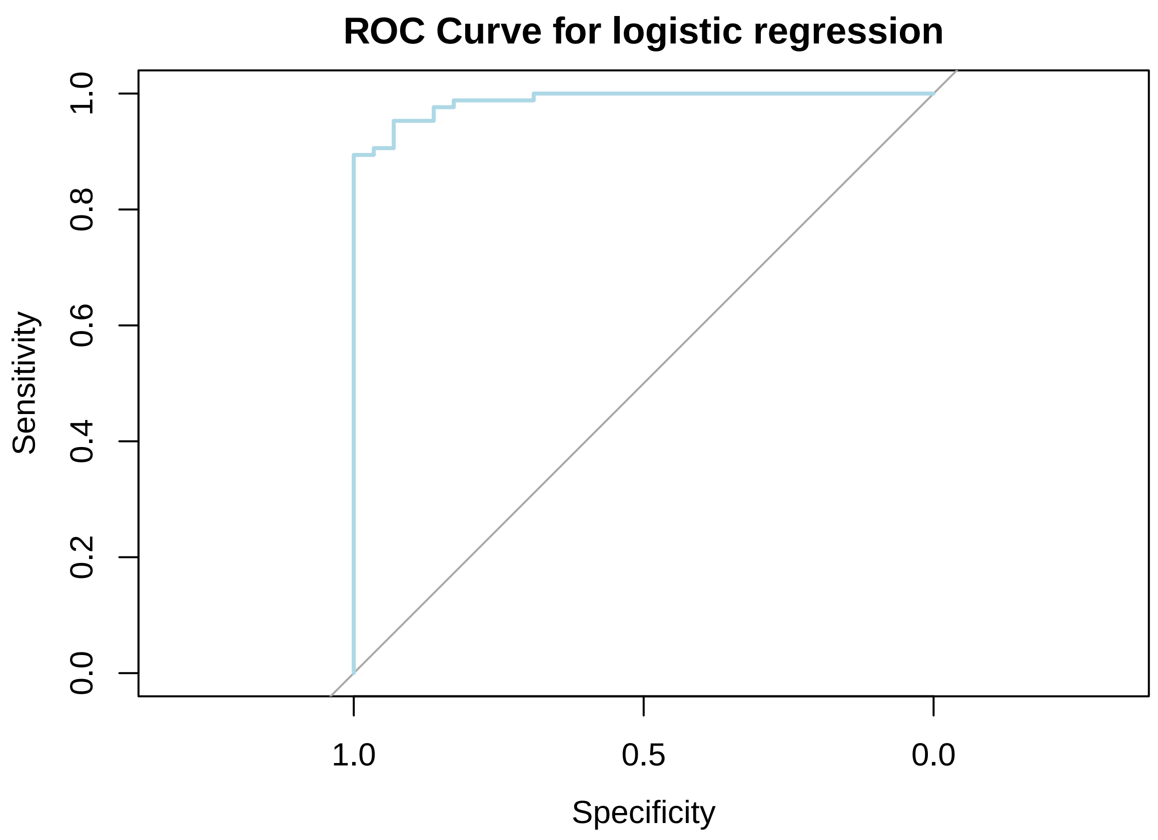
```r
data <- backup_data
set.seed(456)

#Create data split
cols_to_remove <- grep("^(TT|NN|MM)", colnames(data), value = TRUE)
data <- data[, !colnames(data) %in% cols_to_remove]

train.index <- createDataPartition(data$Recurred, p = 0.7, list = FALSE)
train.data <- data[train.index,]
test.data <- data[-train.index,]

#Replace 0 with "Not recurred" and 1 with "Recurred"
train.data$Recurred <- factor(train.data$Recurred, levels = c(0, 1), labels = c("Not recurred", "Recurr
test.data$Recurred <- factor(test.data$Recurred, levels = c(0, 1), labels = c("Not recurred", "Recurred

#Relevel the Recurred column
train.data$Recurred <- relevel(train.data$Recurred, ref = "Recurred")
test.data$Recurred <- relevel(test.data$Recurred, ref = "Recurred")


#Set number of bootstrap iterations and create a matrix to store
n_bags <- 25
bagging_predictions <- matrix(NA, nrow = nrow(test.data), ncol = n_bags)

#Perform bagging
for (i in 1:n_bags) {
  #Bootstrap sampling
  bootstrap_idx <- sample(1:nrow(train.data), replace = TRUE)
  train_bootstrap <- train.data[bootstrap_idx, ]
  total_samples <- nrow(train_bootstrap)
  weight_recurred <- total_samples / (2 * sum(train_bootstrap$Recurred == "Recurred"))
  weight_not_recurred <- total_samples / (2 * sum(train_bootstrap$Recurred == "Not recurred"))
  class_weights <- ifelse(train_bootstrap$Recurred == "Recurred", weight_recurred, weight_not_recurred)

  #Train the logistic regression model
  logistic.model <- glm(Recurred ~ ., data = train_bootstrap, family = binomial, weights = class_weights

  #Use the model to predict
  bagging_predictions[, i] <- predict(logistic.model, newdata = test.data, type = "response")
}

#Combine predictions
logistic_probabilities <- rowMeans(bagging_predictions)

#Convert to predictions
logistic_predictions <- ifelse(logistic_probabilities > 0.5, "Not recurred", "Recurred")
logistic_predictions <- factor(logistic_predictions, levels = levels(test.data$Recurred))

#Confusion Matrix
conf_matrix <- confusionMatrix(logistic_predictions, test.data$Recurred)
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     Recurred Not recurred
##   Recurred           27            6
##   Not recurred        2           79
##
##                Accuracy : 0.9298
##                  95% CI : (0.8664, 0.9692)
##     No Information Rate : 0.7456
##     P-Value [Acc > NIR] : 3.753e-07
##
##                   Kappa : 0.8231
##
```

```
##  Mcnemar's Test P-Value : 0.2888
##
##             Sensitivity : 0.9310
##             Specificity : 0.9294
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.9753
##              Prevalence : 0.2544
##          Detection Rate : 0.2368
##    Detection Prevalence : 0.2895
##       Balanced Accuracy : 0.9302
##
##        'Positive' Class : Recurred
##
```

