CLASSIFICATION OF BENIGN AND MALIGNANT MAMMOGRAPHIC MASSES BASED ON BI-RADS ATTRIBUTES

Barbu Sebastian-Marian (411), Plăian Georgiana (411), Stan Eduard-George (407),

Sîrbu Oana-Adriana (407)

INTRODUCTION

Mammography is the most effective method for breast cancer screening till this day. However, the low positive predictive value of breast biopsy resulting from mammogram interpretation leads to approximately 70% unnecessary biopsies with benign outcomes. To reduce the high number of unnecessary breast biopsies, several computed systems have been proposed to help physicians in their decision to perform a breast biopsy on a suspicious lesion seen in a mammogram or to perform a short-term follow-up examination instead.

The dataset we are using for this project was made with the intention of being used to predict the severity of a mammographic mass lesion from BI-RADS (*Breast Imaging Reporting and Data System*) attributes and the patient's age. The BI-RADS attributes consist of:

- Shape (mass shape): round = 1, oval = 2, lobular = 3, irregular = 4
- Margin (mass margin): circumscribed = 1, micro-lobulated = 2, obscured = 3, ill-defined = 4, spiculated = 5
- Density (mass density): high = 1, iso = 2, low = 3, fat-containing = 4

The dataset also contains BI-RADS assessments that are categorized into several levels from 0 to 6, but the BI-RADS assessment column contains values only from 1 to 5, that signify:

- Incomplete (0): additional imaging or information is needed to make a final assessment
- Negative (1): no significant abnormality is detected
- Benign (2): findings that are definitely benign

- Probably benign (3): findings that are likely benign but require short-term follow-up
- Suspicious (4): findings that are suspicious for malignancy, with a recommendation for biopsy
- Highly suspicious (5): findings that are highly suggestive of malignancy, with a strong recommendation for biopsy
- Known biopsy-proven malignancy (6): the presence of a confirmed malignancy

In addition, the dataset contains the ground truth, which is the severity field, where 0 is benign and 1 is malignant, for 516 benign and 445 malignant masses that have been identified on full field digital mammograms collected at the Institute of Radiology of the University Erlangen-Nuremberg between 2003 and 2006.

Throughout the project, we aimed to observe the percentage of Severity 1 cases (malignant) based on the other attributes, using various models such as:

- K-Nearest Neighbor (KNN)
- Logistic Regression
- Random Forest Classifier
- Support Vector Machines (SVM)

DATA PREPROCESSING

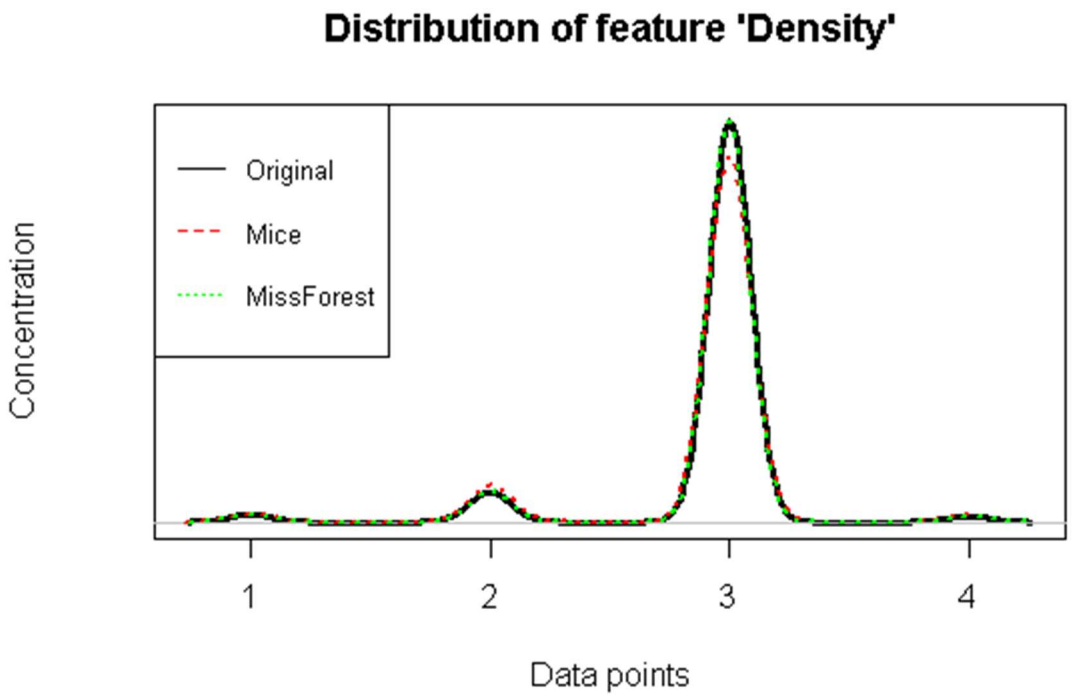
Firstly, we loaded the data into a data frame using *read.csv* function. After previewing the data frame, we observed that missing values are represented by "?" and we decided to convert them to the built-in NA. Moreover, the types automatically chosen for the features are not ideal. We changed the type accordingly, from *chr* to *int* (as *Age* feature has a wide range of values, we kept it integer), respectively, where the values were categorical, from *chr* to *factor* (such as our label, *Severity* column, which can have only two values).

Afterwards, we noticed there is only one occurrence of value “55” in feature *BI.RADS.assessment*, so we decided to remove it completely. After we have done that, we dropped the unused levels, since this feature is of type factor.

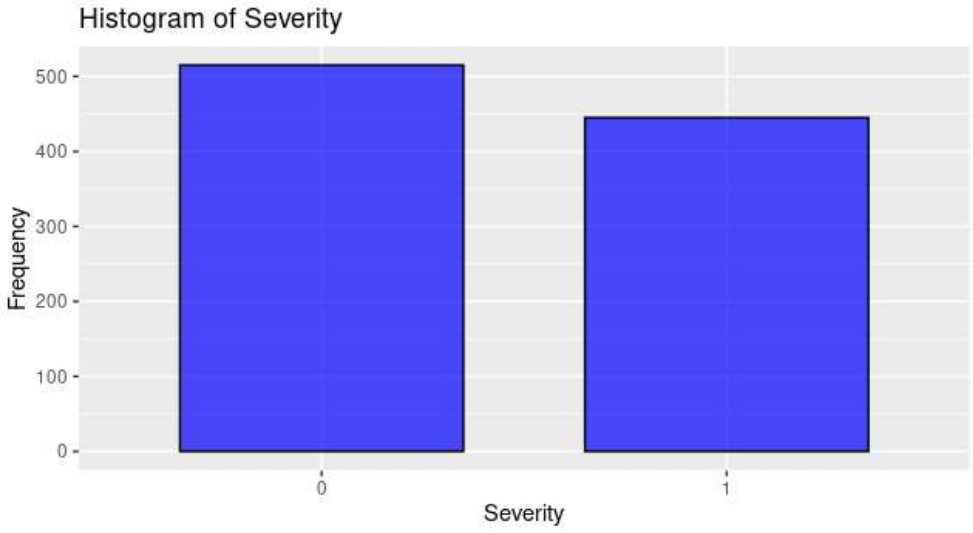
Since we still had a relatively high number of missing values, we had the options to impute or remove the corresponding rows. We decided to impute them. To do that, we used *mice*, an R package for Multivariate Imputation by Chained Equations. It imputes missing values using regression models, iteratively updating variables. Multiple imputations are generated for uncertainty consideration (multiple imputations involve creating several versions of the data frame, each with different imputed values for the missing data; these datasets collectively reflect the uncertainty about the true values of the missing observations).

