TMA4268 Project 2 - Titanic - Machine Learning from Disaster

Yawar Mahmood

2023-04-04

Abstract

The purpose of this analysis is to find the best ML model for predicting if someone died or survived the titanic sinking. The methods studied are Logistic Regression, K Nearest Neighbour, Decision Tree and Random Forests. The performance of these are set up against eachother, where the comparison is based on different parameters as ROC, AUC, Sensitivity, Specificity, and F1-scores. The data set is from Kaggle, with different predictor variables, which can give insight into who survived the titanic. There is conducted a descriptive data analysis to begin with, using histograms and correlation matrices, to make a selection of predictors which the models mentioned above would be trained on. From the descriptive analysis the result seems to be that the chance of survival was greater if you had a expensive cabin, was female, with non to small family. From the model prediction, the best performing model is the Random Forest, with a close second in Logistic regression. A surprising finding is that logistic regression does so well. This may indicate that there is a linear relation between the predictors and outcome (this is not examined here, but can be examined to potentially develope better models).

Introduction: Scope and purpose of your project

The sinking of the Titanic is one of the most infamous shipwrecks in history, and resulted in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. The purpose here is using passenger data, to answer the question on "what sort of people were more likely to survive". Thus, this is a classification problem.

The data used in this data analysis was obtained from kaggle.com, specifically from the ongoing competition named "Titanic - Machine Learning from Disaster". The data can be manually found by following the link https://www.kaggle.com/competitions/titanic/data, where it is also possible to obtain information about the structure of the data, the meaning of the different variables, and whether the variables are categorical or quantitative. The dataset can be downloaded by executing the API call "kaggle competitions download -c titanic" in the terminal.

The purpose of this project is to test various classification methods for determining whether an individual perished or survived the sinking of the Titanic. The aim is to identify the method with the best possible performance, and therefore, simpler methods that are easy to understand are not of great interest. In addition, we seek to gain a better understanding of the methods used and compare the various strengths and weaknesses of each method. To determine which variables will be used to train the various models, we will begin with a descriptive data analysis.

The methods that will be examined are:

- Logistic Regression
- K Nearest Neighbor (KNN)
- Decision Tree
- Random Forest

Descriptive data analysis/statistics

We now want to get a general overview of the data, where the goal is to understand the structure of the data and identify which variables may be of significant importance in classifying who survived or perished.

The first step is to read the data, and remove any irrelevant or incorrect information so that the data can be used for further analysis. The data cleaning process done here is very simple and involves only removing incomplete observations from the data set.

From the training data set, 177 of 891 incomplete records are removed.

```
data <- read.csv('../Downloads/titanicData/train.csv', stringsAsFactors = FALSE)

#dim(data)

data <- na.omit(data)

#dim(data)
```

We begin by categorizing the data set into two types of variables: continuous and categorical.

The continuous variables are: Age, SibSp, Parch, and Fare.

The categorical variables are: Survived, Pclass, Sex, Ticket, Cabin, and Embarked.

What these variables mean is explanied in more detail in https://www.kaggle.com/competitions/titanic/data, as mentioned in the introduction.

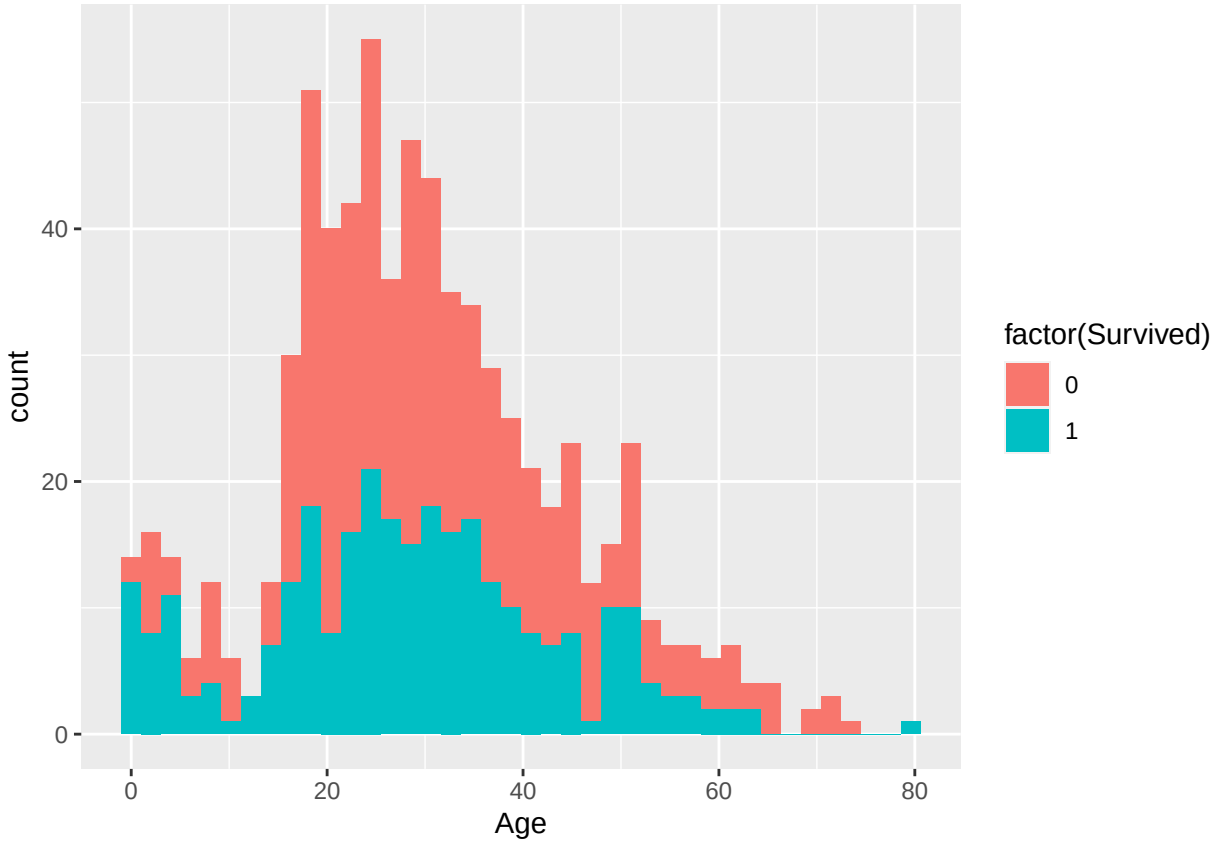
Our initial focus is on analyzing the continuous variables. We present a histogram analysis of these variables, accompanied by a brief commentary on the distribution of the data and any observed relationships between the variables and the survival outcome.

```
numVars <- c("Age", "SibSp", "Parch", "Fare")
numSubsetData <- data[, numVars]

catVars <- c("Survived", "Pclass", "Sex", "Ticket", "Cabin", "Embarked")
catSubsetData <- data[, catVars]
```

Age

```
# hist of Age
hist.Age <- ggplot(data, aes(Age, fill = factor(Survived))) + geom_histogram(bins = 40)
hist.Age
```




```
summary(data$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.42   20.12   28.00   29.70   38.00   80.00
```