Performance analysis: Let us look into how this model has performed by extracting some statistics.

```
## Accuracy for logistic regression model:  0.9298246

## Sensitivity for logistic regression model:  0.9310345

## Specificity for logistic regression model:  0.9294118

## Precision for logistic regression model:  0.8181818
```



```
## AUC for logistic regression model:  0.9874239
```

The accuracy, precision, sensitivity and specificity metrics help us infer a lot about how the model is performing. The confusion matrix printed above shows that the model is performing well in a sense that it is recognizing the 'Recurred' and 'Not recurred' groups correctly, with some more false positives than false negatives. In medicine, this is a good practice because it enables physicians to err on the side of caution. The accuracy seems to be quite good; the ROC curve gives further evidence regarding good model performance and the AUC value seems good as well. However, such a high AUC value could also indicate some over fitting, so that should be further investigated.

5. Random Forest Model + Performance Analysis

Random Forest algorithm is a powerful machine learning algorithm used for both classification and regression tasks, that works by creating a number of decision trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees thereby reducing the risk of over fitting. The package randomForest inherently uses bagging to boost its performance, and hence we will not be bagging explicitly. Once the model is built, we will be using metrics like accuracy, sensitivity, precision, ROC curve and AUC values to gauge the model's performance.

Model building: This step involves the crux of this model- building it. We tidy the data more for this algorithm and set up the design.

```r
data <- backup_data

#Create data split
cols_to_remove <- grep("^(TT|NN|MM)", colnames(data), value = TRUE)
data <- data[, !colnames(data) %in% cols_to_remove]
train.index <- createDataPartition(data$Recurred, p = 0.7, list = FALSE)
train.data <- data[train.index,]
test.data <- data[-train.index,]

#Generate labels for the knn function
train.labels <- train.data$Recurred
test.labels <- test.data$Recurred

#Replace 0 with "Not recurred" and 1 with "Recurred" for better clarity
train.labels <- factor(train.labels, levels = c(0, 1), labels = c("Not recurred", "Recurred"))
test.labels <- factor(test.labels, levels = c(0, 1), labels = c("Not recurred", "Recurred"))

#Relevel the Recurred column
train.labels <- factor(train.labels, levels = c("Recurred", "Not recurred"))
test.labels <- factor(test.labels, levels = c("Recurred", "Not recurred"))

#Remove the recurred column from the test and train sets
train.data <- train.data[, -which(colnames(train.data) == "Recurred")]
test.data <- test.data[, -which(colnames(test.data) == "Recurred")]

#Fit the random forest model
rf.model <- randomForest(x = train.data, y = train.labels, ntree = 100, mtry = sqrt(ncol(train.data)),
print(rf.model)
```

```
##
## Call:
##  randomForest(x = train.data, y = train.labels, ntree = 100, mtry = sqrt(ncol(train.data)),      imp
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 3.35%
## Confusion matrix:
##              Recurred Not recurred class.error
## Recurred           62            6  0.08823529
## Not recurred        3          198  0.01492537
```

```r
#Predict test data using the model
rf.probabilities <- predict(rf.model, newdata = test.data)

#Create a confusion matrix
conf_matrix <- confusionMatrix(rf.probabilities, test.labels)
conf_matrix
```