After discussing with the professor, and because we wanted to have a different point of view and some other results to compare, we also used *missForest* to impute the missing values. *missForest* is a package in R that imputes the missing values using a random forest algorithm, by considering the relationships between variables.

Now that we imputed our data, we plotted a histogram to illustrate the distribution of values for the label. We noticed that the data is balanced, with the occurrence of benign severity being slightly higher, but for *missForest*, it looks like it makes the dataset more balanced, increasing the points that take value 2. To illustrate that better, here is a plot:



Lastly, we plotted a histogram to illustrate the distribution of values for the label after imputation.



Remark: To make this report easier to read, we will present the results for both *mice* and *missForest* in order to make a comparison.

THE USED MODELS

1. K-Nearest Neighbor (KNN)

K-Nearest Neighbor is a simple and widely used machine learning algorithm, particularly in the context of classification and regression tasks. This model is used for supervised learning tasks and thus it is a 'lazy learning algorithm', the predictions are made based on the proximity of the new data point.

To create a KNN model in R, one should first install and load the *class* package. This can be achieved with the following commands:

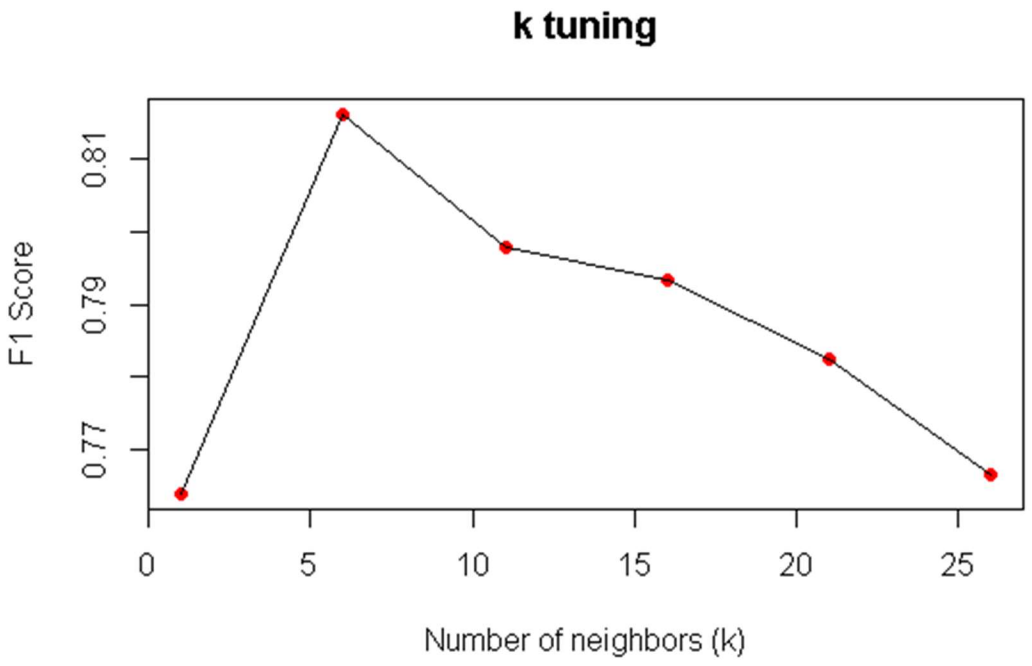
```
install.packages("class")
library(class)
```

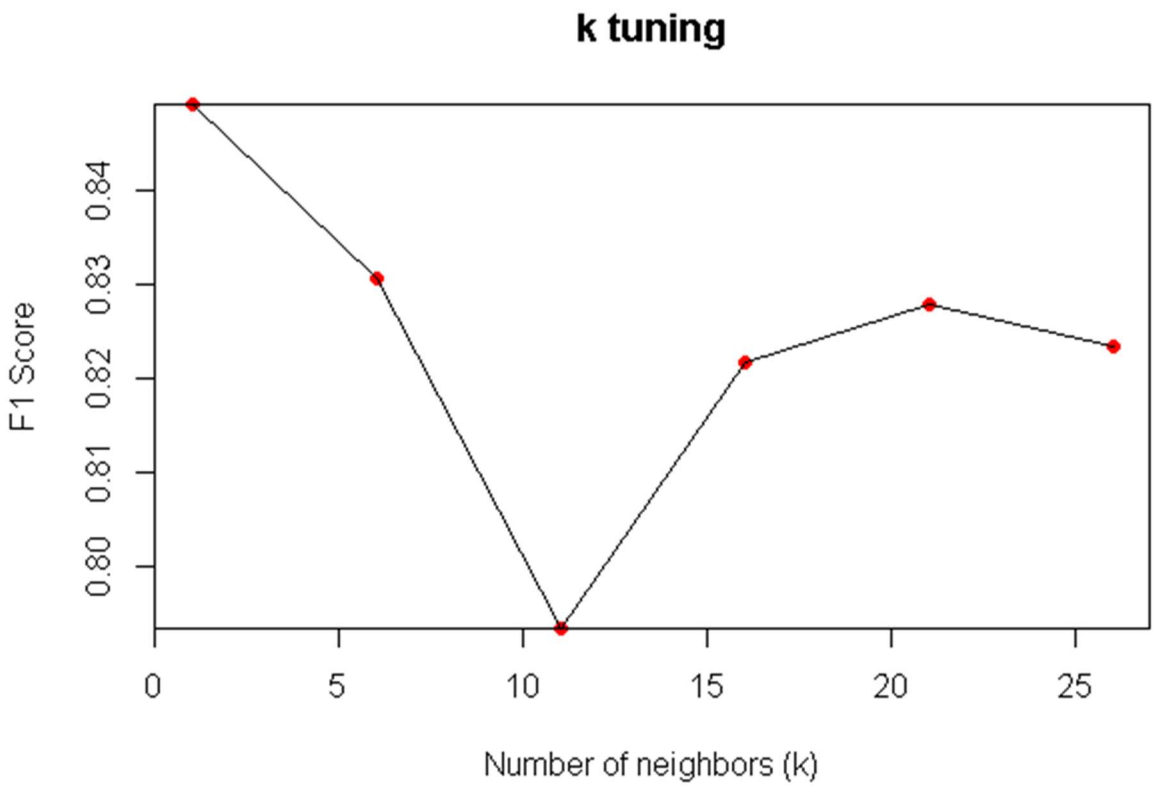
In R, one can create such model using the following code:

```
knn_model <- knn(train = train_data[, c("BI.RADS.assessment", "Age", "Shape", "Margin", "Density")],
        test = test_data[, c("BI.RADS.assessment", "Age", "Shape", "Margin", "Density")],
        cl = train_data$Severity,
        k = 6 / k=1)
```

