Predicting Titanic Survival using Five Algorithms

Thilaksha Silva

02 December 2017

Contents

Introduction

I am stepping into the Machine Learning world with my first Kaggle competition! This real world classification problem helped me to greatly practice some predictive analytics techniques I have studied.

My script consists of five sections:

- Handling Missing Data
- Feature Engineering
- Exploratory Data Analysis

- Prediction
- Discussion

I imputed sensible values for features **Fare**, **Embarked** and **Age**.

I also incorporated feature enginneering to create two new features, passenger **Title** and **FamilySize**, extracted from existing features.

On exploratory data analysis, I genuinely spent much time to analyse data with visual methods to summarize the main characteristics.

With this titanic dataset, I explore five classification algorithms: Logistic Regression, Support Vector Machines (both linear and non-linear), Decision Tree, Random Forest, and Naive Bayes.

I will guide you through my journey of exploring titanic survival. I am eager to learn more and develop skills on machine learning. Any feedback is very welcome!

The Data

```
# Load all the packages required for the analysis
library(dplyr) # Data Manipulation
library(Amelia) # Missing Data: Missings Map
library(ggplot2) # Visualization
library(scales) # Visualization
library(caTools) # Prediction: Splitting Data
library(car) # Prediction: Checking Multicollinearity
library(ROCR) # Prediction: ROC Curve
library(e1071) # Prediction: SVM, Naive Bayes, Parameter Tuning
library(rpart) # Prediction: Decision Tree
library(rpart.plot) # Prediction: Decision Tree
library(randomForest) # Prediction: Random Forest
library(caret) # Prediction: k-Fold Cross Validation
```

```
titanic_train = read.csv('train.csv')
titanic_test = read.csv('test.csv')

# Combining data
titanic <- bind_rows(titanic_train, titanic_test)

# Checking the structure of the data
str(titanic)
```

```
## 'data.frame':    1309 obs. of  12 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : chr  "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : chr  "A/5 21171" "PC 17599" "STON/O2. 3101282" "113803" ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : chr  "" "C85" "" "C123" ...
##  $ Embarked   : chr  "S" "C" "S" "S" ...
```

Handling Missing Data

Checking Missing Data

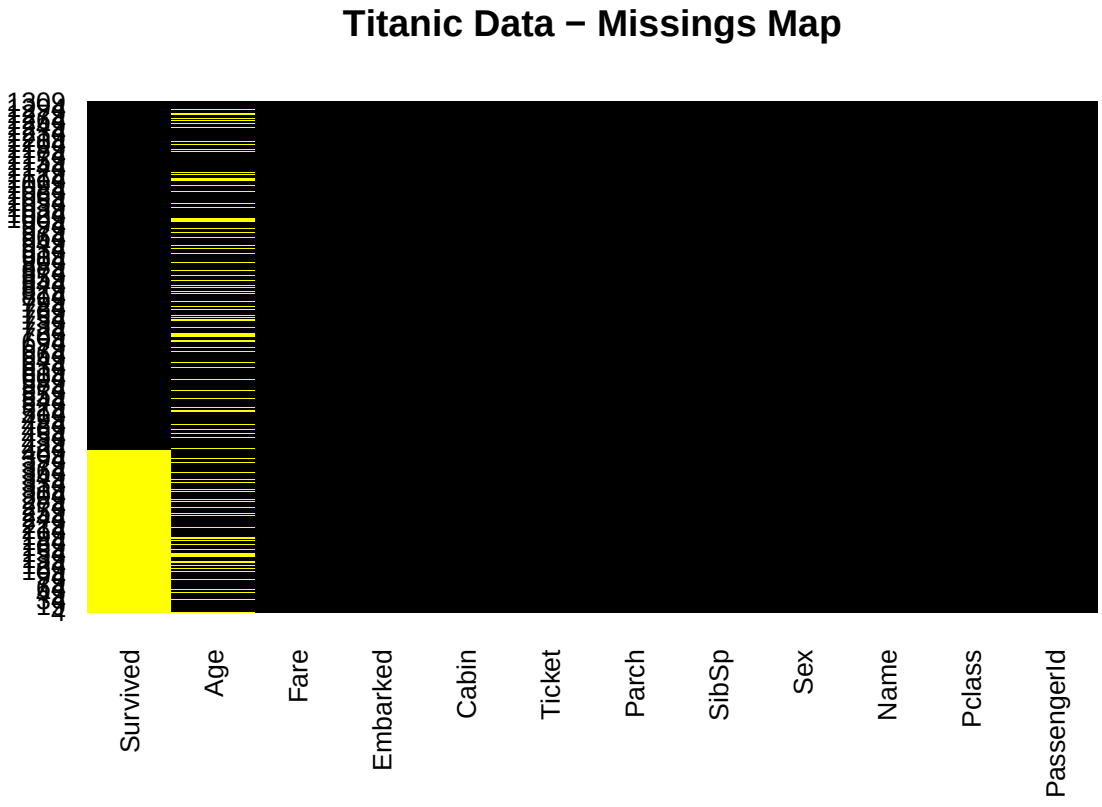
```
# Checking missing values (missing values or empty values)
colSums(is.na(titanic)|titanic=='')
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0         418           0           0           0         263
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##           0           0           0           1        1014           2
```

Cabin has the most number of missing values, 1014 values. Age has 263 missing values while Embarked and Fare have two and one missing values, respectively.

`missmap` allows us to explore how much missing data we have.

```
missmap(titanic, main="Titanic Data - Missings Map",
        col=c("yellow", "black"), legend=FALSE)
```



`missmap` function considers "NA" values as missing values but it does not consider empty values as missing values. The missings map plot shows some of the age data is missing (because having a quick glance at the dataset we realise the missing age data are stored as NA). However, the plot does not show cabin has missing values (because missing cabin data are stored as empty values not NA values).

The next step is to fill the missing data rows instead of just dropping them.

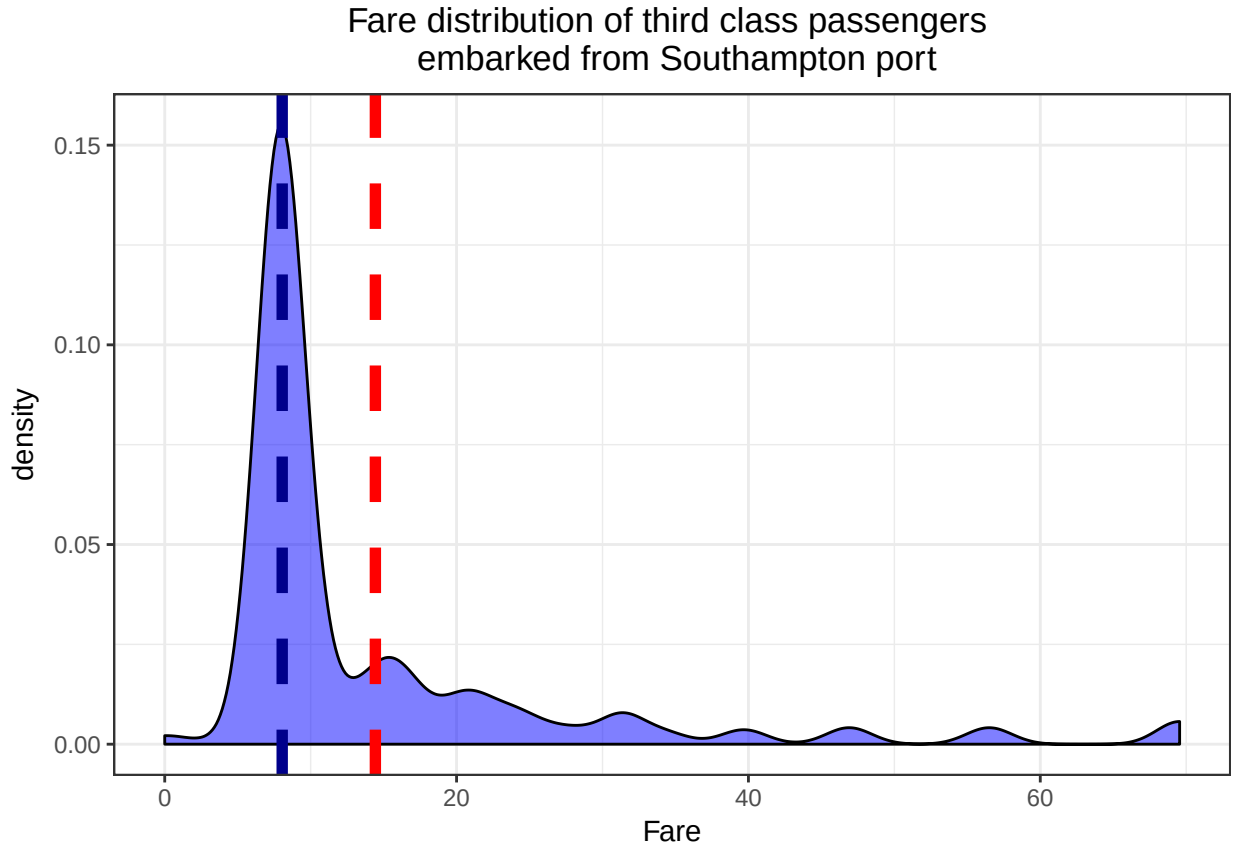
Missing Fare Data Imputation

```
# Extract the row which contains the missing Fare
filter(titanic, is.na(Fare)==TRUE|Fare=='')
```

```
##   PassengerId Survived Pclass               Name  Sex  Age SibSp Parch
## 1        1044       NA      3 Storey, Mr. Thomas male 60.5     0     0
##   Ticket Fare Cabin Embarked
## 1   3701   NA              S
```