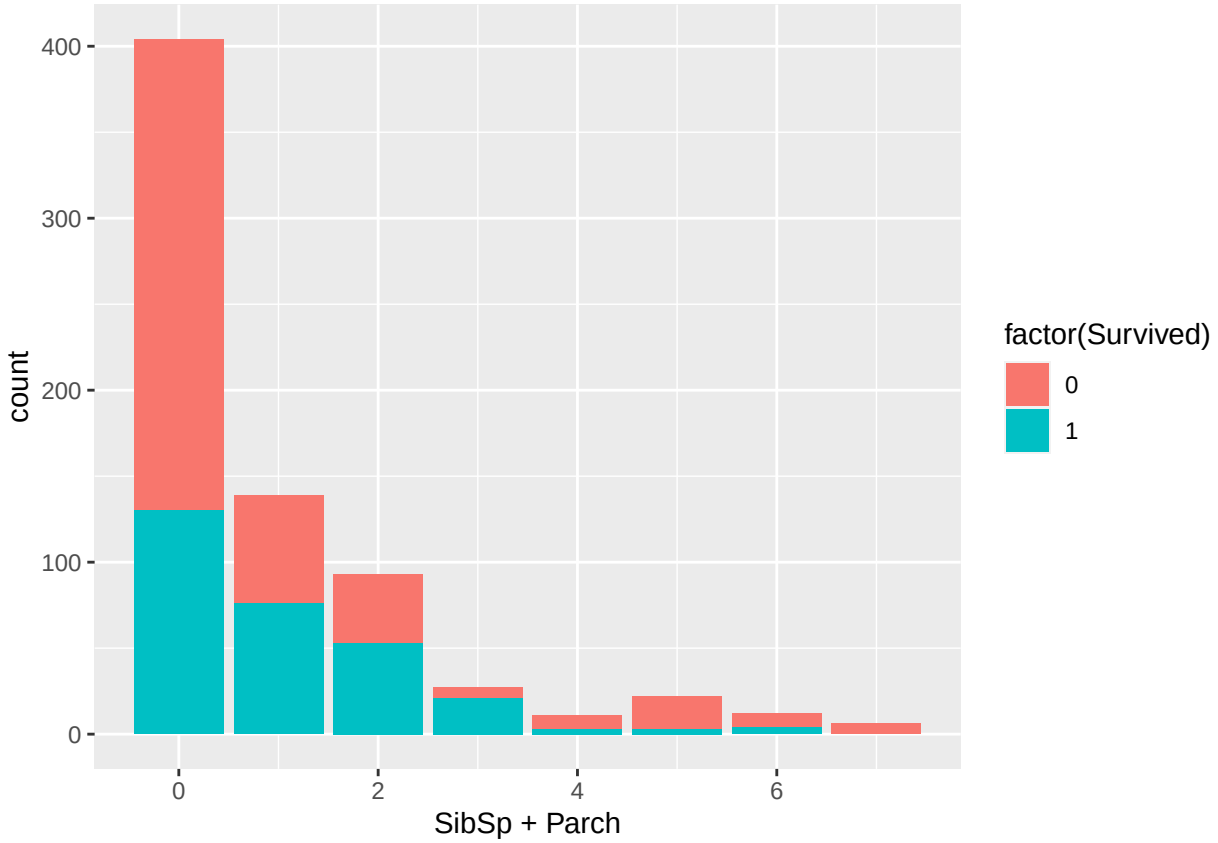
The age appears to follow a somewhat normal distribution, with a mean age of 29.70. The majority of deaths occurred in the adult age group [18-40]. This may indicate, even though there were more less children and elderly individuals on-board, that they were prioritized in the evacuation process. Age is likely a significant variable in our analysis.

SibSp and Parch (family relations)

```
# hist of SibSp
hist.SibSpNParch <- ggplot(data, aes(SibSp + Parch, fill = factor(Survived))) + geom_histogram(stat = "
```

```
## Warning in geom_histogram(stat = "count"): Ignoring unknown parameters:
## `binwidth`, `bins`, and `pad`
```

```
hist.SibSpNParch
```




```
summary(data$SibSp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.5126  1.0000  5.0000
```

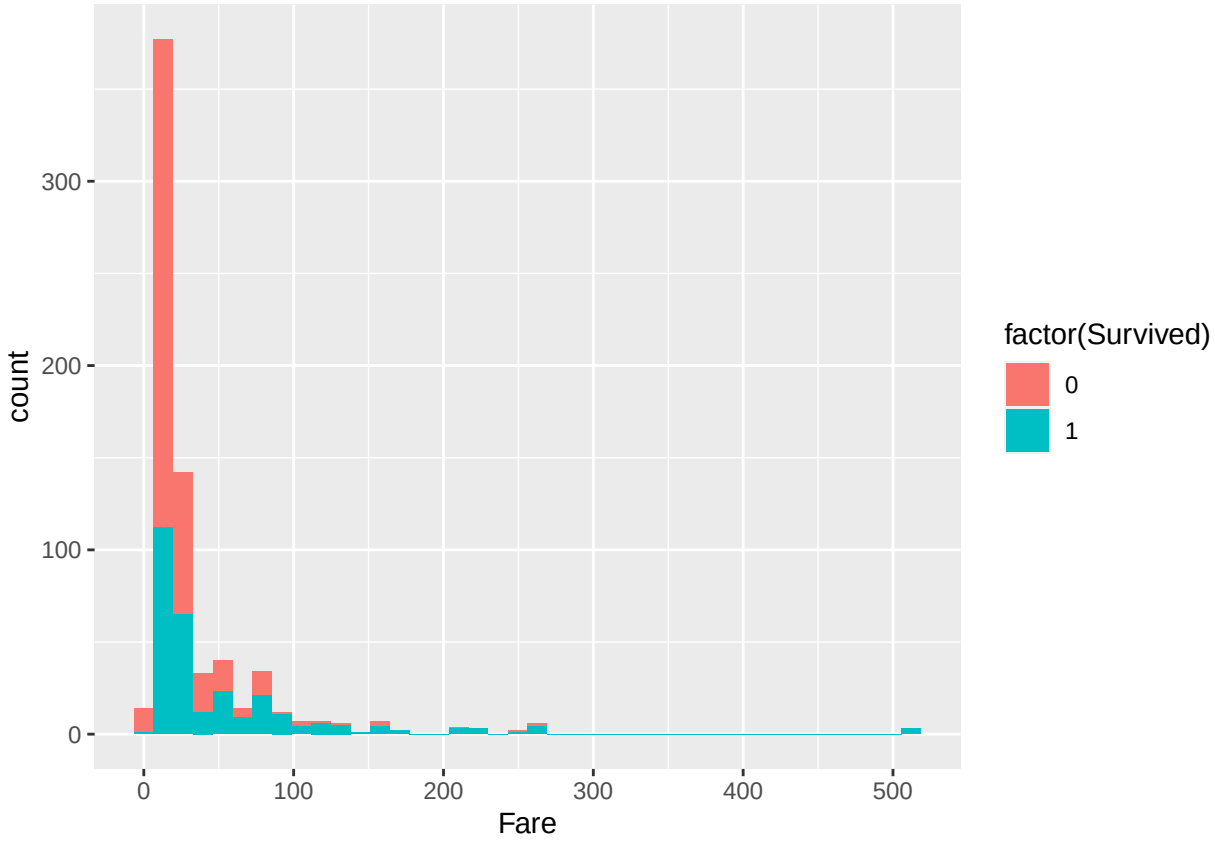
```
summary(data$Parch)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4314  1.0000  6.0000
```

It should be noted that we have combined the variables Parch and SibSp in this histogram, as they are of a similar nature. This is particularly evident from the dataset description on Kaggle, as both variables relate to family relationships. It appears that the more family relationships an individual had, the less likely they were to survive. This seems reasonable, as people may not want to leave without their family. It is possible that people were searching for their family members, which led to their death. This trend seems to be particularly true for families with a size larger than 4. Therefore, Parch and SibSp are likely significant variables in our analysis. The data does not seem to follow any particular distribution.