In our case, *Severity* is the target variable, the one that needs to be predicted, while *BI.RADS.assesment*, *Age, Shape, Margin* and *Density* are the predictors. After applying a best 'k' hyperparameter search, we concluded that:

For *mice:* the best number of neighbors for this model is k = 6.
For *missForest:* the best number of neighbors for this model is k = 1.


k tuning
F1 Score
0.77
0.79
0.81
0
5
10
15
20
25
Number of neighbors (k)


k tuning
F1 Score
0.80
0.81
0.82
0.83
0.84
0
5
10
15
20
25
Number of neighbors (k)

The tables that contain the information shown in the previous graphs can be analyzed here:

For *mice:*

K	1	6	11	16	21	26
F1 score	0.764	0.816	0.797	0.793	0.782	0.766

For *missForest:*

K	1	6	11	16	21	26
F1 score	0.849	0.830	0.793	0.821	0.827	0.823

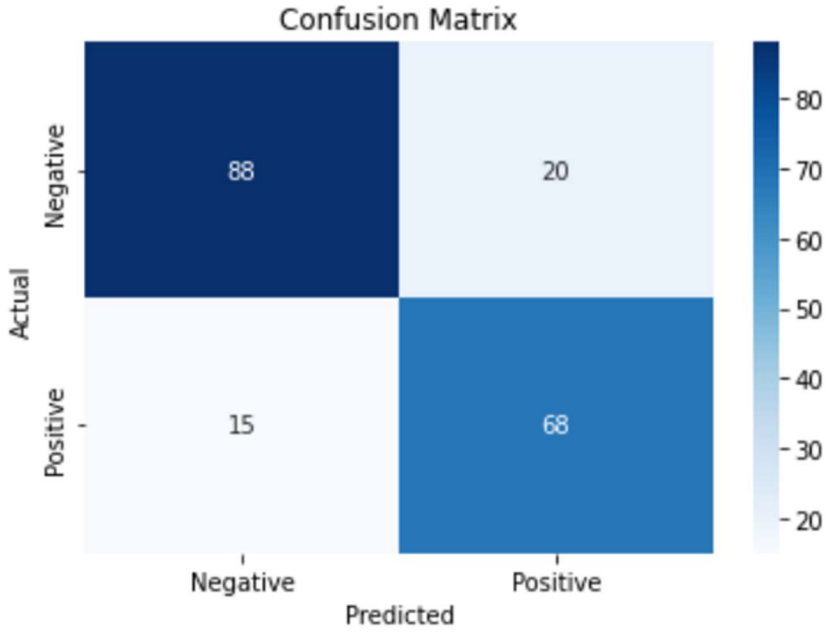
After training the best model, one can make predictions on the test set:

```
predictions <- knn_model
```

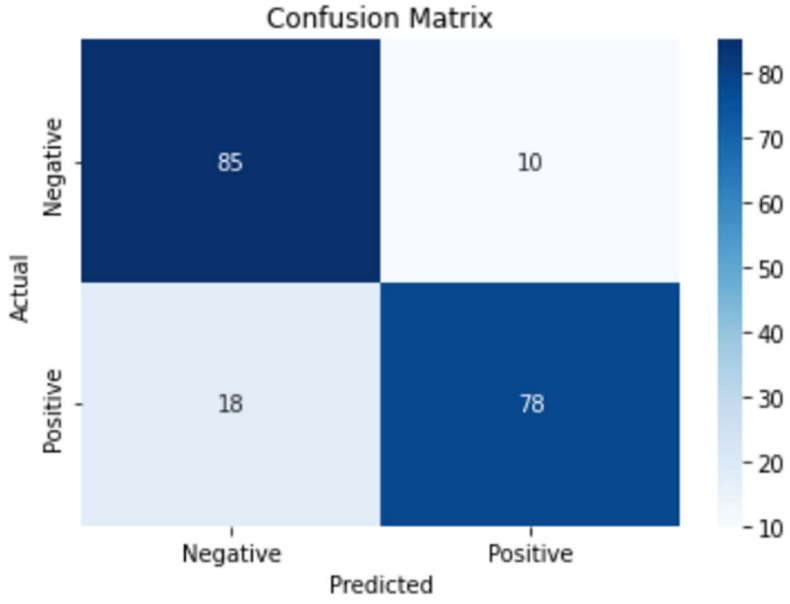
We evaluate the performance of this model using a confusion matrix, defined by:

```
Knn_confusion_matrix <- table(Predicted = knn_model, Actual = test_data$Severity)
```

For *mice:*



For *missForest:*



This compares the ground truth labels of the test set with the predicted ones. The results were computed:

For *mice:*

- Accuracy: 0.7801
- Precision: 0.75
- Recall: 0.784
- F1 score: 0.766

For *missForest:*

- Accuracy: 0.8272
- Precision: 0.777
- Recall: 0.875
- F1 score: 0.823

After checking various metrics like accuracy, precision, recall, and F1 score, we can conclude that our KNN model is performing really well on our dataset, without having the risk of overfitting and using *missForest* resulted in better results.

2. Logistic Regression

Logistic regression is a statistical method used for binary classification, which means it is employed when the dependent variable is binary, indicating the presence or absence of a particular outcome. In our case, since the output should be 0 (benign) or 1 (malignant), this model suits best the task.

To create a Logistic Regression in R, one does not need to install any particular package, since the function *glm*, used for creating our model, is a part of the basic R package called *stats*.

In R, one can create such a model using the following code:

```
Logistic_regression_model <- glm(Severity ~ BI.RADS.assessment + Age + Shape + Margin + Density,
                                 data = train_data,
                                 family = 'binomial')
```

In our case, *Severity* is the target variable, the one that needs to be predicted, while *BI.RADS.assesment*, *Age, Shape, Margin* and *Density* are the predictors.

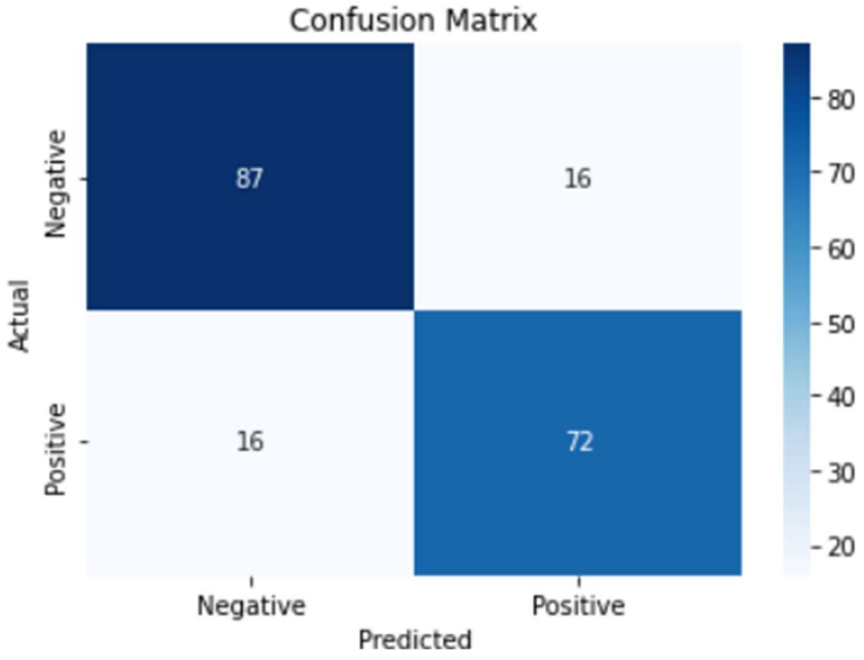
After training this model, one can make predictions on the test set:

```
predictions <- as.factor (as.numeric(predict(logistic_regression_model, newdata = test_data, type = 'response') > 0.5))
```