This male was from the third class and had embarked from Southampton port. Let's look at the distribution of third class passengers embarked from Southampton port.

```
ggplot(filter(titanic, Pclass==3 & Embarked=="S"), aes(Fare)) +
  geom_density(fill="blue", alpha=0.5) +
  geom_vline(aes(xintercept=median(Fare, na.rm=T)), colour='darkblue', linetype='dashed', size=2) +
  geom_vline(aes(xintercept=mean(Fare, na.rm=T)), colour='red', linetype='dashed', size=2) +
  ggtitle("Fare distribution of third class passengers \n embarked from Southampton port") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
```



The mean and median fares are not close. The proportion of passengers with fares around median is very high. On the oher hand, it is not that high around mean. So, I believe it is not a good idea to impute the missing fare by the mean of all fares. I would rather impute the missing fare by the median fare of third class passengers embarked from Southampton port.

```
# Impute the missing Fare value by the median fare of third class passengers embarked from Southampton
titanic$Fare[is.na(titanic$Fare)==TRUE] = median(filter(titanic, Pclass==3 & Embarked=="S")$Fare, na.rm=
```

```
# Checking missing values
colSums(is.na(titanic)|titanic=='')
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0         418           0           0           0         263
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##           0           0           0           0        1014           2
```

The missing fare has been replaced.

Missing Embarked Data Imputation

We have noticed that there are two missing values for the Embarked feature.

```
# Extract the rows which contain the missing Embarked values
filter(titanic, is.na(Embarked)==TRUE|Embarked=='')
```

```
##   PassengerId Survived Pclass                                        Name
## 1          62        1      1                         Icard, Miss. Amelie
## 2         830        1      1 Stone, Mrs. George Nelson (Martha Evelyn)
##      Sex Age SibSp Parch Ticket Fare Cabin Embarked
## 1 female  38     0     0 113572   80   B28
## 2 female  62     0     0 113572   80   B28
```

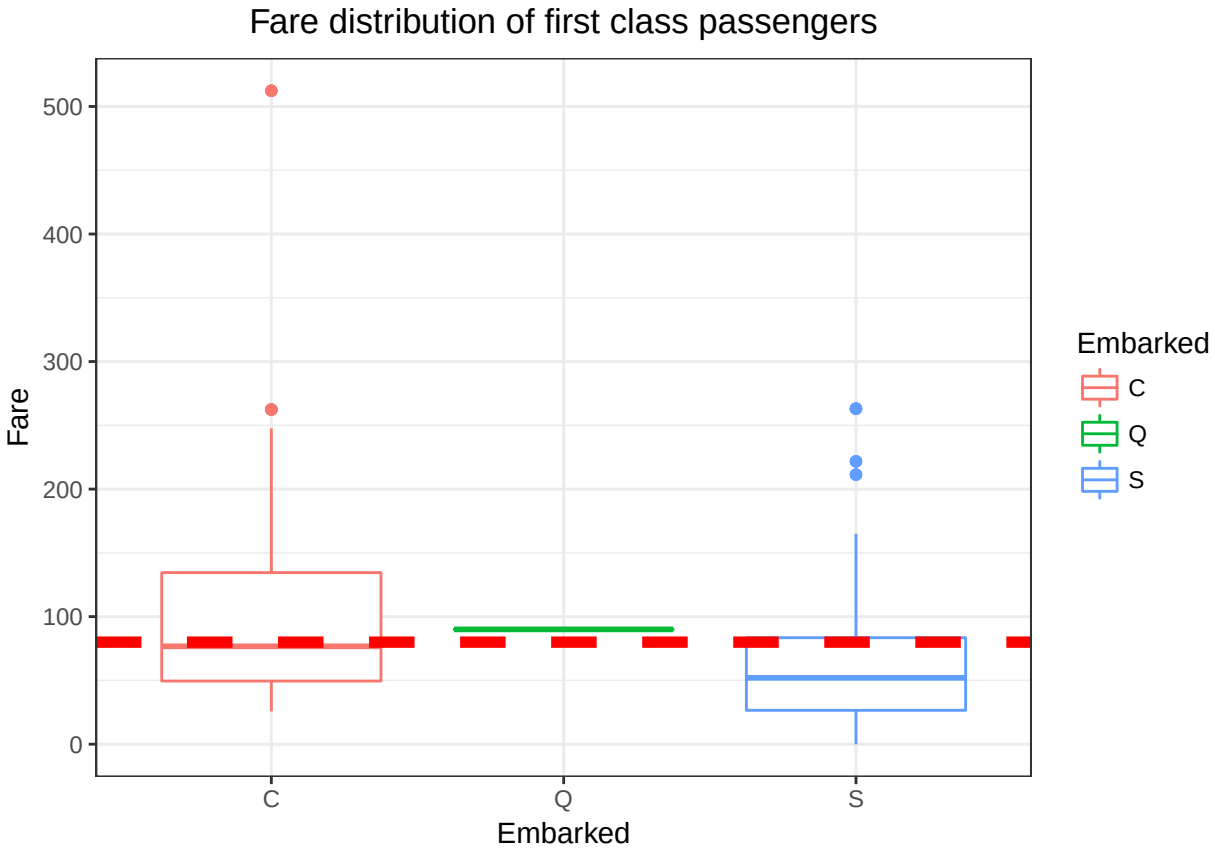
Both were females from the first class with $80 fare and had stayed at the same cabin B28. There is a high chance that both embarked from the same port. Let's look at the frequency of ports of embarkation of first class passengers.

```
# Frequency of ports of embarkation of first class passengers
table(filter(titanic, Pclass==1)$Embarked)
```

```
##
##       C   Q   S
##   2 141   3 177
```

The Southampton port is the most frequent port of embarkation with 177 ports and it is followed by the Cherbourg port with 141. Wait! Yet, we cannot decide to impute two missing values by the most frequent port of embarkation which is the Southampton port.

```
ggplot(filter(titanic, is.na(Embarked)==FALSE & Embarked!='' & Pclass==1),
       aes(Embarked, Fare)) +
  geom_boxplot(aes(colour = Embarked)) +
  geom_hline(aes(yintercept=80), colour='red', linetype='dashed', size=2) +
  ggtitle("Fare distribution of first class passengers") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
```