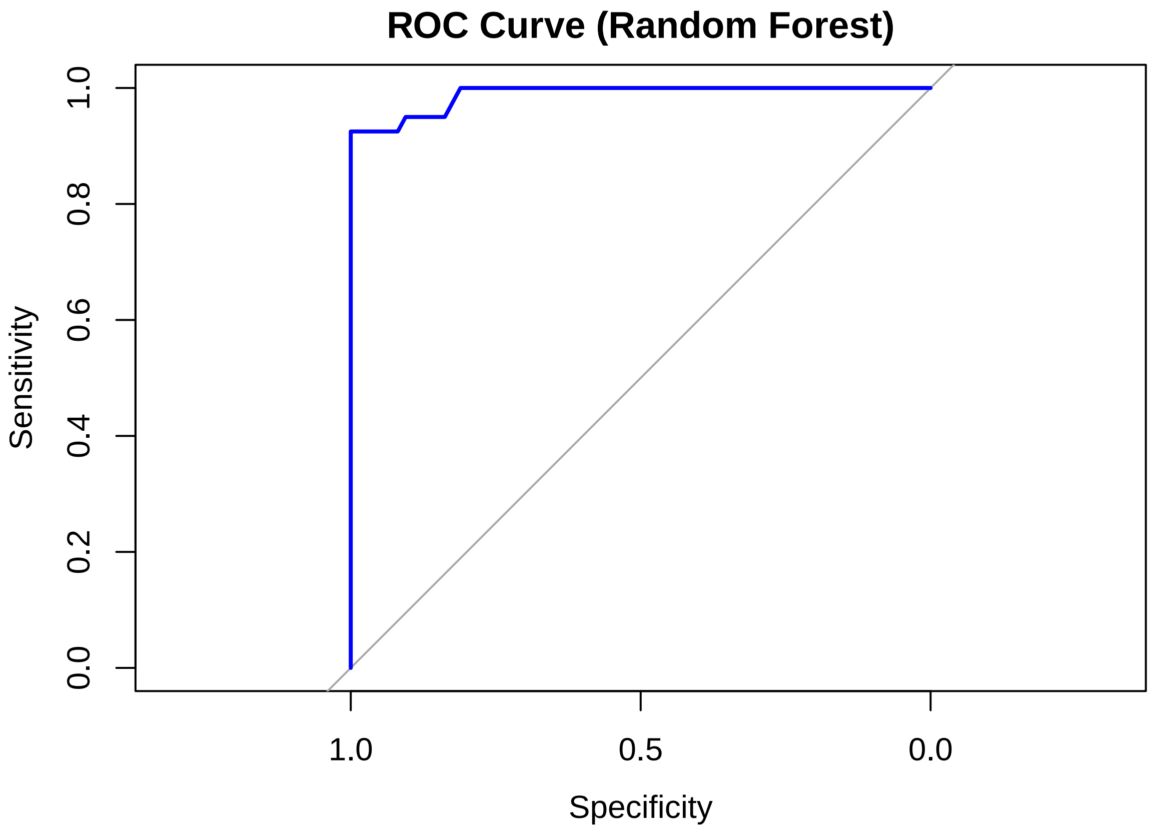
```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     Recurred Not recurred
##   Recurred           36            0
##   Not recurred        4           74
##
##                Accuracy : 0.9649
##                  95% CI : (0.9126, 0.9904)
##     No Information Rate : 0.6491
##     P-Value [Acc > NIR] : 2.456e-16
##
##                   Kappa : 0.9212
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.9000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9487
##              Prevalence : 0.3509
##          Detection Rate : 0.3158
##    Detection Prevalence : 0.3158
##       Balanced Accuracy : 0.9500
##
##        'Positive' Class : Recurred
##
```

Performance analysis: Let us look into how this model has performed by extracting some statistics.

```
## Accuracy for random forest model:  0.9649123
```

```
## Sensitivity for random forest model:  0.9
## Specificity for random forest model:  1
```

```
## Precision for random forest model:  1
```



```
## AUC for random forest model:  0.9890203
```

The accuracy, precision, sensitivity and specificity metrics help us infer a lot about how the model is performing. The confusion matrix printed above shows that the model is performing extremely well in a sense that it is recognizing the 'Recurred' and 'Not recurred' groups correctly; however, this model has some false negatives. In medicine, this is not a good practice since physicians cannot err on the side of caution, and the model can incorrectly classify no cancer recurrence even though there is a likelihood of cancer occurence. The accuracy seems to be quite good; the ROC curve gives further evidence regarding good model performance and the AUC value seems good as well. However, such a high AUC value could also indicate some over fitting, like the logistic regression model, so that should be further investigated.