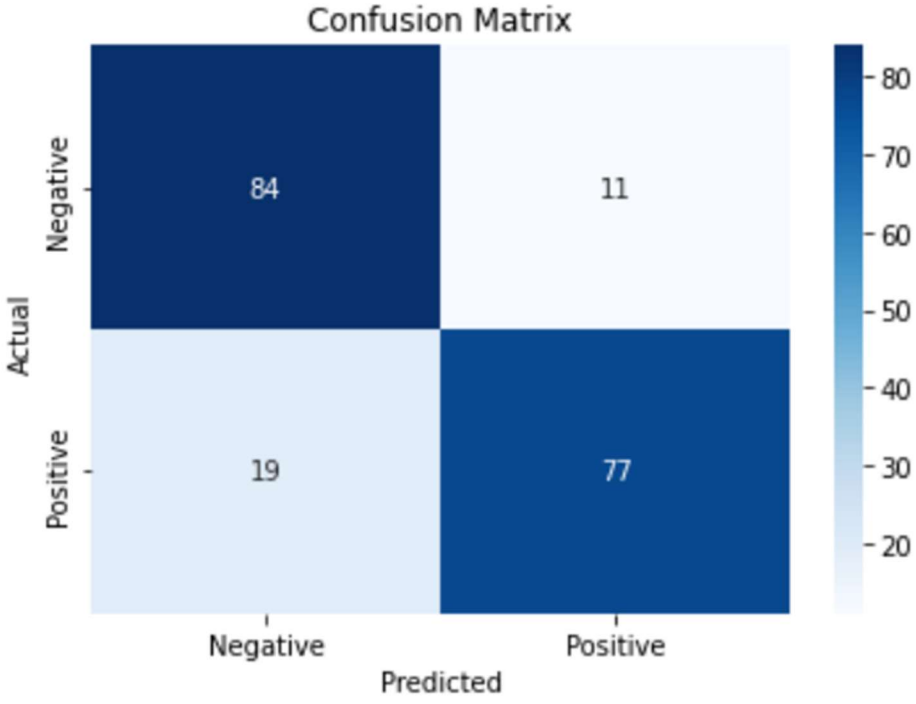
We evaluate the performance of this model using a confusion matrix, defined by:

```
confusion_matrix <- confusionMatrix(predictions,test_data$Severity)
```

For *mice:*



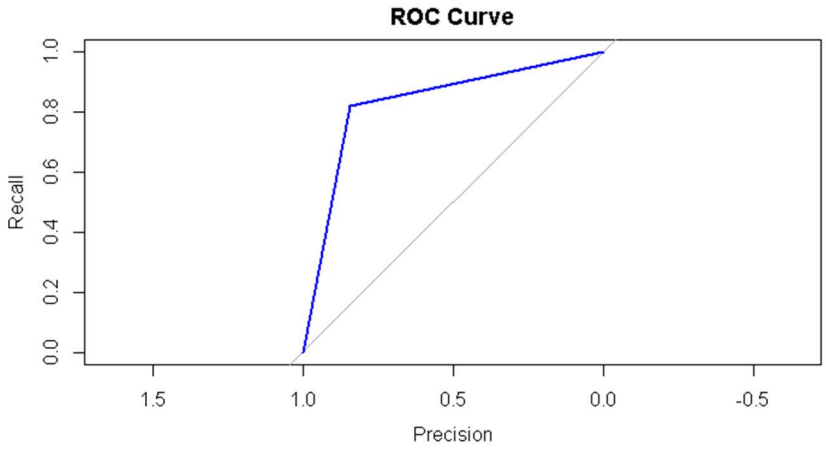
For *missForest:*



This compares the ground truth labels of the test set with the predicted ones. The results were computed:

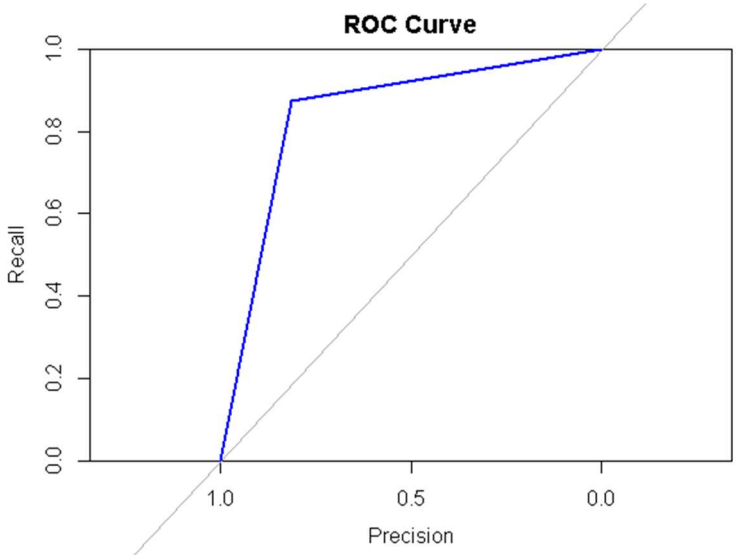
For *mice:*

- Accuracy: 0.8324
- Precision: 0.8181
- Recall: 0.8181
- F1 score: 0.8181



For *missForest:*

- Accuracy: 0.8429
- Precision: 0.875
- Recall: 0.8020
- F1 score: 0.8369



Since we achieved favorable results across various metrics such as accuracy, precision, recall, and F1 score, we can confidently say that our model is not overfitting our data. Instead, it is showing a good performance on our dataset and, again, using *missForest* gave us better results.

Due to the fact that the *glm* function from the base R package uses maximum likelihood estimation, regularization is typically not specifically selected. Therefore, we tried another version of implementing a Logistic Regression model, that would internally perform a cross validation and extract, for our particular case, the best regularization parameter (lambda). This implied importing the *glmet* package.

The implementation can be realised as follows:

```
lr_model <- cv.glmnet(lr_predictors, lr_predicted, alpha = 0, family = "binomial")
```

To extract the best lambda, one can simply compute:

```
best_lambda <- lr_model$lambda.min
```

And print the final result:

```
print(paste("Best Lambda for Logistic Regression:", best_lambda))
```

What we obtained is:

For *mice:*

```
Best Lambda for Regularized Logistic Regression: 0.03138
```

For *missForest:*

```
Best Lambda for Regularized Logistic Regression: 0.0311
```

3. Random Forest Classifier

Random forests are an ensemble of multiple decision trees trained on distinct parts of the same training dataset. This model is used in supervised learning tasks and it is renowned for delivering good results to the classification problem.