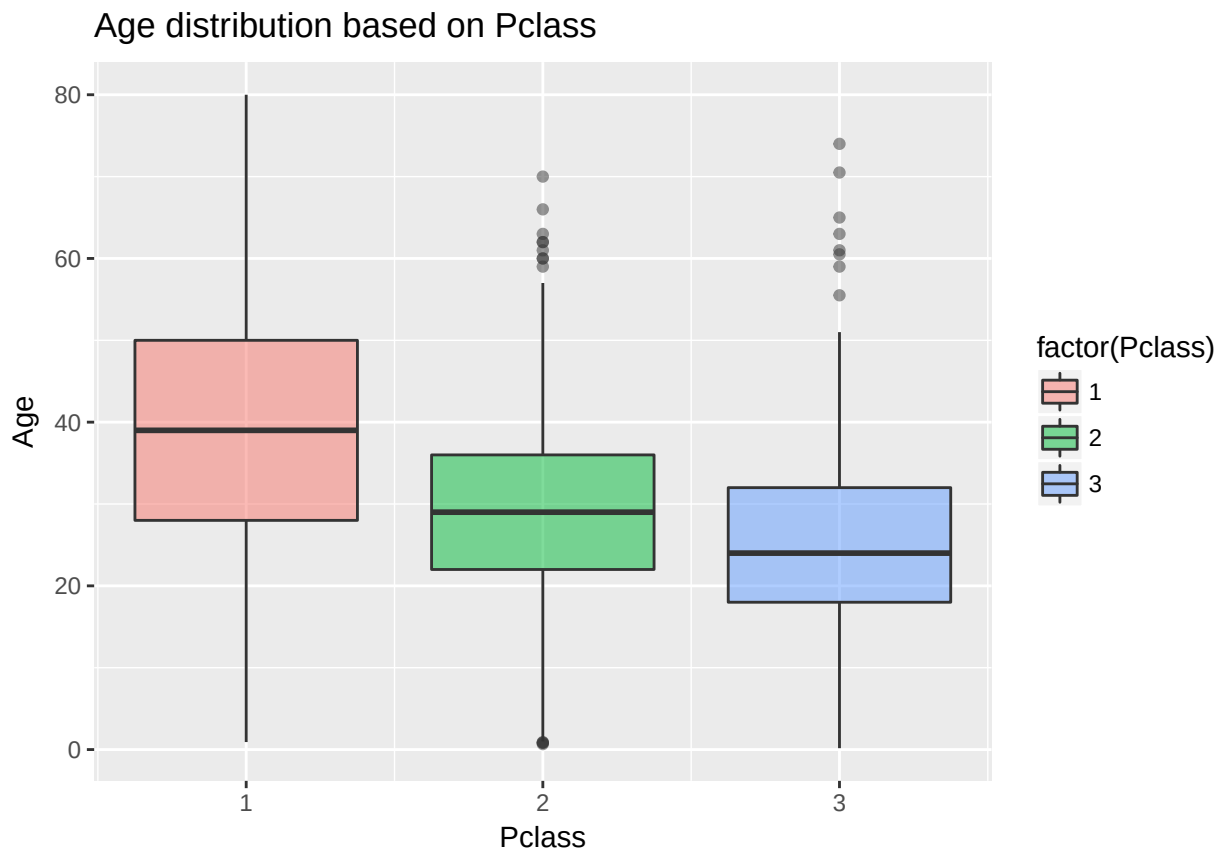
The box plot depicts the median fare for Cherbourg port passengers and $80 fare paid by two embarkement-deficient passengers almost concide. Thus, I am going to impute the missing Embarked values by the Cherbourg port.

```
# Impute the missing Embarked values by the Cherbourg port
titanic$Embarked[titanic$Embarked==""] = "C"

# Checking missing values
colSums(is.na(titanic)|titanic=='')
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0         418           0           0           0         263
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##           0           0           0           0        1014           0
```

Missing Age Data Imputation

```
ggplot(titanic,aes(Pclass,Age)) +
  geom_boxplot(aes(fill=factor(Pclass)),alpha=0.5) +
  ggtitle("Age distribution based on Pclass")
```



It can be clearly seen the median age among classes is not similar (virtually certain, average age among classes is not similar as well). Infact, the passengers in the higher classes tend to be older. Rather than just imputing missing age values by the overall average for age, I will use average age values of each class to impute missing age values based on Pclass.

```
# Imputation of Age based on Pclass
impute.age <- function(age,class){
  vector <- age
  for (i in 1:length(age)){
    if (is.na(age[i])){
      if (class[i] == 1){
        vector[i] <- round(mean(filter(titanic,Pclass==1)$Age, na.rm=TRUE),0)
      }else if (class[i] == 2){
        vector[i] <- round(mean(filter(titanic,Pclass==2)$Age, na.rm=TRUE),0)
      }else{
        vector[i] <- round(mean(filter(titanic,Pclass==3)$Age, na.rm=TRUE),0)
      }
    }else{
      vector[i]<-age[i]
    }
  }
  return(vector)
}
imputed.age <- impute.age(titanic$Age,titanic$Pclass)
titanic$Age <- imputed.age
```

Let's check if the above imputation method worked.

```r
# Checking missing values
colSums(is.na(titanic)|titanic=='')
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0         418           0           0           0           0
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##           0           0           0           0        1014           0
```

It worked. Now we are left with only Cabin missing values. However, due to the high number of missing values of Cabin feature, I keep the Cabin feature as it is and stop here.

Feature Engineering

Passenger Title

Since the title of the passengers is contained within the passenger name feature, let's do some feature engineering to create a new feature with passenger titles.

```r
head(titanic$Name)
```

```
## [1] "Braund, Mr. Owen Harris"
## [2] "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
## [3] "Heikkinen, Miss. Laina"
## [4] "Futrelle, Mrs. Jacques Heath (Lily May Peel)"
## [5] "Allen, Mr. William Henry"
## [6] "Moran, Mr. James"
```

```r
# Grab passenger title from passenger name
titanic$Title <- gsub("^.*, (.*?)\\..*$", "\\1", titanic$Name)

# Frequency of each title by sex
table(titanic$Sex, titanic$Title)
```

```
##
##          Capt Col Don Dona  Dr Jonkheer Lady Major Master Miss Mlle Mme
##   female    0   0   0    1   1        0    1     0      0  260    2   1
##   male      1   4   1    0   7        1    0     2     61    0    0   0
##
##           Mr Mrs  Ms Rev Sir the Countess
##   female   0 197   2   0   0            1
##   male   757   0   0   8   1            0
```

```r
# First, I reassign few categories
titanic$Title[titanic$Title == 'Mlle' | titanic$Title == 'Ms'] <- 'Miss'
titanic$Title[titanic$Title == 'Mme']  <- 'Mrs'

# Then, I create a new category with low frequency of titles
Other <- c('Dona', 'Dr', 'Lady', 'the Countess','Capt', 'Col', 'Don', 'Jonkheer', 'Major', 'Rev', 'Sir')
titanic$Title[titanic$Title %in% Other]  <- 'Other'

# Let's see if it worked
table(titanic$Sex, titanic$Title)
```

```
##
##          Master Miss  Mr Mrs Other
```

```
##   female      0  264   0 198     4
##   male       61    0 757   0    25
```

The title is down to five categories. We will do exploratory analysis based on title in the next section.

Family Size

I have also noticed that a new feature on family size can be created using some existing features such as **SigSp** and **Parch**.

```
FamilySize <- titanic$SibSp + titanic$Parch + 1

table(FamilySize)
```

```
## FamilySize
##   1   2   3   4   5   6   7   8  11
## 790 235 159  43  22  25  16   8  11
```

There are nine family sizes: 1 to 8 and 11. As this is too many categories, let's collapse some categories as follows.

```
# Create a family size feature with three categories
titanic$FamilySize <- sapply(1:nrow(titanic), function(x)
                          ifelse(FamilySize[x]==1, "Single",
                          ifelse(FamilySize[x]>4, "Large", "Small")))

table(titanic$FamilySize)
```

```
##
##  Large Single  Small
##     82    790    437
```

In the next section, we will do some exploratory anaysis based on family size.