6. KNN model

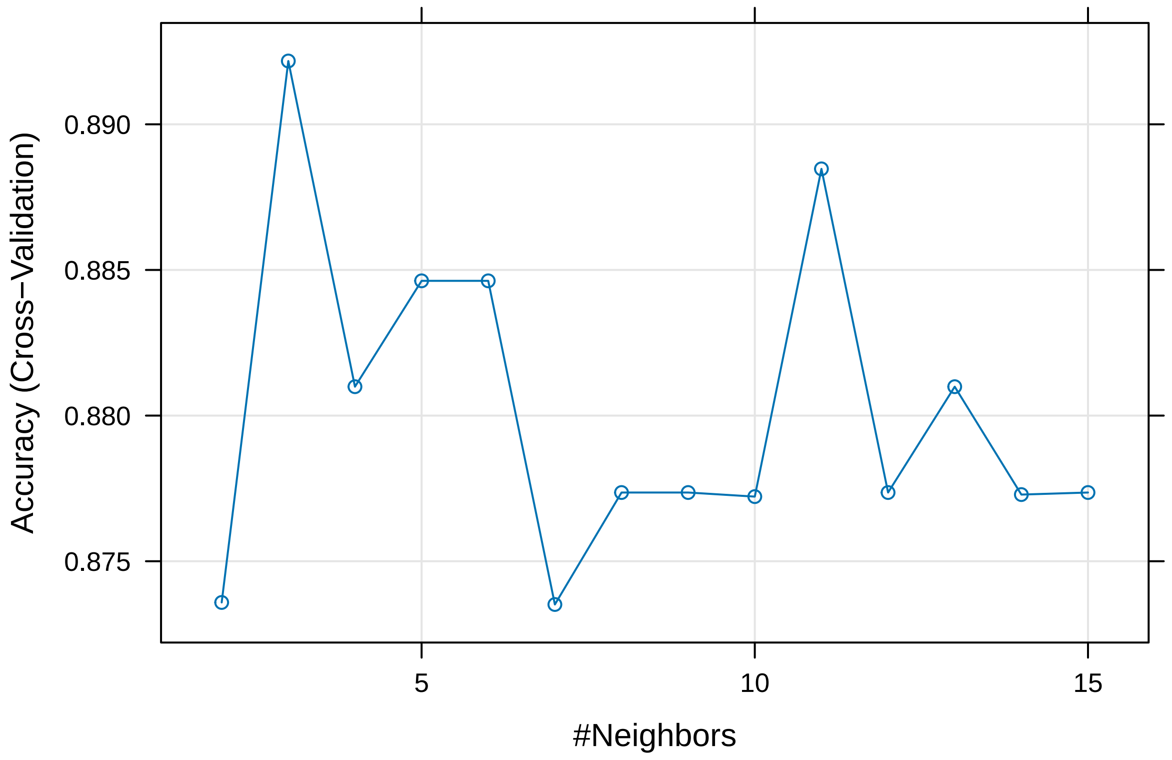
The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning method employed to tackle classification and regression problems. KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain. It is widely used in real-life scenarios since it is non-parametric, which means it does not make any assumptions about the distribution of data.