Fare

```
# hist of Fare
hist.Fare <- ggplot(data, aes(Fare, fill = factor(Survived))) + geom_histogram(bins = 40)
hist.Fare
```




```
summary(data$Fare)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    8.05   15.74   34.69   33.38  512.33
```

The average ticket price was $34.69, with the majority of passengers paying a relatively low fare. The histogram suggests that passengers who paid higher ticket prices were more likely to survive. Thus, ticket price appears to be a significant variable in determining survival. The data does not seem to follow any particular distribution.

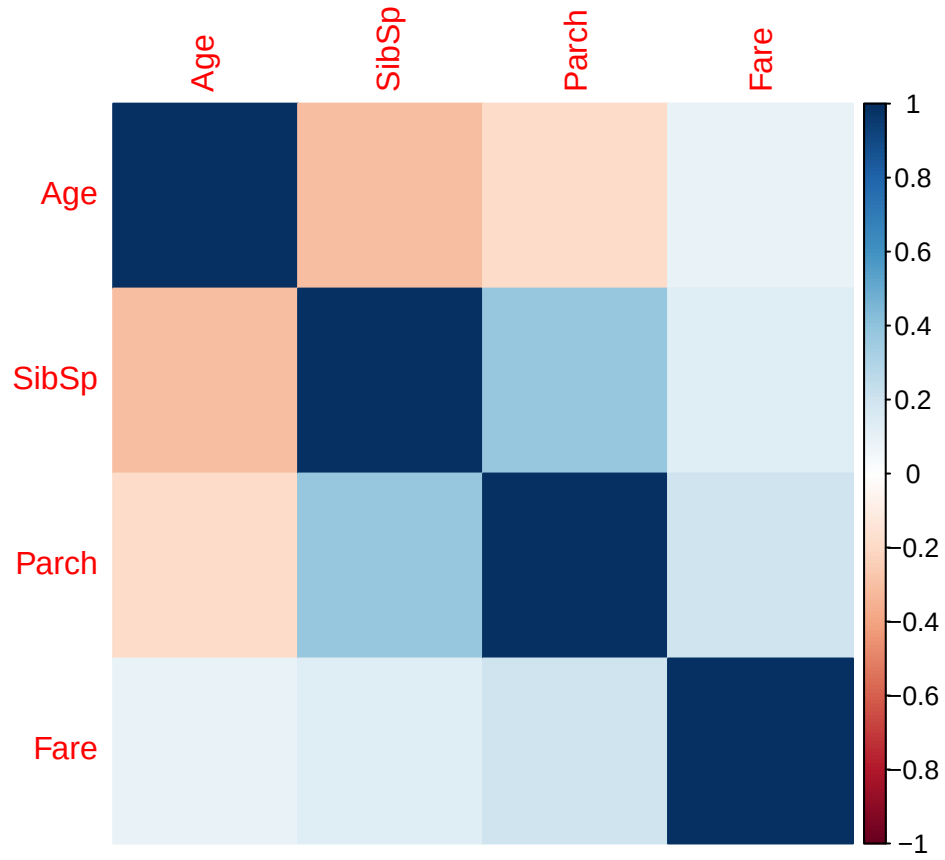
Correlation continous (numerical) variables

We now want to study the correlation between the different variables. This can help support the analysis we have already done and shed light on something we may have overlooked.

```
# Looking at some correlations for our numerical value
corrMatrixNum <- cor(numSubsetData)
corrMatrixNum
```

```
##                Age      SibSp      Parch       Fare
## Age    1.00000000 -0.3082468 -0.1891193 0.09606669
## SibSp -0.30824676  1.0000000  0.3838199 0.13832879
## Parch -0.18911926  0.3838199  1.0000000 0.20511888
## Fare   0.09606669  0.1383288  0.2051189 1.00000000
```

```
corrplot(corrMatrixNum, method = "color")
```



For example, we can see that Parch and SibSp have a high positive correlation, indicating that families tend to travel together, so combining Parch and SibSp was a reasonable decision. Age, SibSp, and Parch seem to have a negative correlation, indicating that children usually did not travel alone. As for Fare, there does not seem to be any significant correlation with the other variables.

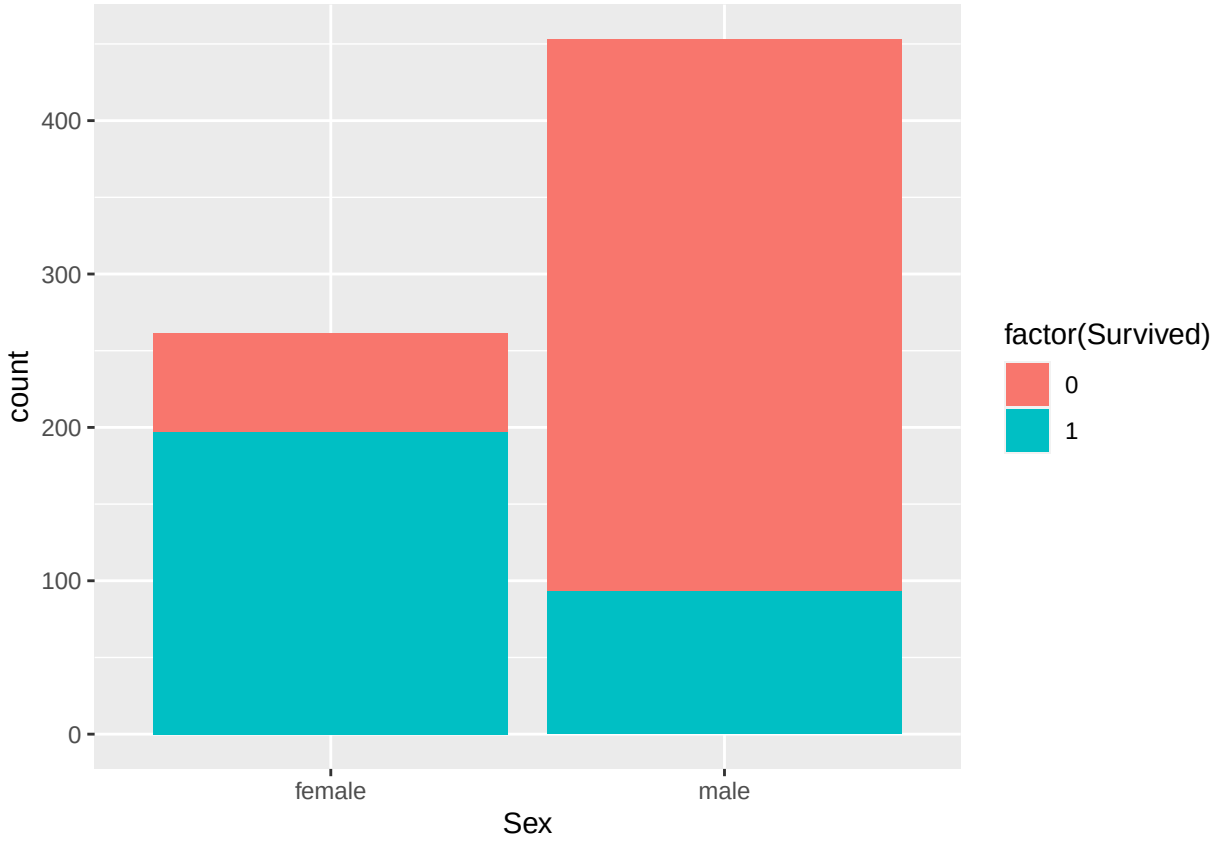
We will now move on to our categorical variables and perform a corresponding analysis for them.