Exploratory Data Analysis

Encoding the categorical features as factors

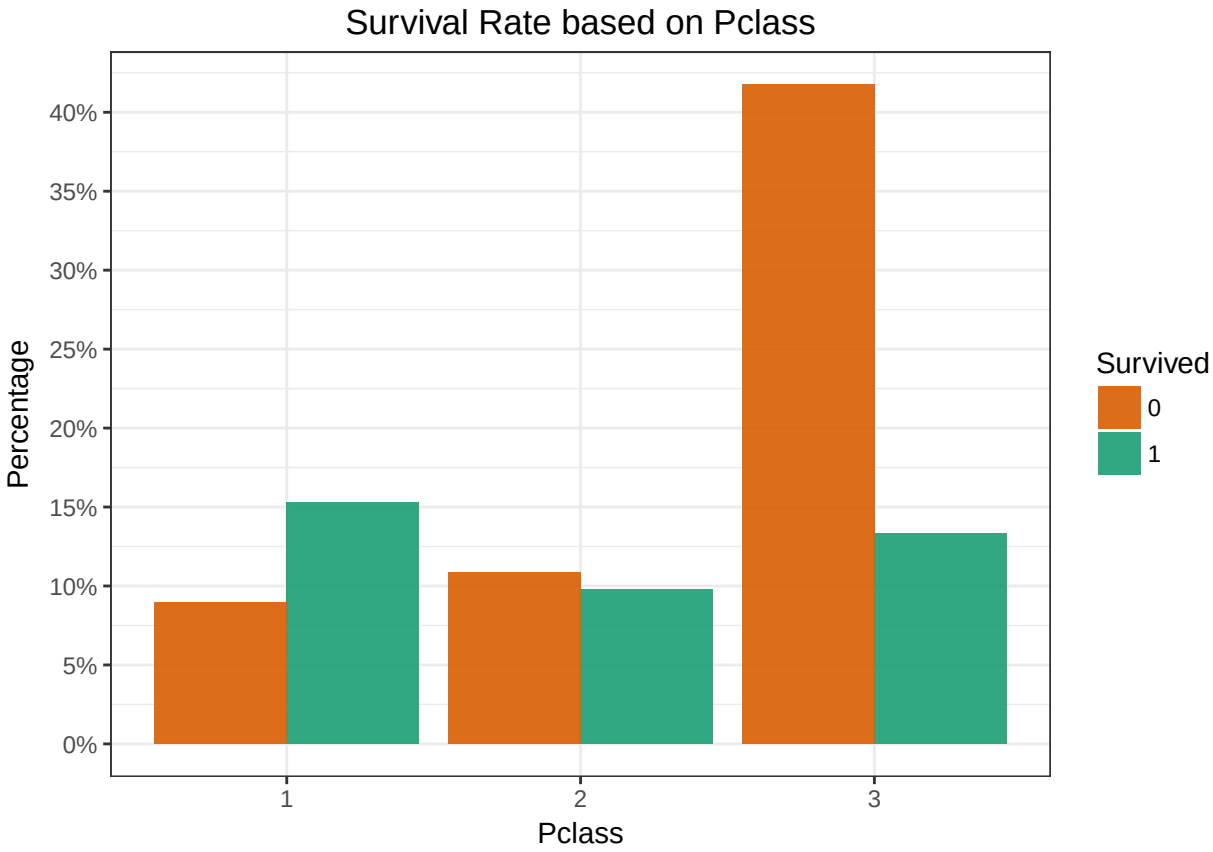
```
titanic$Survived = factor(titanic$Survived)
titanic$Pclass = factor(titanic$Pclass)
titanic$Sex = factor(titanic$Sex)
titanic$Embarked = factor(titanic$Embarked)
titanic$Title = factor(titanic$Title)
titanic$FamilySize = factor(titanic$FamilySize, levels=c("Single","Small","Large"))

#Checking the structure of the data
str(titanic)
```

```
## 'data.frame':    1309 obs. of  14 variables:
##  $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived   : Factor w/ 2 levels "0","1": 1 2 2 2 1 1 1 1 2 2 ...
##  $ Pclass     : Factor w/ 3 levels "1","2","3": 3 1 3 1 3 3 1 3 3 2 ...
##  $ Name       : chr  "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bradley (Florence Briggs Thayer)"
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
```

```
##  $ Age        : num  22 38 26 35 35 25 54 2 27 14 ...
##  $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket     : chr  "A/5 21171" "PC 17599" "STON/O2. 3101282" "113803" ...
##  $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin      : chr  "" "C85" "" "C123" ...
##  $ Embarked   : Factor w/ 3 levels "C","Q","S": 3 1 3 3 3 2 3 3 3 1 ...
##  $ Title      : Factor w/ 5 levels "Master","Miss",..: 3 4 2 4 3 3 3 1 4 4 ...
##  $ FamilySize : Factor w/ 3 levels "Single","Small",..: 2 2 1 2 1 1 1 3 2 2 ...
```

Exploratory Data Analysis on Pclass, Sex and Age

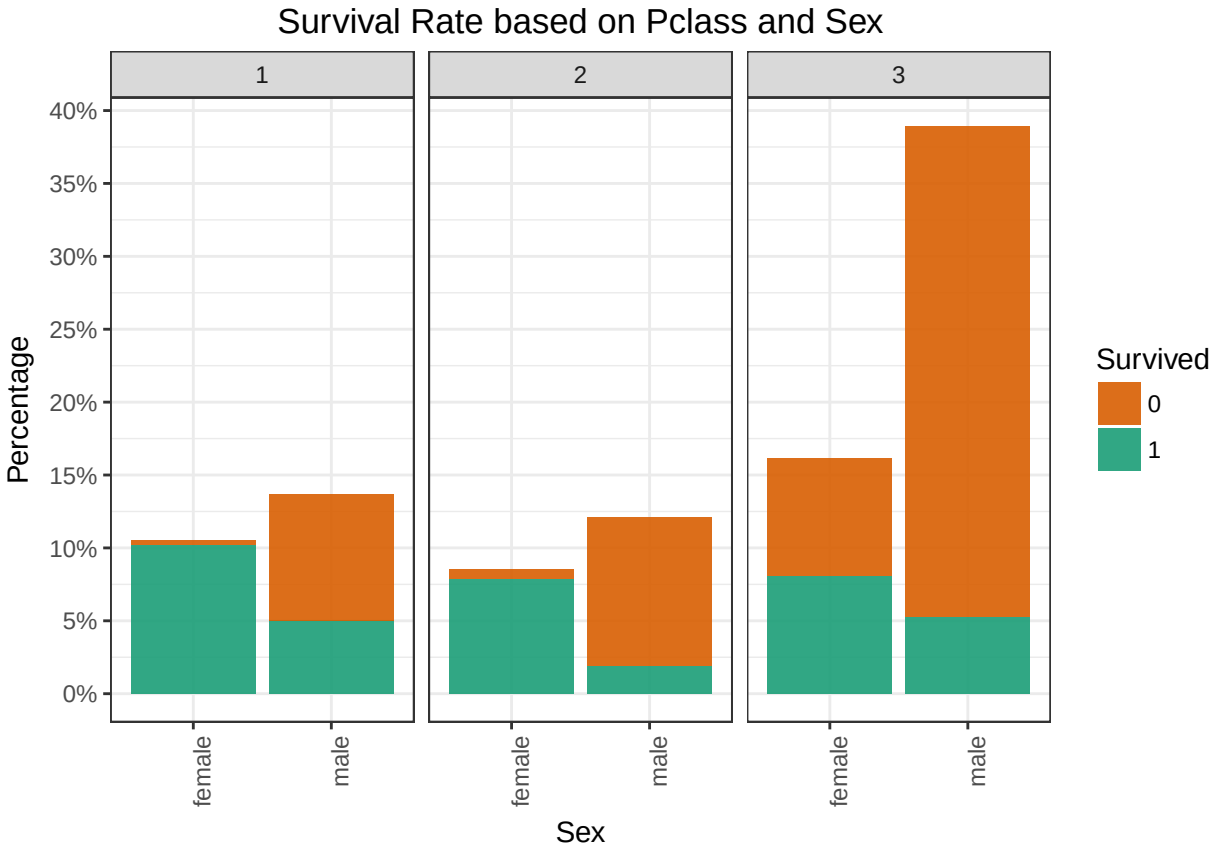
```
ggplot(filter(titanic, is.na(Survived)==FALSE), aes(Pclass, fill=Survived)) +
  geom_bar(aes(y = (..count..)/sum(..count..)), alpha=0.9, position="dodge") +
  scale_fill_brewer(palette = "Dark2", direction = -1) +
  scale_y_continuous(labels=percent, breaks=seq(0,0.6,0.05)) +
  ylab("Percentage") +
  ggtitle("Survival Rate based on Pclass") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
```



Of course as expected, the wealthier passengers in the first class had a higher survival rate, roughly 15%, than the second class and third class passengers.