Model building: This step involves the crux of this model- building it. We tidy the data more for this algorithm and set up the design.

```r
data <- backup_data

set.seed(456)
#Create data split
train.index <- createDataPartition(data$Recurred, p = 0.7, list = FALSE)
train.data <- data[train.index,]
test.data <- data[-train.index,]

#Generate labels for the knn function
train.labels <- train.data$Recurred
test.labels <- test.data$Recurred

#Replace 0 with "Not recurred" and 1 with "Recurred" for better clarity
train.labels <- factor(train.labels, levels = c(0, 1), labels = c("Not recurred", "Recurred"))
test.labels <- factor(test.labels, levels = c(0, 1), labels = c("Not recurred", "Recurred"))

#Relevel the Recurred column
train.labels <- factor(train.labels, levels = c("Recurred", "Not recurred"))
test.labels <- factor(test.labels, levels = c("Recurred", "Not recurred"))

#Remove the recurred column from the test and train sets
train.data <- train.data[, -which(colnames(train.data) == "Recurred")]
test.data <- test.data[, -which(colnames(test.data) == "Recurred")]

#Normalize the data
preProc <- preProcess(train.data, method = c("center", "scale"))
train.data <- predict(preProc, train.data)
test.data <- predict(preProc, test.data)

#Define k-fold cross-validation control
levels(train.labels) <- make.names(levels(train.labels))
levels(test.labels) <- make.names(levels(test.labels))
train_control <- trainControl(method = "cv", number = 5, classProbs = TRUE)
k_values <- data.frame(k = 2:15)

#Train the cross validation
knn.cv <- train(
  x = train.data,
  y = train.labels,
  method = "knn",
  trControl = train_control,
  tuneGrid = k_values
)

#Select best k value
k = 3
cat("Best k: 3")
```

```
## Best k: 3
```

```r
plot(knn.cv)
```



```r
#Predict using model
knn.probabilities <- predict(knn.cv, newdata = test.data)

#Confusion Matrix
conf_matrix <- confusionMatrix(knn.probabilities, test.labels)
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Recurred Not.recurred
##   Recurred            21            1
##   Not.recurred         8           84
##
##                Accuracy : 0.9211
##                  95% CI : (0.8554, 0.9633)
##     No Information Rate : 0.7456
##     P-Value [Acc > NIR] : 1.564e-06
##
##                   Kappa : 0.7739
##
##  Mcnemar's Test P-Value : 0.0455
##
```

```
##             Sensitivity : 0.7241
##             Specificity : 0.9882
##          Pos Pred Value : 0.9545
##          Neg Pred Value : 0.9130
##              Prevalence : 0.2544
##          Detection Rate : 0.1842
##    Detection Prevalence : 0.1930
##       Balanced Accuracy : 0.8562
##
##        'Positive' Class : Recurred
##
```

Performance analysis: Let us look into how this model has performed by extracting some statistics.

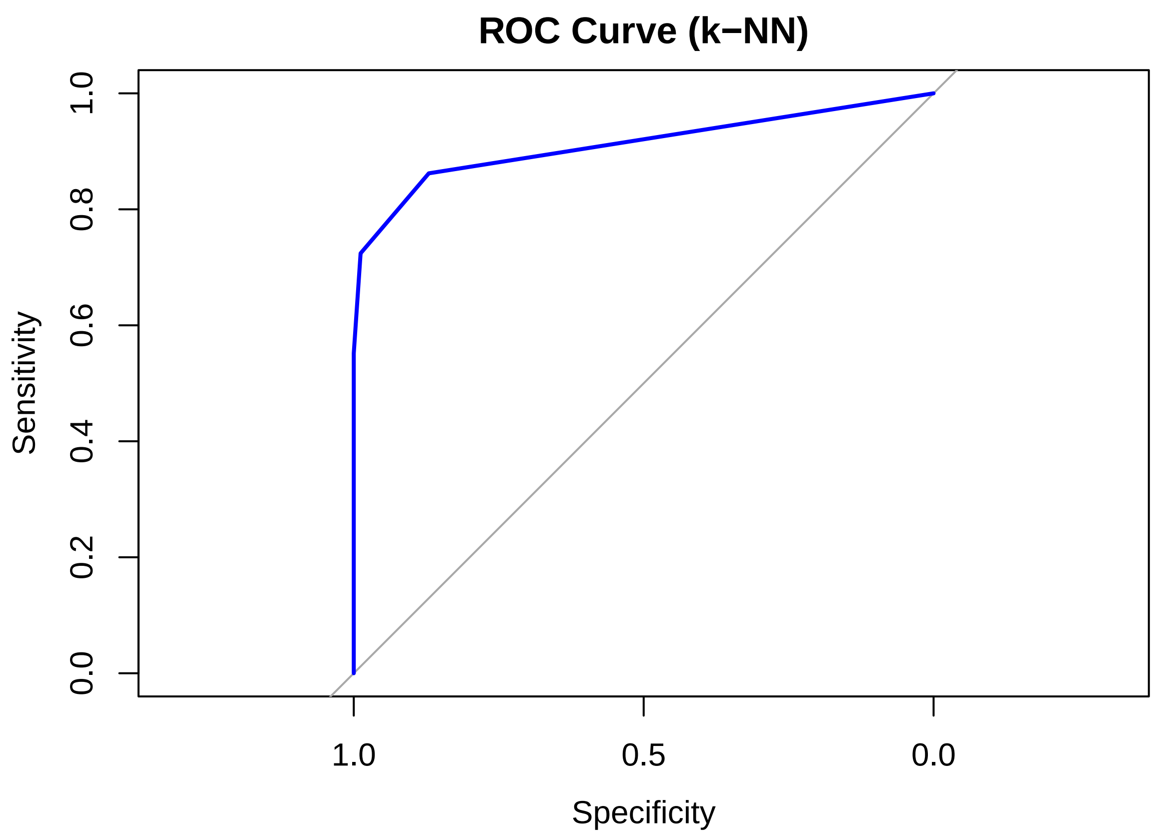
```
## Accuracy for kNN model:  0.9210526
```

```
## Sensitivity for kNN model:  0.7241379
```

```
## Specificity for kNN model:  0.9882353
```

```
## Precision for kNN model:  0.9545455
```

```
## AUC:  0.911359
```