Sex

```
#hist of Sex
hist.Sex <- ggplot(data, aes(Sex, fill = factor(Survived))) + geom_histogram(stat = "count")
```

```
## Warning in geom_histogram(stat = "count"): Ignoring unknown parameters:
## `binwidth`, `bins`, and `pad`
```

```
hist.Sex
```



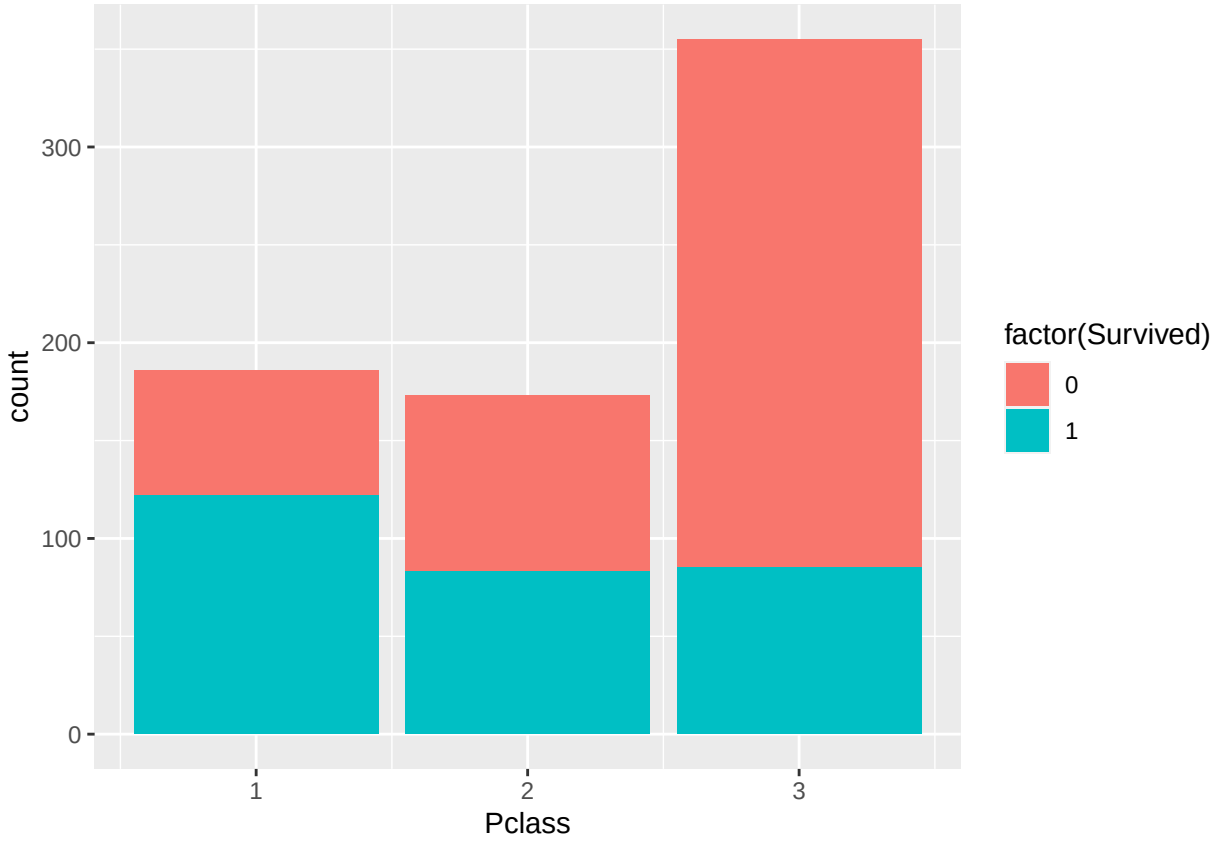
The distribution shows that most of the passengers on board were men. It also appears that the percentage of men who died was much higher than the percentage of women who died. This suggests that women were prioritized over men during the evacuation. When considered alongside the continuous variable of age, this indicates that the statement "Women and children first," also known as the Birkenhead drill, was followed. Sex is likely a significant variable in our analysis.

Pclass

```
#hist of Pclass
hist.Pclass <- ggplot(data, aes(Pclass, fill = factor(Survived))) + geom_histogram(stat = "count")
```

```
## Warning in geom_histogram(stat = "count"): Ignoring unknown parameters:
## `binwidth`, `bins`, and `pad`
```

```
hist.Pclass
```



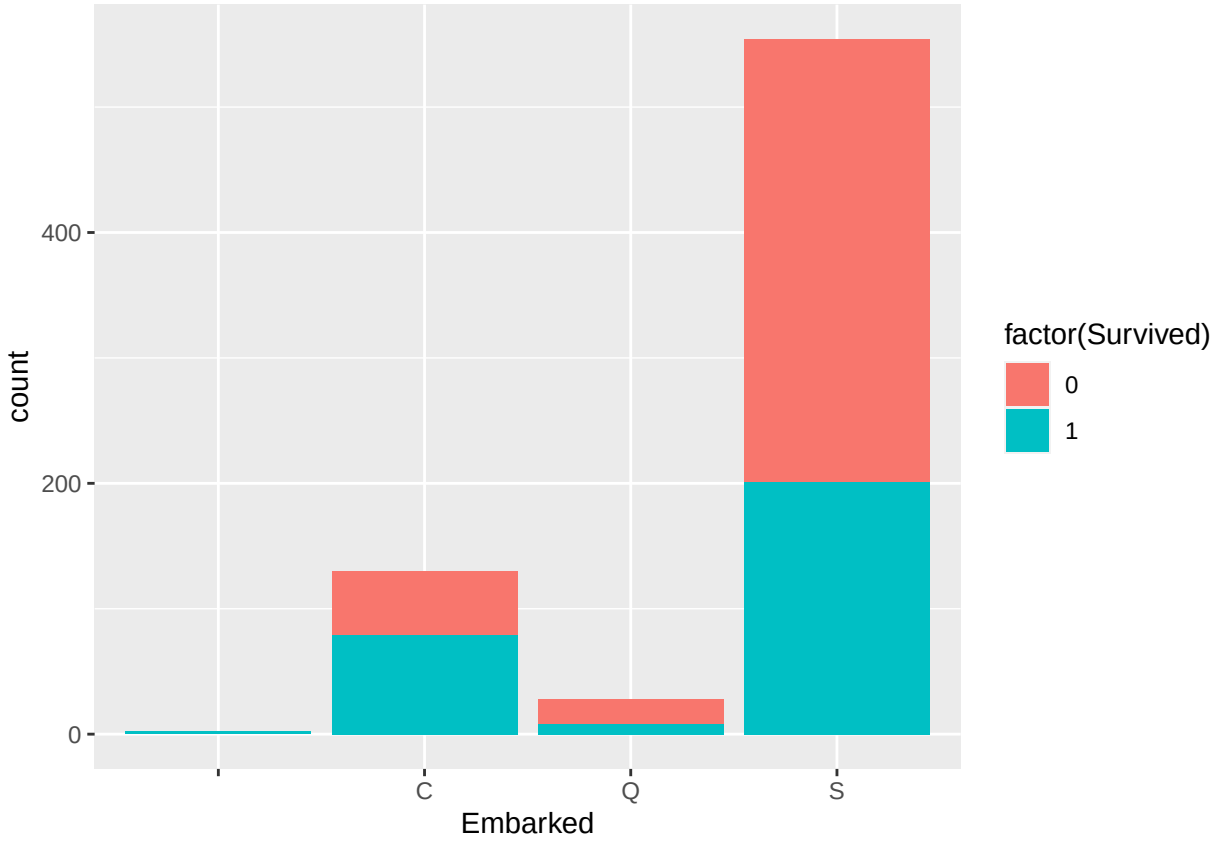
It appears that most of the passengers were in 3rd class, while the number in 1st and 2nd class was fairly similar. This corresponds well with Fare, as most paid a low price for their tickets. The percentage of deaths was higher for 3rd class than for 2nd and 1st class. The class in which people were traveling seems to be a significant factor in our analysis. This also corresponds well with our Fare variable, as most of those who died paid a low price for their ticket. Pclass is likely a significant variable in our analysis.