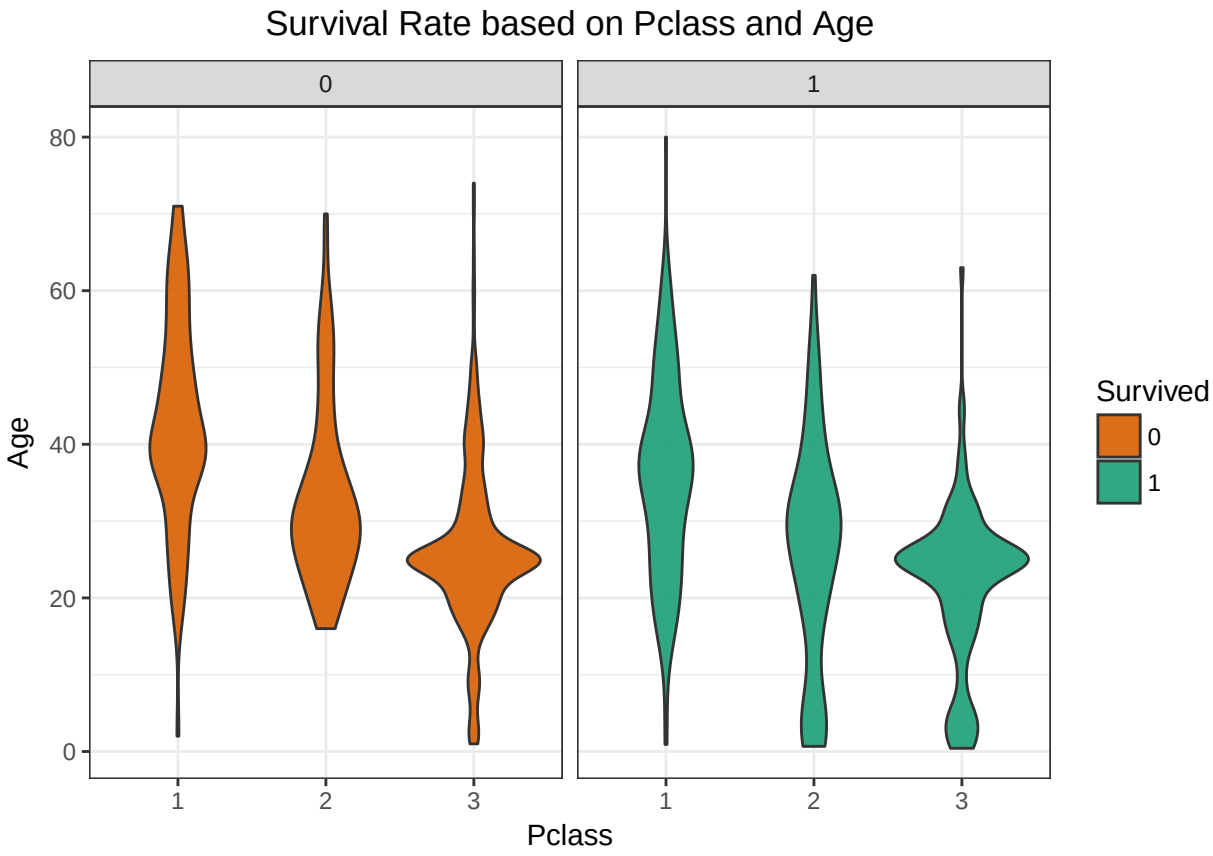
Let's continue on by visualising data of some of the features.

```
ggplot(filter(titanic, is.na(Survived)==FALSE), aes(Sex, fill=Survived)) +
  geom_bar(aes(y = (..count..)/sum(..count..)), alpha=0.9) +
  facet_wrap(~Pclass) +
  scale_fill_brewer(palette = "Dark2", direction = -1) +
  scale_y_continuous(labels=percent, breaks=seq(0,0.4,0.05)) +
  ylab("Percentage") +
  ggtitle("Survival Rate based on Pclass and Sex") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



It can be seen that females had a higher survival rate than males in each class. This makes sense due to women and children first policy.

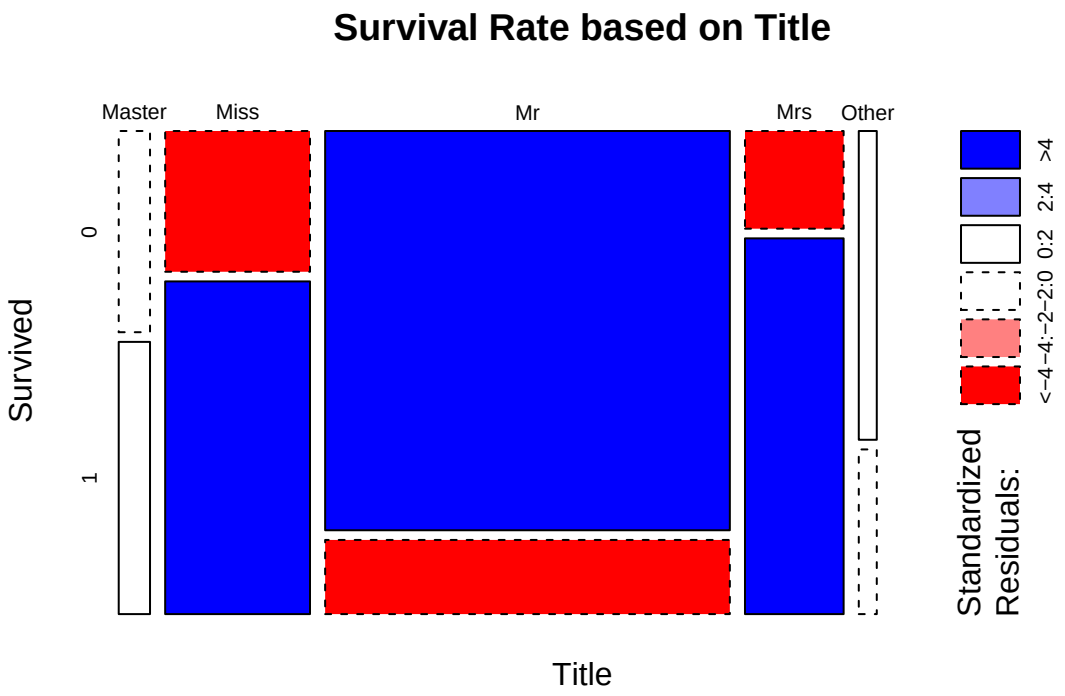
```
ggplot(filter(titanic, is.na(Survived)==FALSE), aes(Pclass, Age)) +
  geom_violin(aes(fill=Survived), alpha=0.9) +
  facet_wrap(~Survived) +
  scale_fill_brewer(palette = "Dark2", direction = -1) +
  ggtitle("Survival Rate based on Pclass and Age") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
```



Overall, the passengers in the higher classes tend to be older disregard to whether they survived or not.

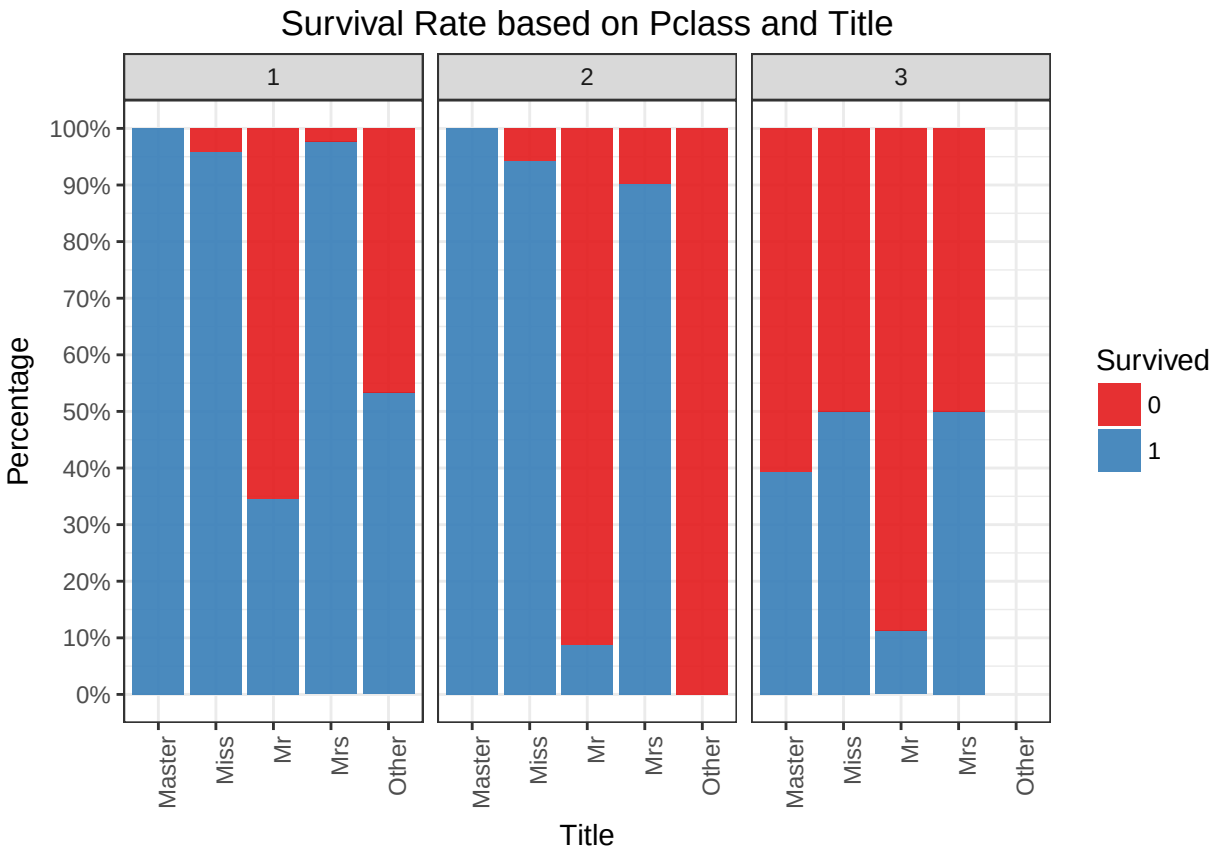
Exploratory Data Analysis on Title and FamilySize