To create a Random Forest Classifier in R, one should first install and load the *randomForest* package. This can be achieved with the following commands:

```
install.packages("randomForest")
library(randomForest)
```

In R programming language, one can create such a model using the following code:

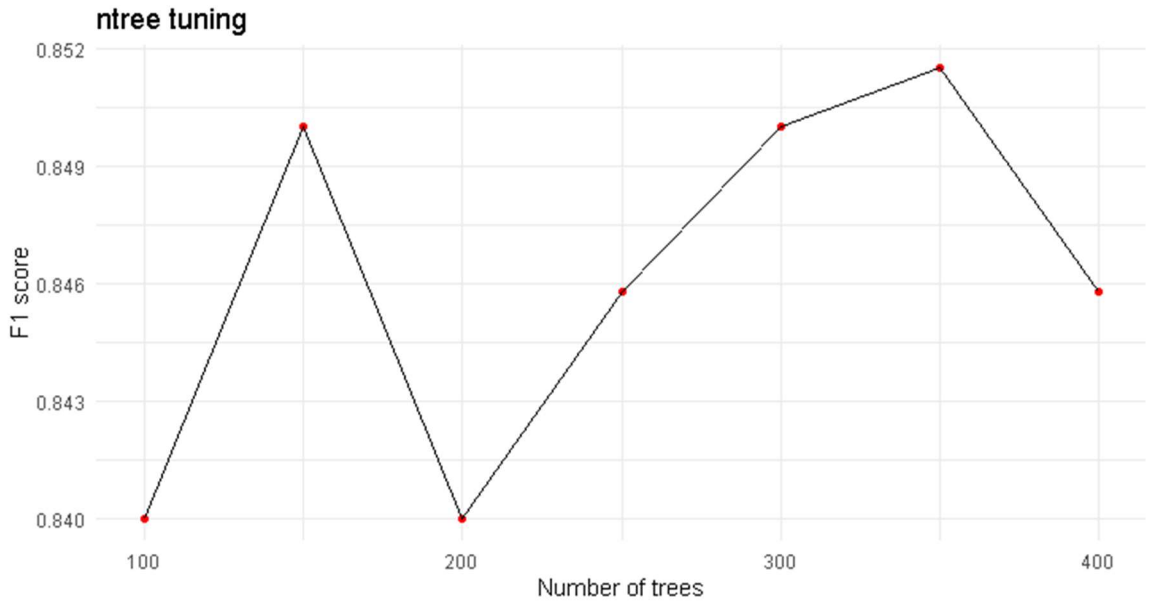
```
random_forest_model <- randomForest(Severity ~ BI.RADS.assessment + Age + Shape + Margin +
Density,
                    data = train_data,
                    ntree = 350 / ntree = 300)
```

As mentioned before, *Severity* is the target variable, the one that needs to be predicted, while *BI.RADS.assesment*, *Age, Shape, Margin* and *Density* are the predictors. We assigned the training dataset to the 'data' argument.

The following tables display the relationships between the number of decision trees and the F1 score. To be more suggestive, based on these tables, we attached the following graphs.

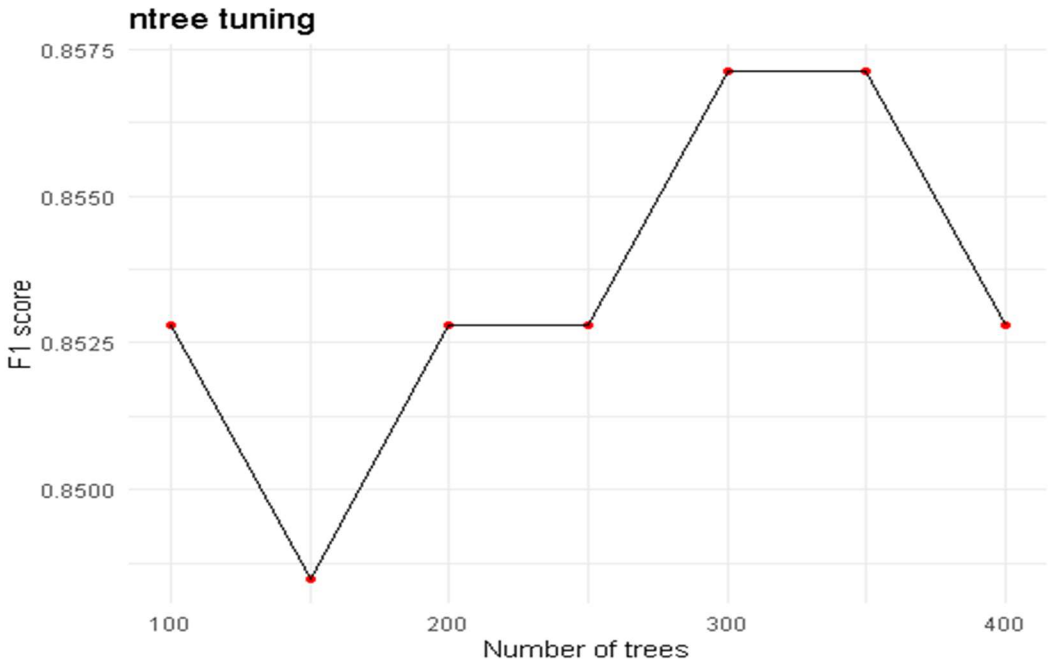
For *mice:*

n tree	100	150	200	250	300	350
F1 score	0.84	0.85	0.84	0.845	0.85	0.851



For *missForest:*

n tree	100	150	200	250	300	350
F1 score	0.852	0.848	0.852	0.852	0.857	0.857



After performing a grid search to find the best hyperparameters for the model, we concluded that the appropriate number of decision trees, determining how many Decision Trees will be included in the final Random Forest model, is set to 'ntree = 350' (for *mice*) and 'ntree = 300' (for *missForest*). In the graphs above, the relationship between the F1 score metric and the selected number of decision trees are illustrated to justify these choices.

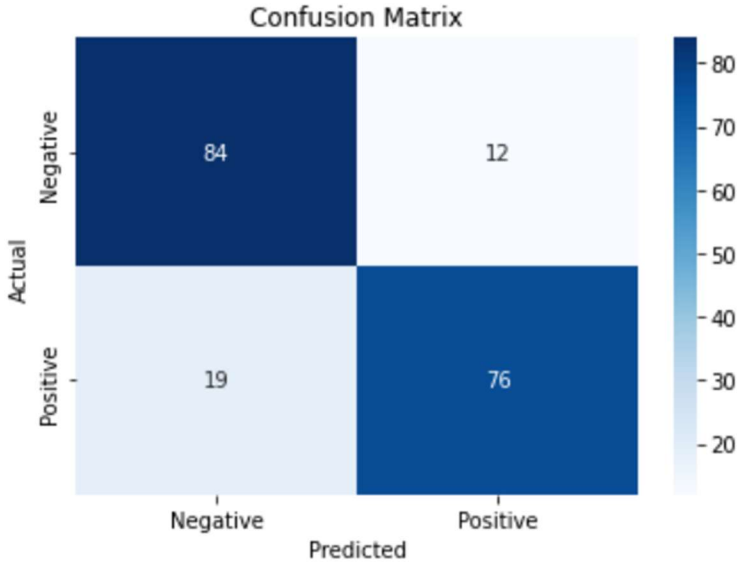
After training this model, one can make predictions on the test set:

```
predictions <- predict(random_forest_model, newdata = test_data)
```