Embarked

```
#hist of embarked
hist.Embarked <- ggplot(data, aes(Embarked, fill = factor(Survived))) + geom_histogram(stat = "count")
```

```
## Warning in geom_histogram(stat = "count"): Ignoring unknown parameters:
## `binwidth`, `bins`, and `pad`
```

```
hist.Embarked
```



It appears that most of the passengers on the Titanic were from S = Southampton, which is also where the journey began. Q = Queenstown and C = Cherbourg had significantly fewer passengers. The percentage of deaths does not seem to be significant for the different locations, and this variable is likely of little importance in our analysis.

Ticket and Cabin

We are disregarding the Ticket and Cabin variables for now, as these have a different format compared to the variables we have studied so far, and therefore it is difficult to extract the data in a meaningful way.

Methods

We will now train 5 different methods

- Logistic Regression
- K Nearest Neighbor (KNN)
- Decision Tree
- Random Forest.

```
set.seed(25935)

# Run algorithms using 10-fold cross validation
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"
```

The reason we use 10-fold cross-validation is to evaluate the performance of the models in a robust and unbiased way. In 10-fold cross-validation, the data is randomly divided into 10 equal parts, and the algorithm is trained and evaluated 10 times. In each iteration, 9 parts of the data are used for training the algorithm, and the remaining part is used for testing its performance. This process is repeated 10 times, with each part serving as the test set once. The model with the best performance is returned, and is the model we use in further analysis.

```
set.seed(123)

varsToKeep <- c("Survived", "Age", "Fare", "Sex", "SibSp", "Pclass", "Parch")
dataSubset <- data[varsToKeep]

# Split the data into a 80% training set and a 20% test set
trainIndex <- createDataPartition(dataSubset$Survived, p = 0.8, list = FALSE)
trainData <- dataSubset[trainIndex,]
testData <- dataSubset[-trainIndex,]

# convert Survived to factor
trainData$Survived <- as.factor(trainData$Survived)
testData$Survived <- as.factor(testData$Survived)
```

From the descriptive data analysis, it is evident that Age, Fare, Sex, SibSp, Pclass and Parch are predictors of high significance. There, we will only look at these predictors in further analysis

We aim to divide our data into two data sets: a training data set that will be used to train the models, and a testing data set that will be used to evaluate the models performance and compare them. The partition between these data sets are 80%/20%, i.e, 80% of the original data set is the training set and 20% is the test set.

Logistic Regression

```
set.seed(25935)
# logistic regression
fit.glm <- train(Survived ~ ., data=trainData, method="glm", family = "binomial")
```

The logistic regression is a S shaped curve that maps input to a probability value between 0 and 1. The model uses a logistic function to transform a linear combination of the predictor variables into a probability.

This probability is given by

$$p = \frac{1}{1 + e^{-t}}$$

where $t = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots$. The values for the different $\beta - s$ are given in the summary of the logistic regression model. If the predicted probability p is greater than or equal to a threshold value, which in our case is set to 0.5, the observation is classified as belonging to the positive class, otherwise, it is classified as belonging to the negative class.

Some strengths of the model:

- Logistic regression is a straightforward and easy to understand model. In our case, this is of no/small importance.

- The coefficients of logistic regression are very interpretable, and can be easily used to perform t-tests and F-tests. In our case, this is of no/small importance.
- The model is pretty robust, and can perform well with even outliers in the data. It also performs relatively good, even though the assumptions are not necessarily fully met. This is an advantage for performance, and thus, is a good thing for our analysis.

Some weaknesses of the model:

- The model assumes a linear relationship between the predictor variables and the outcomes, which is not known is true or false for our data. The model may therefore not be to suitable for more complex linear relationships.
- Logistic regression may suffer from overfitting when there are too many predictors relative to the number of observations. This is not a problem in our case.

K Nearest Neighbor (KNN)

```
set.seed(25935)

# KNN
k.values <- data.frame(k = 1:10)

fit.knn <- train(Survived ~ ., data=trainData, method="knn", metric=metric, trControl=control, tuneGrid
```

KNN works by assuming that similar observations exists in close proximity. Given an observation, the algorithm finds the K nearest neighbors of the new point from the training data, based on Euclidean distance. The predicted value of the new observation is then determined by taking the majority vote of the K nearest neighbors.

The K is a hyperparameter that is to be determined, and in our case, we want to chose k such that we maximize the accuracy. It is hard to find a good k value. A small k means that the prediction is based on only one nearest neighbor, which can lead to over fitting and poor generalization. A large k means that the prediction is based on many neighbors, which can lead to under fitting and oversimplification.

Some strengths of the model:

- KNN does not make any assumptions about the distribution of the data, which can be an advantage, as the data we work with is unknown and relatively complex. In our case, where we do not have alot of knowledge of our data set, this is an advantage.
- KNN performs localized predictions, which can be useful when there are local patterns in the data. This is not known for us in our data, but this may be an advantage.

Some weaknesses of the model:

- It can be computationally expensive for large datasets, and the choice of K can have a significant impact on the performance of the algorithm. It is also hard to tune this hyperparameter. As the performance is highly dependent on k, and we want the best performing model, this is a disadvantage for us.
- KNN is sensitive to outliers, which can signifiantly affect the model.

Decision Tree

```
set.seed(25935)

# Decision Tree

fit.tree <- train(Survived ~ ., data=trainData, method="rpart", metric=metric, trControl=control)
```

Decision Tree is a simple yet powerful algorithm that is used for classification problems. The basic idea of the algorithm is by splitting the data into smaller and smaller subsets based on most significant variables in the data that can predict the value.

The process starts with chosing the most significant variable as a root. This feature is then used to split the data into two or more subsets. The process of selecting the most significant feature continues in this manner, until the tree is fully grown, and there are no subsets to split.

Once the decision tree is built, it can be used to make predictions by following the path of the tree from the root to a leaf, where a prediction is made.

Some strengths of the model:

- They are easy to visualize and interpret, even though the model is pretty complex. This holds no to small value in our case.
- As KNN, they do not make any assumption about the data. This is an advantage in our case, as we do not have alot of knowledge of our relationships in our data set.
- Decision trees can automatically select the most relevant features to split on.

Some weaknesses of the model:

- They can be unstable, as small changes to the data can result in completely different tree structures. This is a disadvantage, as performance can be dependent on randomness in which partition of the data the model is trained on.
- Can easily overfit the training data if the tree is too complex, leading to poor generalization. This does not seem to be a problem in our case.

Random Forest

```
set.seed(25935)

# Random Forest

fit.forest <- train(Survived ~ ., data=trainData, method="rf", metric=metric, trControl=control)
```

Random forest is a "generalization" of the decision tree method, as it makes a multitude of decision trees and then combines their predictions to obtain a final prediction.

The algorithm begins by bootstrapping the data to create multiple subsets of the data, that are used to train individual decision trees. The algorithm then constructs a decision tree for each bootstrap sample using a random subset of the features. Once all the decision trees have been trained, their predictions are combined to make a final prediction. The final prediction is done in the form of a majority vote.

Some strengths of the model:

- Robust to noise and outliers in the data, which makes the algorithm suitable for many different data sets
- Even though the model is complex, it can give a measure of importance of each feature in the data set. This can give an understanding of the underlying relationships in the data. This is of small to no importance for our purposes.

Some weaknesses of the model:

- They can be difficult to interpret and visualize due to their complexity. This is of small to no importance for our purposes.
- They are demanding to compute. This is of small to no importance for our purposes.