```
mosaicplot(~ Title + Survived, data=titanic, main='Survival Rate based on Title', shade=TRUE)
```



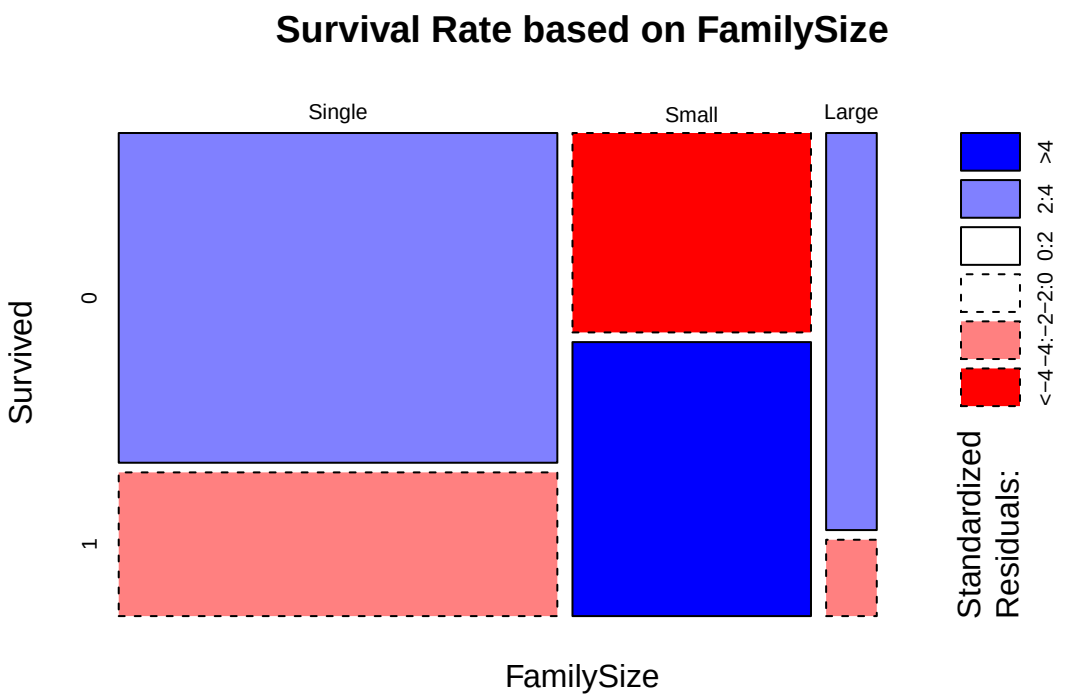
Generally, male "Mr" passengers had the poorest survival rate.

```
ggplot(filter(titanic, is.na(Survived)==FALSE), aes(Title)) +
  geom_bar(aes(fill=Survived), alpha=0.9, position="fill") +
  facet_wrap(~Pclass) +
  scale_fill_brewer(palette="Set1") +
  scale_y_continuous(labels=percent, breaks=seq(0,1,0.1)) +
  ylab("Percentage") +
  ggtitle("Survival Rate based on Pclass and Title") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



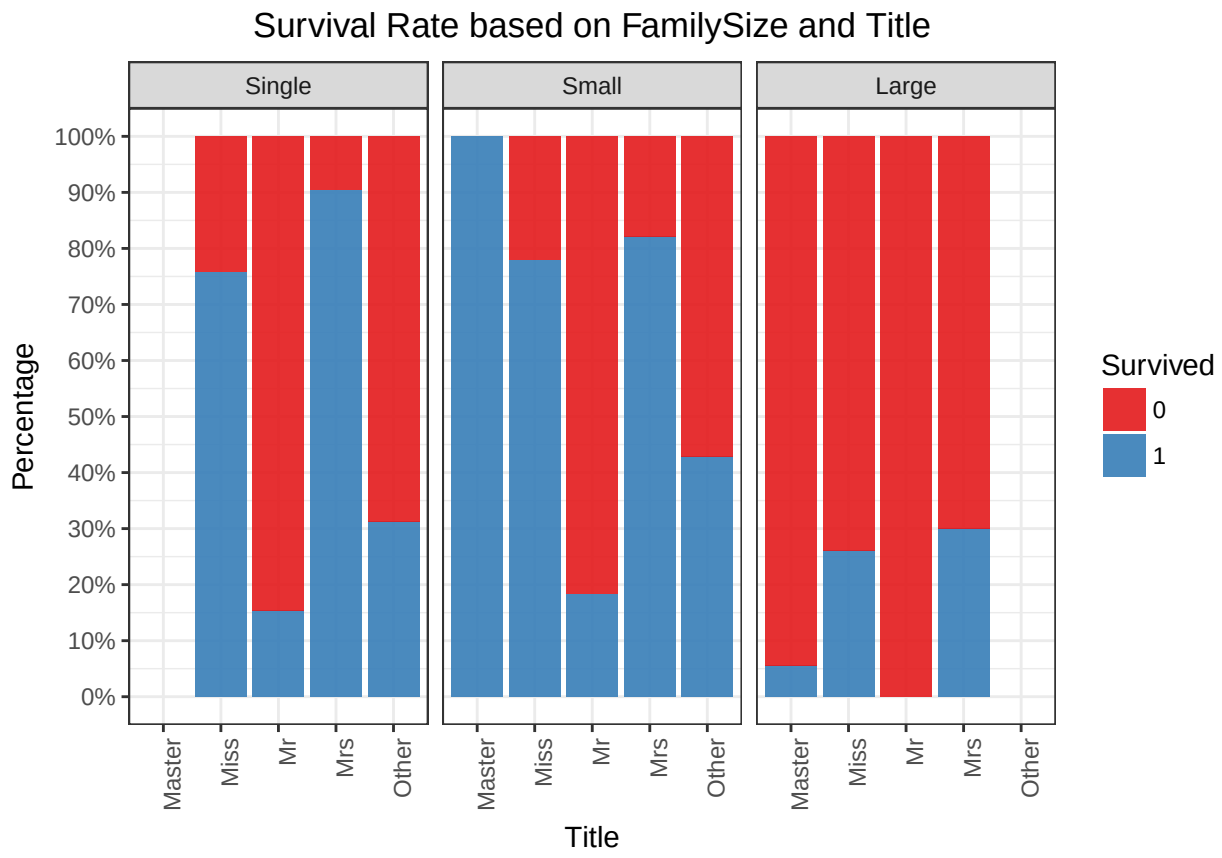
The same information can be depicted from the above graph - the male "Mr" passengers had the lowest survival rate amongst all the classes.

```
mosaicplot(~ FamilySize + Survived, data=titanic, main='Survival Rate based on FamilySize', shade=TRUE)
```