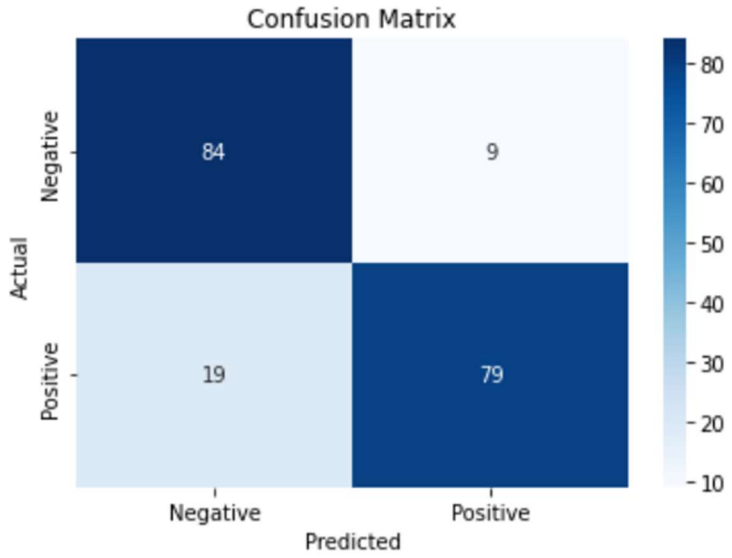
We evaluate the performance of this model using a confusion matrix, defined by:

```
confusion_matrix <- confusionMatrix(predictions, test_data$Severity)
```

For *mice:*



For *missForest:*



This compares the ground truth labels of the test set with the predicted ones. The results were computed:

For *mice:*

- Accuracy: 0.8376
- Precision: 0.8636
- Recall: 0.8
- F1 score: 0.8306

For *missForest:*

- Accuracy: 0.8534
- Precision: 0.8977
- Recall: 0.8061
- F1 score: 0.849

Having observed good values across all metrics (accuracy, precision, recall, F1 score), we can affirm that our model is not overfitting; rather, it is demonstrating a great performance on our dataset, especially for the data that was imputed using *missForest.*

4. Support Vector Machines (SVM)

Support Vector Machines are a class of supervised machine learning algorithms, used for classification and regression tasks. Their particularity is being effective in high-dimensional spaces and are widely used in various applications, including bioinformatics.

To create Support Vector Machines in R, one should first install and load the *e1071* package. This can be achieved with the following commands:

```
install.packages("e1071")
library(e1071)
```

In R, one can create such a model using the following code:

```
svm <- svm(Severity ~ BI.RADS.assessment + Age + Shape + Margin + Density,
                                                    data = train_data,
                                                    kernel = 'radial'
                                                    cost = 10)
```

As we have seen before, *Severity* is the target variable, and *BI.RADS.assesment*, *Age, Shape, Margin* and *Density* are the predictors.

The following tables are illustrating the correlation between the cost parameter of our SVM model and the F1 score metric.

For *mice:*

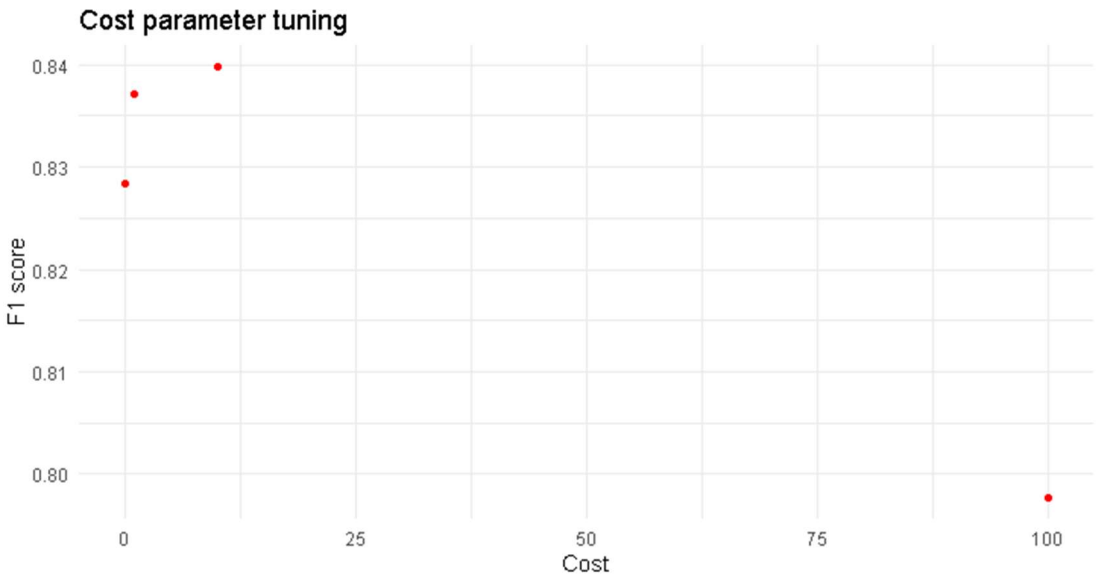
cost	0.1	1	10	100
F1 score	0.8284	0.8372	0.8397	0.7976

For *missForest:*

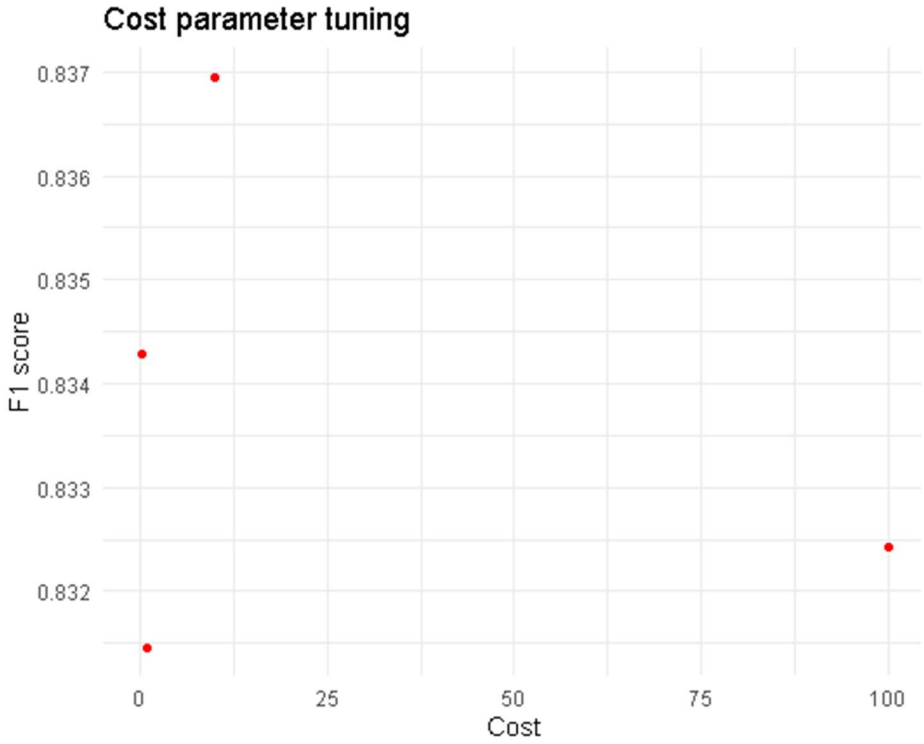
cost	0.1	1	10	100
F1 score	0.8342	0.8314	0.8369	0.8324

Based on these tables, 2 graphs were computed to visualize F1_score = f(cost).