The accuracy, precision, sensitivity and specificity metrics help us infer a lot about how the model is performing. The confusion matrix printed above shows that the model is performing decently well in a sense that it is recognizing the 'Recurred' and 'Not recurred' groups correctly; however, this model has quite a lot of false negatives and also some false positives. In medicine, this is not a good practice since physicians cannot err on the side of caution, and the model can incorrectly classify no cancer recurrence even though there is a likelihood of cancer occurrence. The accuracy seems to be quite good; the ROC curve gives further evidence regarding good model performance and the AUC value seems good as well.

7. Model comparison

```
##                 Model  Accuracy Sensitivity Specificity Precision
## 1 Logistic Regression 0.9298246   0.9310345   0.9294118 0.8181818
## 2       Random Forest 0.9649123   0.9000000   1.0000000 1.0000000
## 3                k-NN 0.9210526   0.7241379   0.9882353 0.9545455
```

The table above appropriately shows the strengths and weaknesses of each model. When we look at accuracy, random forest model does the best because of how strong the model is inherently- it can work with multiple data types and uses multiple decision trees to reach a decision. However, when it comes to sensitivity, logistic regression is doing the best, while kNN is doing the best in case of specificity. The random forest model seems to show some evidence of over fitting, which has probably been caused due to some amount of data leakage. This must be investigated and rectified before deploying this model in practice.

8. Ensemble Model + Performance Analysis

Ensemble models are a machine learning approach to combine multiple other models in the prediction process. These models are referred to as base estimators. Ensemble models allow us to overcome the technical challenges of building a single estimator and instead combines the benefits of all the base estimators. We chose a voting ensemble to combine the strengths of our models and achieve more balanced and reliable predictions. By averaging the probabilities (soft voting), the ensemble helps to offset the weaknesses of individual models, reduces variability, and improves the chances of making accurate predictions. This approach works especially well when the models have similar but slightly different levels of accuracy across various parts of the dataset.

Ensemble model function: Let us build the ensemble model.

```
#Construct a function for ensemble model
ensemble_model <- function(logistic_model, rf_model, knn_model, test_data, test_labels) {
  #Logistic Regression
  lr_probabilities_recurred <- predict(logistic_model, newdata = test_data, type = "response")
  lr_probabilities_not_recurred <- 1 - lr_probabilities_recurred

  #Random Forest
  rf_probabilities_recurred <- predict(rf_model, newdata = test_data, type = "prob")[, "Recurred"]
  rf_probabilities_not_recurred <- predict(rf_model, newdata = test_data, type = "prob")[, "Not recurre

  #kNN
  knn_probabilities_recurred <- predict(knn_model, newdata = test_data, type = "prob")[, "Recurred"]
  knn_probabilities_not_recurred <- predict(knn_model, newdata = test_data, type = "prob")[, "Not.recur

  #Average probabilities across models
  ensemble_probabilities <- data.frame(
    Not.recurred = (lr_probabilities_not_recurred + rf_probabilities_not_recurred + knn_probabilities_n
    Recurred = (lr_probabilities_recurred + rf_probabilities_recurred + knn_probabilities_recurred) / 3
```

```r
  )

  #Predict the class with the highest probability
  ensemble_predictions <- ifelse(
    ensemble_probabilities$Recurred > ensemble_probabilities$Not.recurred,
    "Recurred", "Not.recurred"
  )

  #Convert to factor with the same levels as test_labels
  ensemble_predictions <- factor(ensemble_predictions, levels = levels(test_labels))

  #Build confusion matrix
  conf_matrix_ensemble <- confusionMatrix(ensemble_predictions, test_labels)
  return(list(
  predictions = ensemble_predictions,
  probabilities = ensemble_probabilities,
  conf_matrix = conf_matrix_ensemble
))
}
```