Large families had the worst survival rate than singletons and small families.

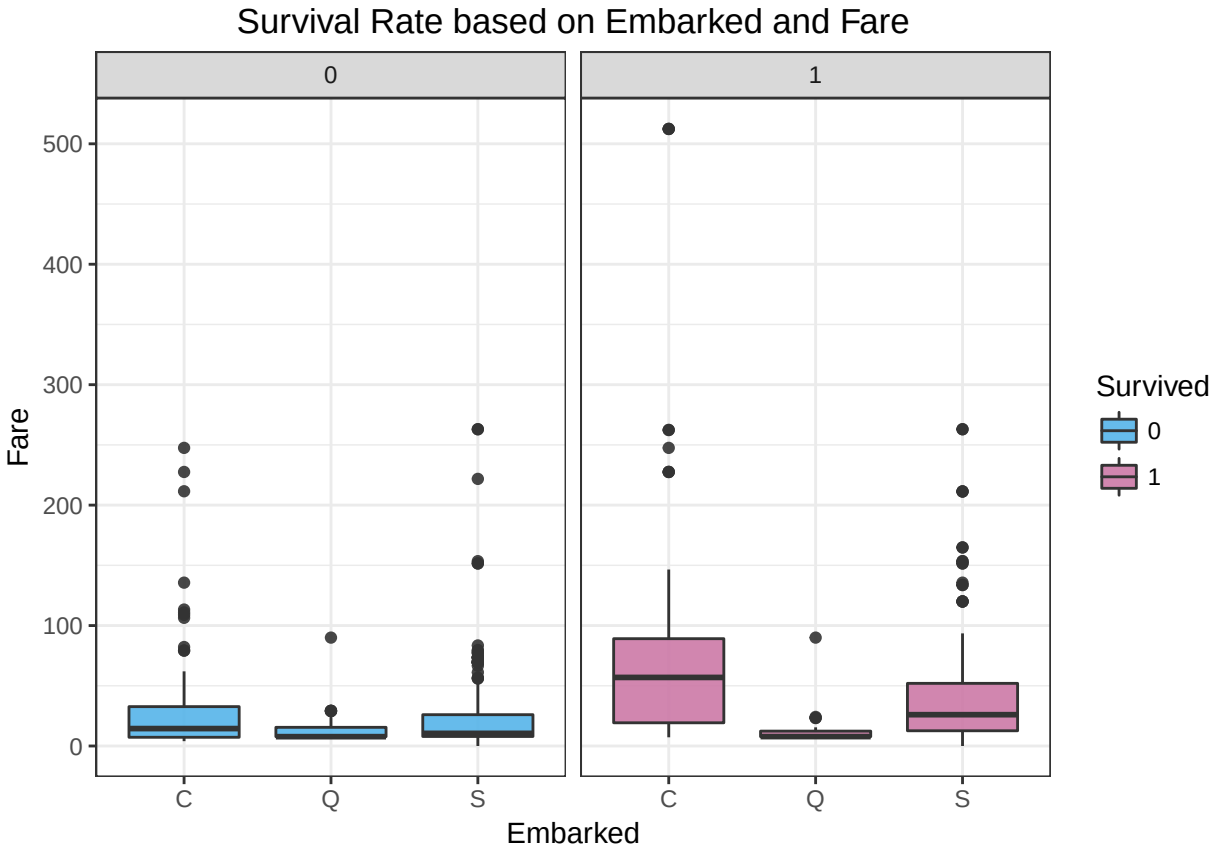
```
ggplot(filter(titanic, is.na(Survived)==FALSE), aes(Title)) +
  geom_bar(aes(fill=Survived), alpha=0.9, position="fill") +
  facet_wrap(~FamilySize) +
  scale_fill_brewer(palette="Set1") +
  scale_y_continuous(labels=percent, breaks=seq(0,1,0.1)) +
  ylab("Percentage") +
  ggtitle("Survival Rate based on FamilySize and Title") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The filled bar chart illustrates that we can preserve our rule: large families had the worst survival rate than singletons and small families. Infact, each member of the large families - Master, Miss, Mr and Mrs - suffered the lowest survival rate than their counterparts in other types of families.

Exploratory Data Analysis on Fare and Embarked

```
ggplot(filter(titanic, is.na(Survived)==FALSE), aes(Embarked, Fare)) +
  geom_boxplot(aes(fill=Survived), alpha=0.9) +
  facet_wrap(~Survived) +
  scale_fill_manual(values=c("#56B4E9", "#CC79A7")) +
  ggtitle("Survival Rate based on Embarked and Fare") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
```



Interestingly, there is a substantial variation of fares in the survived category, especially from Cherbourg and Southampton ports.

Visual analysis of data concludes:

- the wealthier passengers in the first class had a higher survival rate;
- females had a higher survival rate than males in each class;
- male "Mr" passengers had the lowest survival rate amongst all the classes; and
- large families had the worst survival rate than singletons and small families.

Woo-ah! After rectifying the missing values and exploring data visually, finally, we are ready to predict whether or not each passenger in the test set survived the sinking of the Titanic. I identify the following features may contribute to the prediction of the survival and include them in the classification algorithms: **Pclass**, **Sex**, **Age**, **SibSp**, **Parch**, **Fare**, **Embarked**, **Title** and **FamilySize**.

- I ignore the feature **Name** as I have created a new feature Title from it and I believe the title has more predictive power than just name.
- I ignore the feature **Ticket** as I believe it does not preserve any predictive power on survival.
- I ignore the feature **Cabin** since it has many missing values.

Prediction

Splitting the dataset into the Training set and Test set

After rectifying the missing values and encoding the categorical features as factors, now we are good to split the dataset into the training and test sets.

```
# Splitting the dataset into the Training set and Test set
train_original <- titanic[1:891, c("Survived","Pclass","Sex","Age","SibSp","Parch","Fare","Embarked","Ti
test_original <- titanic[892:1309, c("Pclass","Sex","Age","SibSp","Parch","Fare","Embarked","Title","Fa
```

Splitting the training set into the Training set and Validation set

Now I split the training set into the training set (80% of training data) and validation set (20% of training data) for the evaluation purposes of the fitted models.

```
# Splitting the Training set into the Training set and Validation set
set.seed(789)
split = sample.split(train_original$Survived, SplitRatio = 0.8)
train = subset(train_original, split == TRUE)
test = subset(train_original, split == FALSE)
```

Logistic Regression

The first maching learning classification algorithm I use in the analysis is the most popular and the simplest Logistic Regression. But wait! we cannot simply blindly fit the Logistic Regression. This algorithm comes with a price. We need to CHECK THE ASSUMPTIONS!!! Let's check the Logistic Regression assumptions: features should be independent from each other and residuals are not autocorrelated.