For *mice:*



For *missForest:*



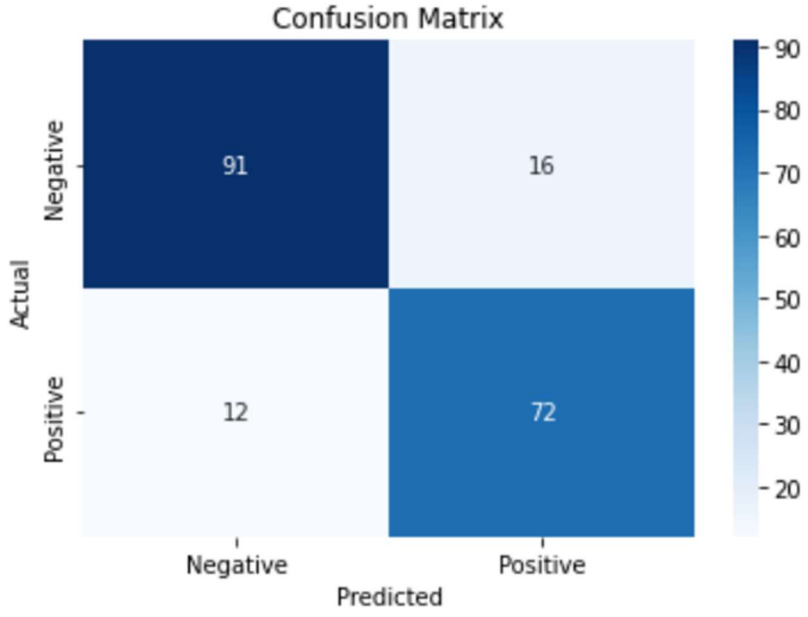
One can make predictions on the test set after training the model:

```
svm_predictions <- predict(svm_model, newdata = test_data)
```

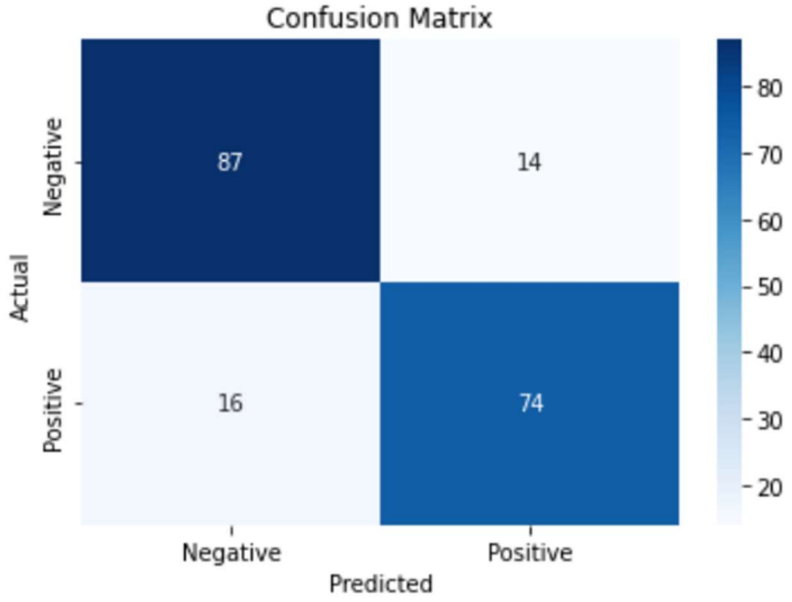
We evaluate the performance of this model using a confusion matrix, defined by:

```
svm_confusion_matrix <- table(Predicted = svm_predictions, Actual = test_data$Severity)
```

For *mice:*



For *missForest:*



This compares the ground truth labels of the test set with the predicted ones. The results were computed:

For *mice:*

- Accuracy: 0.8534
- Precision: 0.8571
- Recall: 0.8181
- F1 score: 0.8372

For *missForest:*

- Accuracy: 0.8429
- Precision: 0.8222
- Recall: 0.8409
- F1 score: 0.8314

Our SVM model is performing admirably across all metrics—accuracy, precision, recall, and F1 score. It is making very good predictions on our datasets.

MODEL COMPARISON

Observing all the metrics resulted from all four models, we can tell that:

For *mice:*

- KNN, being a simple model, obtained the weakest results, compared to the other models, but still got a good performance
- SVM got the best results
- Random Forest was very close to SVM, having slightly weaker results
- Logistic Regression met expectations, catching up with SVM and Random Forest

	Accuracy	**Precision**	**Recall**	**F1 score**
KNN	0.7801	0.75	0.784	0.766
Logistic regression	0.8324	0.8181	0.8181	0.8181
Random Forest Classifier	0.8376	0.8636	0.8	0.8306
SVM	0.8534	0.8571	0.8181	0.8372