Results and interpretation

We now wish to evaluate how our models perform when we run them on our test data set. We will then present the models' sensitivity, specificity, ROC curves, AUC values, confusion matrices, and F1-scores. These metrics will be used to compare the models and find the best performing model. We will also present some aspects of the models, such as tuning parameters.

Under follows the Sensitivity, Specificity and F1-scores of the models:

```
# predictions on train data
predictionsGLM <- predict(fit.glm, newdata = testData)
# confusion matrix
confusionMatrixGLM <- confusionMatrix(predictionsGLM, testData$Survived)

sensitivityGLM <- confusionMatrixGLM$table[2,2]/(confusionMatrixGLM$table[2,2]+confusionMatrixGLM$table
specificityGLM <- confusionMatrixGLM$table[1,1]/(confusionMatrixGLM$table[1,1]+confusionMatrixGLM$table

precisionGLM <- confusionMatrixGLM$table[2,2]/(confusionMatrixGLM$table[2,2] + confusionMatrixGLM$table
recallGLM <- confusionMatrixGLM$table[2,2]/(confusionMatrixGLM$table[2,2] + confusionMatrixGLM$table[2,
F1GLM <- 2*(precisionGLM*recallGLM)/(precisionGLM + recallGLM)

# predictions on train data
predictionsKNN <- predict(fit.knn, newdata = testData)
# confusion matrix
confusionMatrixKNN <- confusionMatrix(predictionsKNN, testData$Survived)

sensitivityKNN <- confusionMatrixKNN$table[2,2]/(confusionMatrixKNN$table[2,2]+confusionMatrixKNN$table
specificityKNN <- confusionMatrixKNN$table[1,1]/(confusionMatrixKNN$table[1,1]+confusionMatrixKNN$table

precisionKNN <- confusionMatrixKNN$table[2,2]/(confusionMatrixKNN$table[2,2] + confusionMatrixKNN$table
recallKNN <- confusionMatrixKNN$table[2,2]/(confusionMatrixKNN$table[2,2] + confusionMatrixKNN$table[2,
F1KNN <- 2*(precisionKNN*recallKNN)/(precisionKNN + recallKNN)

# predictions on train data
predictionsTREE <- predict(fit.tree, newdata = testData)
# confusion matrix
confusionMatrixTREE <- confusionMatrix(predictionsTREE, testData$Survived)

sensitivityTREE <- confusionMatrixTREE$table[2,2]/(confusionMatrixTREE$table[2,2]+confusionMatrixTREE$ta
```

```
specificityTREE <- confusionMatrixTREE$table[1,1]/(confusionMatrixTREE$table[1,1]+confusionMatrixTREE$ta

precisionTREE <- confusionMatrixTREE$table[2,2]/(confusionMatrixTREE$table[2,2] + confusionMatrixTREE$ta
recallTREE <- confusionMatrixTREE$table[2,2]/(confusionMatrixTREE$table[2,2] + confusionMatrixTREE$table
F1TREE <- 2*(precisionTREE*recallTREE)/(precisionTREE + recallTREE)

# predictions on train data
predictionsFOREST <- predict(fit.forest, newdata = testData)
# confusion matrix
confusionMatrixFOREST <- confusionMatrix(predictionsFOREST, testData$Survived)

sensitivityFOREST <- confusionMatrixFOREST$table[2,2]/(confusionMatrixFOREST$table[2,2]+confusionMatrixF
specificityFOREST <- confusionMatrixFOREST$table[1,1]/(confusionMatrixFOREST$table[1,1]+confusionMatrixF

precisionFOREST <- confusionMatrixFOREST$table[2,2]/(confusionMatrixFOREST$table[2,2] + confusionMatrixF
recallFOREST <- confusionMatrixFOREST$table[2,2]/(confusionMatrixFOREST$table[2,2] + confusionMatrixFORE
F1FOREST <- 2*(precisionFOREST*recallFOREST)/(precisionFOREST + recallFOREST)

sensitivityNspecificityDF <- data.frame(
  Model = c("Logistic Regression: ", "KNN: ", "Decision Tree: ", "Random Forest: "),
  Sensitivity = c(sensitivityGLM, sensitivityKNN, sensitivityTREE, sensitivityFOREST),
  Specificity = c(specificityGLM, specificityKNN, specificityTREE, specificityFOREST),
  F1 = c(F1GLM, F1KNN, F1TREE, F1FOREST)
)

kable(sensitivityNspecificityDF, caption = "Sensitivity, Specificity and F1-score table")
```

Table 1: Sensitivity, Specificity and F1-score table

Model	Sensitivity	Specificity	F1
Logistic Regression:	0.8823529	0.8021978	0.7894737
KNN:	0.6181818	0.6666667	0.5762712
Decision Tree:	0.8490566	0.7977528	0.7758621
Random Forest:	0.9574468	0.8105263	0.8181818

And the confusion matrices:

Logistic Regression

```
confusionMatrixGLM$table
```

```
##            Reference
## Prediction  0  1
##          0 73 18
##          1  6 45
```

KNN

```
confusionMatrixKNN$table
```

```
##           Reference
## Prediction  0  1
##          0 58 29
##          1 21 34
```

Decision Tree

```
confusionMatrixTREE$table
```

```
##           Reference
## Prediction  0  1
##          0 71 18
##          1  8 45
```

Random Forest

```
confusionMatrixFOREST$table
```

```
##           Reference
## Prediction  0  1
##          0 77 18
##          1  2 45
```

And the ROC plots:

```
# roc
predictionsGLMprob <- predict(fit.glm, newdata = testData, type = "prob")
predictionsGLMpositive <- predictionsGLMprob$"1"
roc.glm <- roc(testData$Survived, as.numeric(predictionsGLMpositive))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
# roc
predictionsKNNprob <- predict(fit.knn, newdata = testData, type = "prob")
predictionsKNNpositive <- predictionsKNNprob$"1"
roc.knn <- roc(testData$Survived, as.numeric(predictionsKNNpositive))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

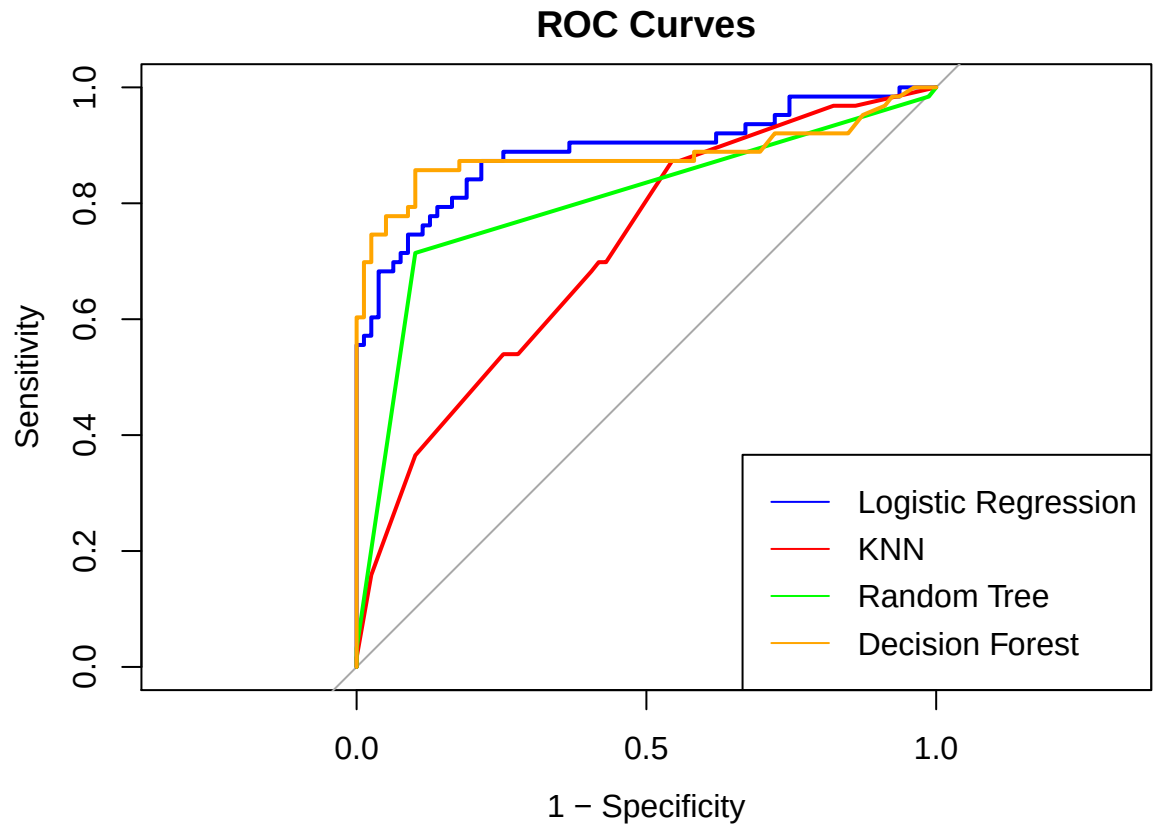
```r
# roc
predictionsTREEprob <- predict(fit.tree, newdata = testData, type = "prob")
predictionsTREEpositive <- predictionsTREEprob$"1"
roc.tree <- roc(testData$Survived, as.numeric(predictionsTREEpositive))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
# roc
predictionsFORESTprob <- predict(fit.forest, newdata = testData, type = "prob")
predictionsFORESTpositive <- predictionsFORESTprob$"1"
roc.forest <- roc(testData$Survived, as.numeric(predictionsFORESTpositive))
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
plot(roc.glm, col = "blue", print.thres = FALSE, legacy.axes = TRUE, main = "ROC Curves")
lines(roc.knn, col = "red")
lines(roc.tree, col = "green")
lines(roc.forest, col = "orange")
legend("bottomright", legend = c("Logistic Regression", "KNN", "Random Tree", "Decision Forest"), col =
```



And the AUC values:

```r
aucDF <- data.frame(
  Model = c("Logistic Regression: ", "KNN: ", "Decision Tree: ", "Random Forest: "),
  AUC = c(auc(roc.glm), auc(roc.knn), auc(roc.tree), auc(roc.forest))
)

kable(aucDF, caption = "AUC table")
```

Table 2: AUC table

Model	AUC
Logistic Regression:	0.8890898
KNN:	0.7185051
Decision Tree:	0.8027928
Random Forest:	0.8824593

On the models

Logistic Regression

```r
summary(fit.glm)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6613  -0.6809  -0.4050   0.6612   2.3800
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.916853   0.638974   7.695 1.42e-14 ***
## Age         -0.039041   0.008775  -4.449 8.62e-06 ***
## Fare         0.002123   0.002519   0.843   0.3992
## Sexmale     -2.568454   0.239365 -10.730  < 2e-16 ***
## SibSp       -0.344772   0.137881  -2.501   0.0124 *
## Pclass      -1.139602   0.174466  -6.532 6.49e-11 ***
## Parch       -0.082098   0.129821  -0.632   0.5271
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 768.44  on 571  degrees of freedom
## Residual deviance: 522.26  on 565  degrees of freedom
## AIC: 536.26
##
## Number of Fisher Scoring iterations: 5
```

We see that the different predictor variables have small p-values, besides Parch, SibSp and Fare. This may indicate that the predictors Parch, SibSp and Fare are not as important as they first seemed. Either way, this is beyond the scope of our analysis.

We can note the efficiency of the model, by looking at how many Fisher Scoring iterations it took to find the maximum likelihood estimators for the different predictor variables. It took only 5 iterations, which indicates that the model is very effective to calculate and stable.

We can also look at the AIC (Akaike Information Criterion) to say something about how good the model fits the data. A lower AIC value indicates a better model fit. In our case, this value is 522.26, which is relatively low. As the model assumes a linear relation between the predictor variables and the response, this may indicate that there is a linear relation between the predictor variables and the response. To test for this is beyond the scope of this report, but a interesting observation.

This fit score may be be challenged by the range of the residuals, as given under Deviance Residuals, which here is relatively large [-2.6613, 2.3800]. I think this result alone can put the goodness of fit of the model in question, but seen in light of other observations discussed later, the model has a good fit to the data.

KNN

```
fit.knn
```

```
## k-Nearest Neighbors
##
## 572 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 516, 514, 516, 515, 515, 514, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.6626696  0.2866216
##    2  0.6434891  0.2430196
##    3  0.6923753  0.3457929
##    4  0.6729204  0.2974422
##    5  0.6940995  0.3473742
##    6  0.6937322  0.3492299
##    7  0.7094017  0.3785913
##    8  0.6920999  0.3431129
##    9  0.6887132  0.3368029
##   10  0.6801540  0.3169071
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

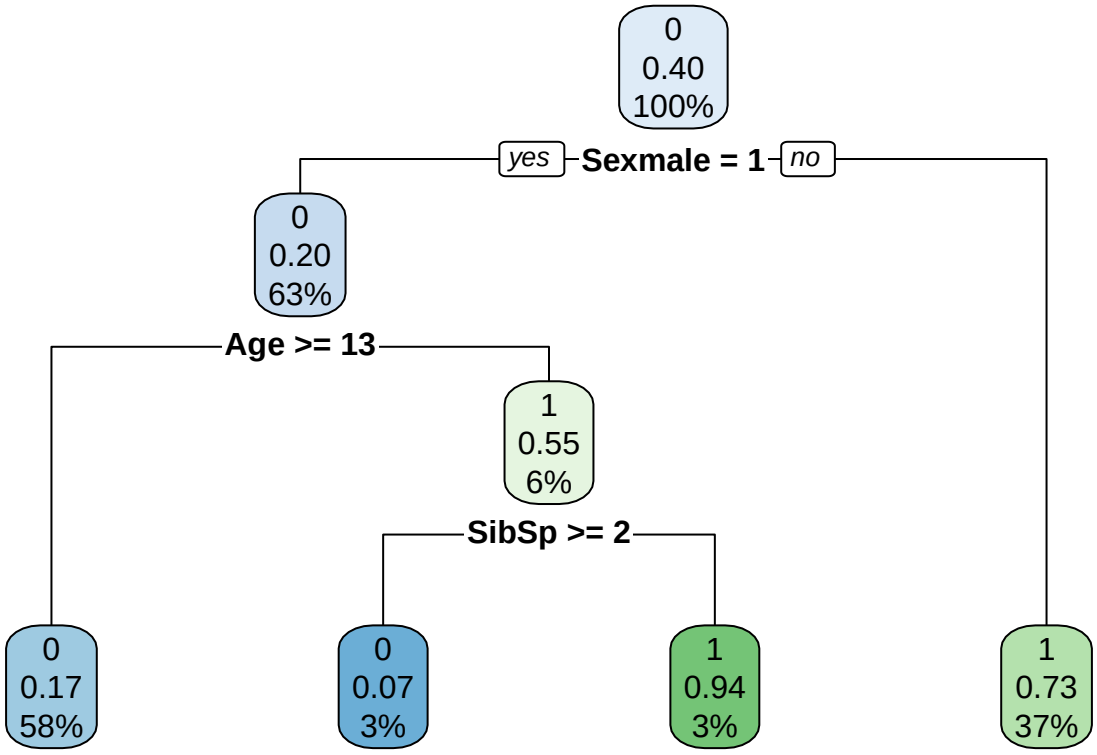
There is one hyperparameter in the KNN model, which is the value of k. As discussed under the model section, the determination of this parameter is essential for the performance of the model regarding under- and over fitting. The determination of the value of k is done by iterating over the integers in the closed range [1, 10]. The k value is chosen by cross validation (10-fold), using a grid search to find the k value in the specified interval, which maximizes the models accuracy. Here, that value is k = 7. With this k value, the model has an accuracy of 0.7094017.