Apply the ensemble model: We will use the function we built above on our pre-built base models.

```r
#Use the function built above on our models
ensemble.model <- ensemble_model(
  logistic_model = logistic.model,
  rf_model = rf.model,
  knn_model = knn.caret,
  test_data = test.data,
  test_labels = test.labels
)
ensemble_probabilities <- ensemble.model$probabilities
conf_matrix <- ensemble.model$conf_matrix
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##               Reference
## Prediction     Recurred Not.recurred
##   Recurred           21            4
##   Not.recurred        8           81
##
##                Accuracy : 0.8947
##                  95% CI : (0.8233, 0.9444)
##     No Information Rate : 0.7456
##     P-Value [Acc > NIR] : 5.979e-05
##
##                   Kappa : 0.7093
##
##  Mcnemar's Test P-Value : 0.3865
##
##             Sensitivity : 0.7241
##             Specificity : 0.9529
##          Pos Pred Value : 0.8400
##          Neg Pred Value : 0.9101
##              Prevalence : 0.2544
##          Detection Rate : 0.1842
##    Detection Prevalence : 0.2193
##       Balanced Accuracy : 0.8385
##
##        'Positive' Class : Recurred
##
```

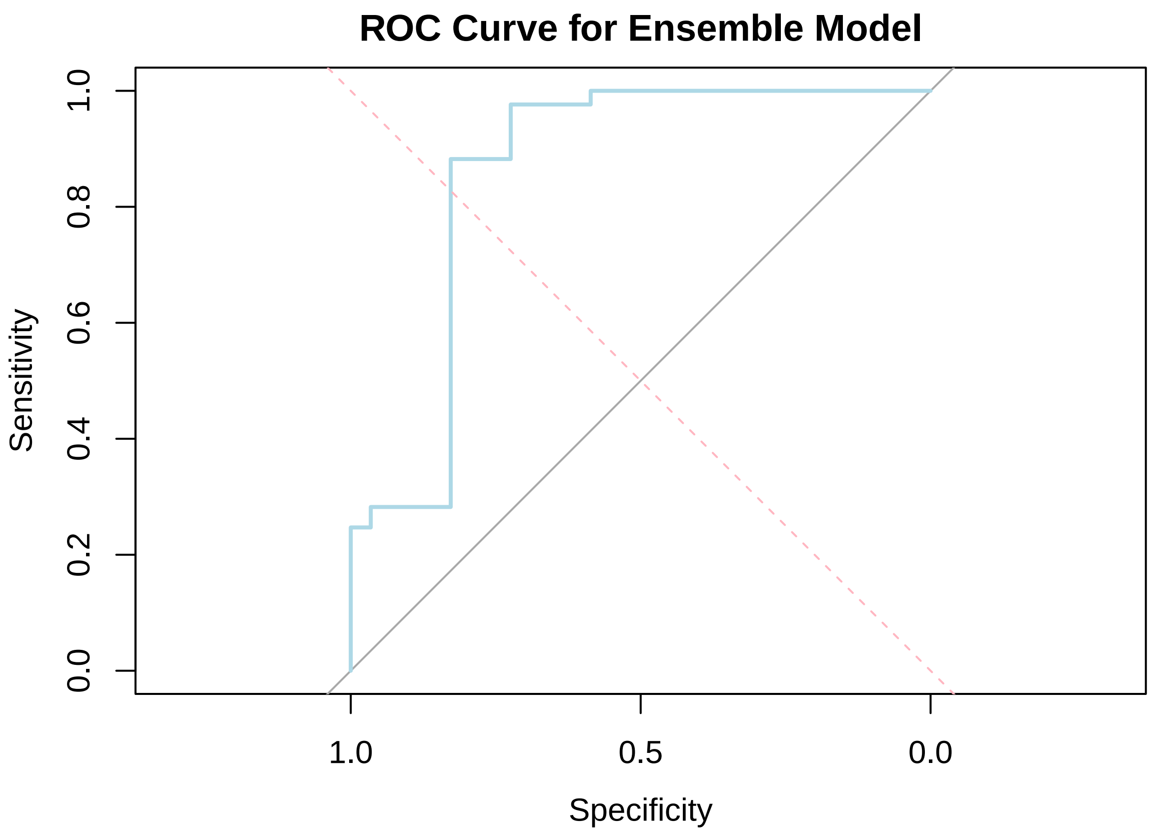
Performance analysis: Let us look into how this model has performed by extracting some statistics.

```
## Accuracy for ensemble model:  0.8947368
```

```
## Sensitivity for ensemble model:  0.7241379
```

```
## Specificity for kNN model:  0.9529412
```

```
## Precision for kNN model:  0.84
```