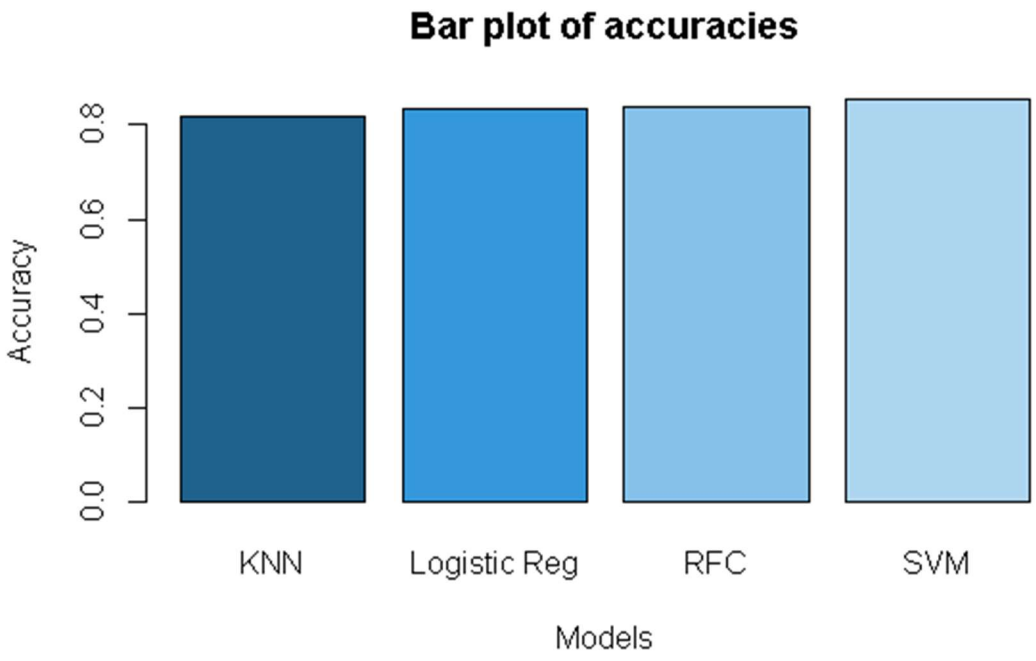
For *missForest:*

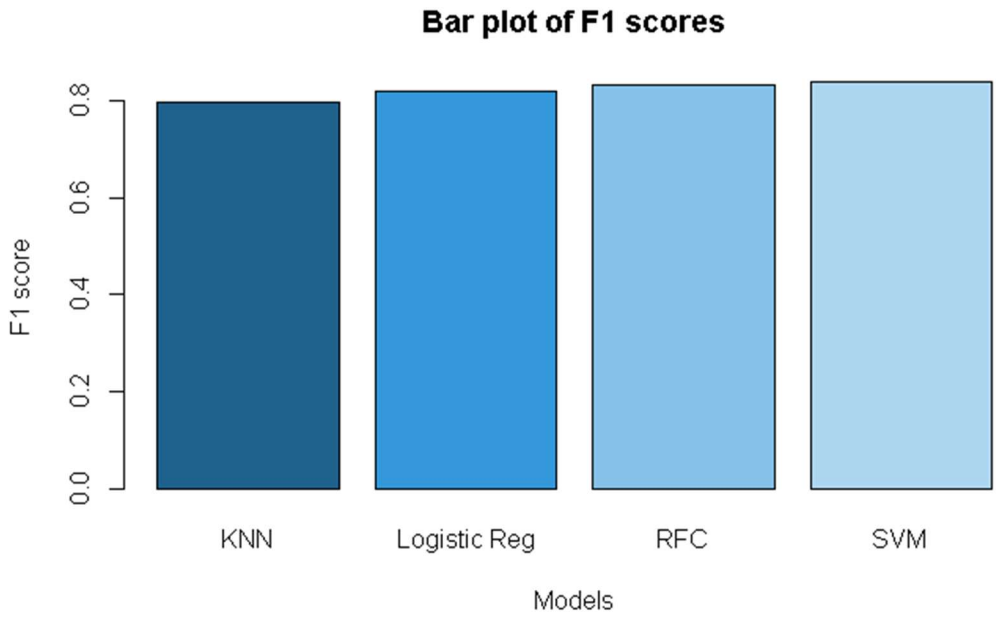
- KNN, being a simple model, obtained the weakest results, compared to the other models, but still got a good performance
- Random Forest Classifier got the best results
- SVM was very close to Random Forest, having slightly weaker results
- Logistic Regression met expectations, catching up with SVM and Random Forest

	Accuracy	**Precision**	**Recall**	**F1 score**
KNN	0.8272	0.777	0.875	0.823
Logistic regression	0.8429	0.875	0.8020	0.8369
Random Forest Classifier	0.8534	0.8977	0.8061	0.849
SVM	0.8429	0.8222	0.8409	0.8314

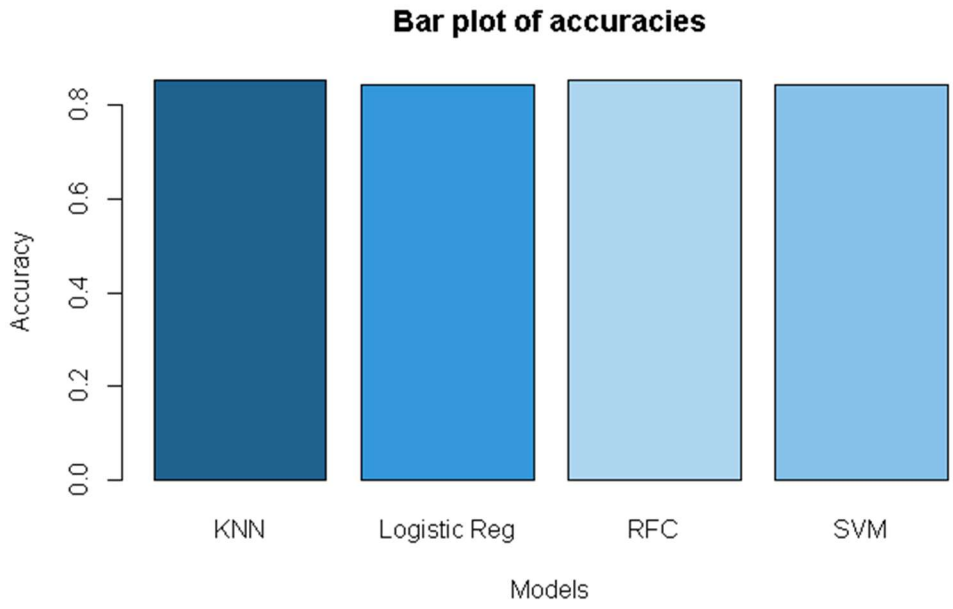
To display our results better, we made some plots, where lighter blue means higher score. These are the plots:

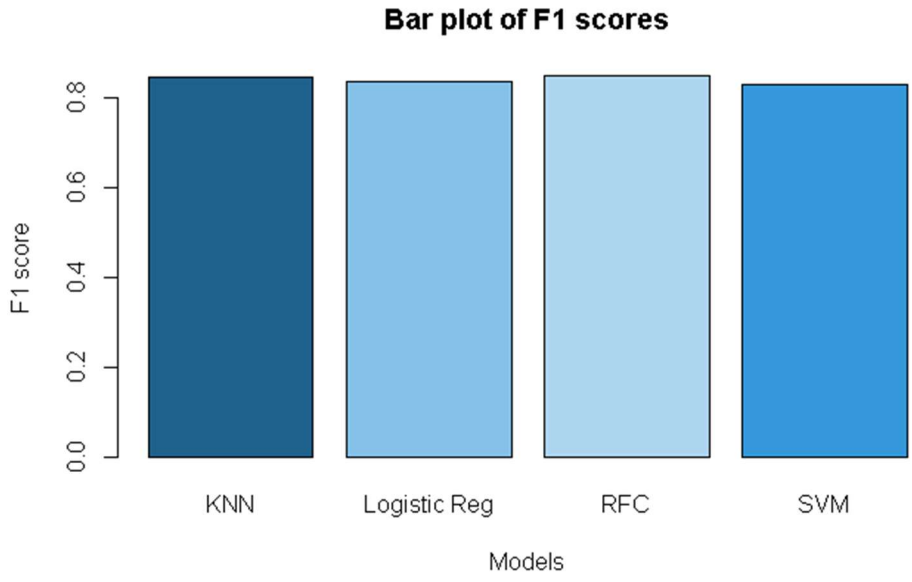
For *mice:*





For *missForest:*





Final Conclusion:

Using *missForest* to impute the missing data resulted in better results, but we were satisfied with the outcomes from *mice* also. The predictions were good and all of the models that we used fitted our datasets admirably.