A question one can ask is that if the optimal value for k lies in this interval? This is not for sure, but the larger k-s one choose, may also increase the computational cost of the cross-validation procedure, which is not desirable.

Based on the complexity of finding k, the algorithm is more computationally expensive. Also, KNN stores all training examples and classifies based on this stored data, it may not be the best model considering computer memory, the model may be avoided for larger data sets (not that this is an issue with modern machines, and in our case).

Decision Tree

```
rpart.plot(fit.tree$finalModel)
```




```
fit.tree
```

```
## CART
##
## 572 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 516, 514, 516, 515, 515, 514, ...
```

```
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03303965  0.7709630  0.5011931
##   0.03524229  0.7656674  0.4893304
##   0.43171806  0.6676508  0.2197559
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03303965.
```

Above, a visual representation of the decision tree is given. Even though this model is one of our more complex models, it is really straight forward to interpret. This can be an advantage in many cases.

There is one hyperparameter in the Decision Tree model, which is the value of C_p. This stands for the complexity parameter of the tree, i.e, the number of splits and the depth of the tree, and its ability to generalize to new data. Smaller values of C_p results in simpler trees, while larger values have the opposite effect. In our case, the tree is very simple, and the value of $C_p = 0.03303965$ is small. Hence, this may indicate that the model is not over fitted to the training data, and will probably give high accuracy on new data. The value of C_p is determined by using cross validation (10-fold), and the model with highest accuracy is chosen.

Random Forest

```
fit.forest
```

```
## Random Forest
##
## 572 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 516, 514, 516, 515, 515, 514, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8092278  0.5899248
##   4     0.7970065  0.5693730
##   6     0.7901715  0.5576928
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

There is one hyperparameter in the Random Forest model, which is the value of mtry. This value controls the number of predictors which are considered at each split. The predictors are randomly chosen from the pool of predictors in the model (here there are 6 predictors), and determines the best spilt based on these. In general, a larger value of mtry leads to a more improved accuracy, but has a higher risk of over fitting. On the contrary, a smaller value may increase the risk of under fitting, but reduces the risk of over fitting.

In our case, the value chosen for mtry is 2. The value of mtry is determined by using cross validation (10-fold), and the model with highest accuracy is chosen. This value is small, which eliminates the risk of over fitting, but the model may be under fitted. This does not seem to be the case, as this is our most accurate model.

ROC and AUC

When making a model that is best at prediction/classification, we want to maximize the AUC value and get a ROC curve that is as close to the top-left corner as possible. This indicates that the model is better at distinguishing between the positive (1: alive) and negative (0: death) class. If we look at this metric of measure alone, we see that the models who are most outstanding are Logistic Regression and Random Forests. Logistic Regression has a higher AUC than the Random Forest, but for practical purposes, they are equivalent in our case.

It is not surprising that the Random Forest outperforms the decision tree, as the random forest is built upon decision trees and address some of the limitations of decisions tree. The decision tree is not far off either, only 0.08 in AUC value behind the random forest. The thing that surprises me the most, is that the Logistic Regression model outperforms all the "more complex" models, besides Random Forest. Even though this may be surprising, this may reveal some information about our data that we did not foresee. As discussed above, the model assumes a linear relationship between the predictor variables and the outcomes. That Logistic Regression does so well, may indicate that there is a linear relation between the predictors and the outcome.This is interesting insight, which can be tested further.

The KNN performs the worst, with a AUC score significantly less than the other models. As discussed, it is relatively hard to determine a good k value. We did a search in the interval [1,10], and maybe, this interval does not hold the optimal k.

Just based on the ROC and AUC, I would say that Logistic Regression and Random Forest, for all practical reasons, are of equal accuracy when it comes to prediction.

Sensitivity, Specificity and confusion matrices

When making a model that is best at prediction/classification, we are looking for a balance between sensitivity and sensitivity, and that the values are high.

Sensitivity measures the proportion of positive cases that are correctly identified by the model. On the other hand, Specificity measures the proportion of negative cases that are correctly identified by the model. This can be used to say something about the TYPE I and TYPE II errors of the methods, but this is not of interest in our case.

Across the board, the model with the highest value for Sensitivity and Specificity is the Random Forest. One would assume, from previous discussion on ROC and AUC, that Logistic Regression has a Sensitivity and Specificity value that is close to that of the Random Forest. This is correct for Specificity, but when it comes to Sensitivity, the value is far off. In fact, the Specificity for Random Forest, Decision Tree and Logistic Regression seems to be approximately the same. To look further into which model has the best Specificity, we therefore turn to the confusion matrices.

The logistic Regression identifies 73 correct and 6 cases wrong. Decision Tree identifies 71 correct and 8 cases wrong. Random Forest identifies 77 correct and 2 cases wrong. The differences are small, but they are there. Random Forest is clearly the best model, when considering this. Another curios discovery is that all the models are better at identifying if a person died than if they survived. This is evident in the confusion matrices, that the models make fewer mistakes when identifying a dead person. Another curious discovery is that all models (excluding KNN) have similar (identical) results in the confusion matrices when it comes to predicting people that survived. This may be a coincident, but I cannot seem to find a good explanation for this result.

KNN is again the model which performs the worst, but this method has the greatest balance between the Specificity and Sensitivity value (the difference is 0.05). A model which has the same balance, but performs better is the Decision Tree.

All in all, the best performing model based on Sensitivity, Specificity and confusion matrices is the Random Forest.

F1-scores

F1 value is a measure of accuracy, which is used to consider both precision and recall of a model. A large value means that the model is able to correctly identify a large number of positive cases (high recall) and classify these as positive cases (high precision).

Looking at the F1, we see that the Random Forest model performs far better than any other model with a F1 value of 0.82. This is followed by Logistic Regression at 0.79, and Decision Tree at 0.77. The worst performing model is again KNN, with a value of 0.57. The value for KNN indicates that the model has relative low performance.

The outstanding models are again Logistic Regression and Random Forests.

Summary

The objective of this analysis was to identify the best-performing model among four machine learning techniques:

- Logistic Regression
- K Nearest Neighbor (KNN)
- Decision Tree
- Random Forest

The findings reveal that both Logistic Regression and Random Forest models are highly accurate, with the Random Forest approach slightly outperforming the former. The selection of the model ultimately depends on the interpretability requirement. For a more interpretable model, Logistic Regression should be preferred. However, since the objective of this analysis was to identify the most accurate model, Random Forest was chosen.

Hence, Random Forest is the most accurate model among the models studied, for this data set.

Some final stats for our Random Forest model is presented below:

```
RandomForestStats <- data.frame(
  Model = c("F1-score", "AUC", "Sensitivity",   "Specificity"),
  Results = c(F1FOREST, auc(roc.forest), sensitivityFOREST, specificityFOREST)
)

kable(RandomForestStats, caption = "Stats for Random Forest Model")
```

Table 3: Stats for Random Forest Model

Model	Results
F1-score	0.8181818
AUC	0.8824593
Sensitivity	0.9574468
Specificity	0.8105263

During the data analysis, it was observed that there might be a linear relationship between the predictor variables and the outcome. This observation was supported by the performance of the Logistic Regression model, which assumes a linear relationship between the predictors and the outcome.

From the descriptive analysis the result seems to be that the chance of survival was greater if you had a expensive cabin or was female or traveled with a small or no family at all.