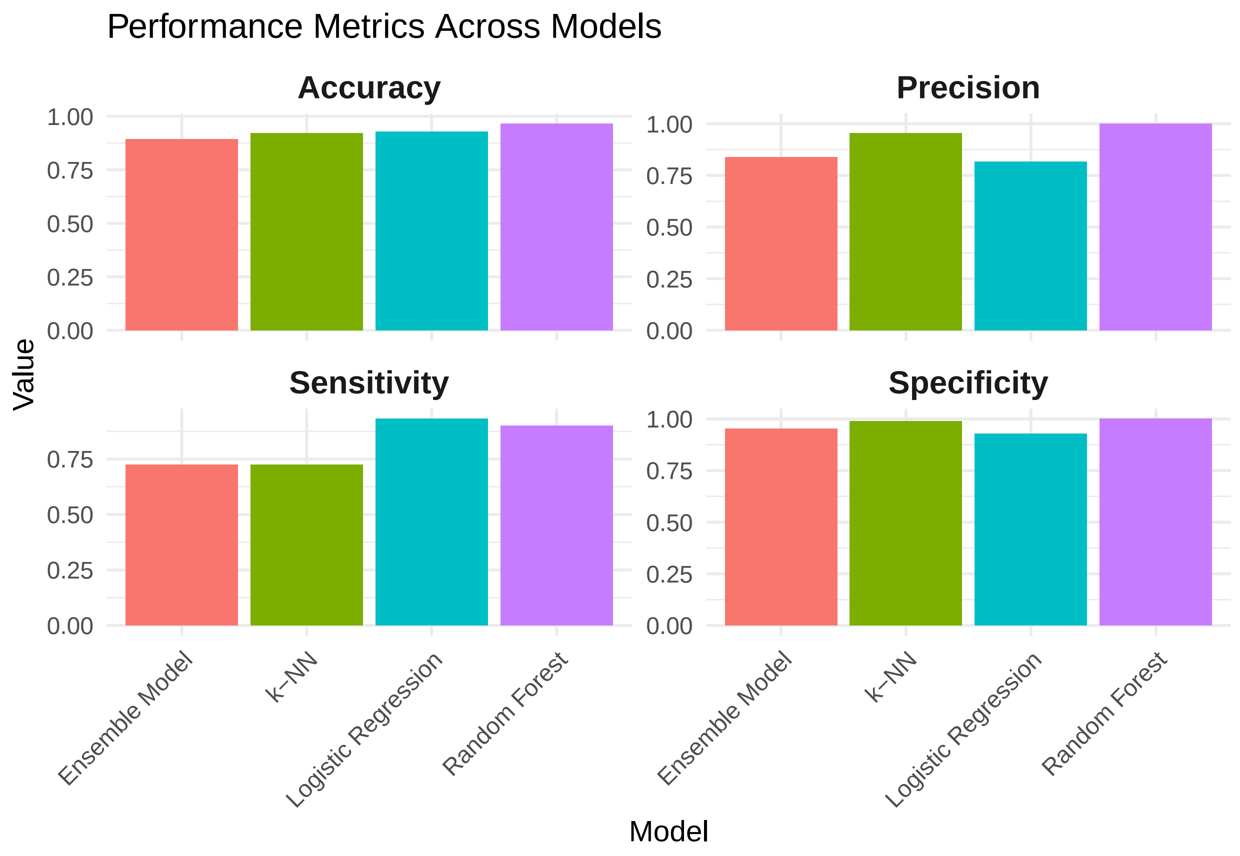


```
## AUC for ensemble model:  0.8596349
```

The accuracy, precision, sensitivity and specificity metrics help us infer a lot about how the model is performing. The confusion matrix printed above shows that the model is performing decently well in a sense that it is recognizing the 'Recurred' and 'Not recurred' groups correctly; however, this model has quite a lot of false negatives and also some false positives. In medicine, this is not a good practice since physicians cannot err on the side of caution, and the model can incorrectly classify no cancer recurrence even though there is a likelihood of cancer occurrence. The accuracy seems to be quite good; the ROC curve gives further evidence regarding good model performance and the AUC value seems good as well.

Compare ensemble model with other models: Let us compare the ensemble model to the other models individually.

```
##                  Model  Accuracy Sensitivity Specificity Precision
## 1 Logistic Regression 0.9298246   0.9310345   0.9294118 0.8181818
## 2       Random Forest 0.9649123   0.9000000   1.0000000 1.0000000
## 3                k-NN 0.9210526   0.7241379   0.9882353 0.9545455
## 4      Ensemble Model 0.8947368   0.7241379   0.9529412 0.8400000
```



These barplots give us insight into how the ensemble model fairs when compared to the other base models. In terms of the metrics being studied, it can be inferred that the ensemble model seems to be doing the best, closely followed by the logistic regression model. Hence, either of these models can be used to efficiently predict the recurrence of thyroid cancer in practice. However, some fine tuning may be necessary to reduce chances of over fitting to give accurate and reliable results.

Conclusion

This brings us to the end of this project where we successfully built models and accurately predicted the recurrence of thyroid cancer. Our choice of models is justified since all three models selected work well with classification data like this dataset, particularly random forest and logistic regression.