```
# Show the correlation of numeric features
cor(train[,unlist(lapply(train,is.numeric))])
```

```
##              Age      SibSp      Parch      Fare
## Age    1.0000000 -0.2758417 -0.2079948 0.1107712
## SibSp -0.2758417  1.0000000  0.4529568 0.1571153
## Parch -0.2079948  0.4529568  1.0000000 0.2361560
## Fare   0.1107712  0.1571153  0.2361560 1.0000000
```

In statistics, two variables are strongly correlated if the correlation coefficient is either greater than 0.75 (some say 0.70 and some even say 0.8) or less than -0.75. Having a glance at the correlation matrix, none of the numeric features are strongly correlated. Hence, the **Multicollinearity** (a given feature in the model can be approximated by a linear combination of the other features in the model) does not exist among numeric features.

```
# Show the p-value of Chi Square tests
ps = chisq.test(train$Pclass, train$Sex)$p.value
pe = chisq.test(train$Pclass, train$Embarked)$p.value
pt = chisq.test(train$Pclass, train$Title)$p.value
pf = chisq.test(train$Pclass, train$FamilySize)$p.value
se = chisq.test(train$Sex, train$Embarked)$p.value
st = chisq.test(train$Sex, train$Title)$p.value
sf = chisq.test(train$Sex, train$FamilySize)$p.value
et = chisq.test(train$Embarked, train$Title)$p.value
ef = chisq.test(train$Embarked, train$FamilySize)$p.value
```

```
tf = chisq.test(train$Title, train$FamilySize)$p.value
cormatrix = matrix(c(0, ps, pe, pt, pf,
                     ps, 0, se, st, sf,
                     pe, se, 0, et, ef,
                     pt, st, et, 0, tf,
                     pf, sf, ef, tf, 0),
                   5, 5, byrow = TRUE)
row.names(cormatrix) = colnames(cormatrix) = c("Pclass", "Sex", "Embarked", "Title", "FamilySize")
cormatrix
```

```
##                  Pclass           Sex     Embarked         Title
## Pclass     0.000000e+00  2.532566e-03 1.053100e-23  5.962301e-10
## Sex        2.532566e-03  0.000000e+00 1.321593e-02 1.116723e-150
## Embarked   1.053100e-23  1.321593e-02 0.000000e+00  1.383169e-04
## Title      5.962301e-10 1.116723e-150 1.383169e-04  0.000000e+00
## FamilySize 1.108964e-10  4.591649e-15 2.490631e-06  2.204782e-51
##              FamilySize
## Pclass     1.108964e-10
## Sex        4.591649e-15
## Embarked   2.490631e-06
## Title      2.204782e-51
## FamilySize 0.000000e+00
```

I use Chi Square test to test the independence of factors/categorical features. Since all the p-values < 0.05, we reject each Ho:Two factors are independent at 5% significance level and indeed at any reasonable level of significance. This violates the independence assumption of features and can be confirmed that multicollinearity does exist among factors. I will deal with this issue down the road and now go ahead and fit the logistic regression model.

```
# Fitting Logistic Regression to the Training set
classifier = glm(Survived ~ ., family = binomial(link='logit'), data = train)

# Choosing the best model by AIC in a Stepwise Algorithm
# The step() function iteratively removes insignificant predictor variables from the model.
classifier <- step(classifier)
```

```
## Start:  AIC=612.29
## Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
##     Title + FamilySize
##
##              Df Deviance    AIC
## - SibSp       1   580.29 610.29
## - Embarked    2   582.69 610.69
## - Fare        1   580.81 610.81
## - Parch       1   581.59 611.59
## <none>            580.29 612.29
## - Sex         1   584.37 614.37
## - Age         1   585.69 615.69
## - FamilySize  2   590.68 618.68
## - Title       4   616.14 640.14
## - Pclass      2   624.73 652.73
##
## Step:  AIC=610.29
## Survived ~ Pclass + Sex + Age + Parch + Fare + Embarked + Title +
##     FamilySize
```

```
##
##              Df Deviance    AIC
## - Embarked    2   582.71 608.71
## - Fare        1   580.82 608.82
## - Parch       1   582.05 610.05
## <none>            580.29 610.29
## - Sex         1   584.37 612.37
## - Age         1   585.70 613.70
## - FamilySize  2   609.33 635.33
## - Title       4   616.76 638.76
## - Pclass      2   624.87 650.87
##
## Step:  AIC=608.71
## Survived ~ Pclass + Sex + Age + Parch + Fare + Title + FamilySize
##
##              Df Deviance    AIC
## - Fare        1   583.56 607.56
## - Parch       1   584.41 608.41
## <none>            582.71 608.71
## - Sex         1   586.56 610.56
## - Age         1   588.11 612.11
## - FamilySize  2   615.00 637.00
## - Title       4   619.32 637.32
## - Pclass      2   628.03 650.03
##
## Step:  AIC=607.56
## Survived ~ Pclass + Sex + Age + Parch + Title + FamilySize
##
##              Df Deviance    AIC
## - Parch       1   585.45 607.45
## <none>            583.56 607.56
## - Sex         1   587.31 609.31
## - Age         1   589.24 611.24
## - FamilySize  2   615.02 635.02
## - Title       4   619.43 635.43
## - Pclass      2   662.23 682.23
##
## Step:  AIC=607.45
## Survived ~ Pclass + Sex + Age + Title + FamilySize
##
##              Df Deviance    AIC
## <none>            585.45 607.45
## - Sex         1   589.15 609.15
## - Age         1   591.29 611.29
## - Title       4   623.47 637.47
## - FamilySize  2   622.80 640.80
## - Pclass      2   664.64 682.64
```

```
summary(classifier)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Age + Title + FamilySize,
##     family = binomial(link = "logit"), data = train)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6190  -0.5712  -0.3802   0.5462   2.4571
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      18.72233  506.79974   0.037 0.970531
## Pclass2          -1.42636    0.32191  -4.431 9.38e-06 ***
## Pclass3          -2.55761    0.31174  -8.204 2.32e-16 ***
## Sexmale         -14.63628  506.79929  -0.029 0.976960
## Age              -0.02518    0.01062  -2.370 0.017789 *
## TitleMiss       -15.04068  506.79960  -0.030 0.976324
## TitleMr          -3.36409    0.60123  -5.595 2.20e-08 ***
## TitleMrs        -14.59001  506.79969  -0.029 0.977033
## TitleOther       -2.96720    0.85828  -3.457 0.000546 ***
## FamilySizeSmall  -0.23457    0.25791  -0.909 0.363087
## FamilySizeLarge  -2.65324    0.50371  -5.267 1.38e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 949.90  on 712  degrees of freedom
## Residual deviance: 585.45  on 702  degrees of freedom
## AIC: 607.45
##
## Number of Fisher Scoring iterations: 13
```

Mmm... The factor **Sex** is not statistically significant at any reasonable level of significance (p-value = 0.976960 > 0.05 or 0.01 or even 0.1); however, it is still in the best model. Furthermore, notice the standard error of **Sexmale**, **TitleMiss** and **TitleMrs** are very large. This has something (No!!! everything) to do with multicollinearity. As an effect of multicollinearity, the standard error (hence, the variance) of model coefficients for Sexmale, TitleMiss and TitleMrs became very large.

```
vif(classifier)
```

```
##                     GVIF Df GVIF^(1/(2*Df))
## Pclass      1.667655e+00  2        1.136388
## Sex         5.751701e+06  1     2398.270330
## Age         1.894599e+00  1        1.376444
## Title       1.224494e+07  4        7.691208
## FamilySize  1.812345e+00  2        1.160273
```

Ah hah! `vif` function delivers very high Generalized Variable Inflation Factor (GVIF) for factors Sex and Title. This confirmes the multicollinearity between factors **Sex** and **Title**.

How to handle multicollinearity?

We want to make our model robust. I omit the factor **Sex** from the logistic regression model because it exhibits a high degree of multicollinearity.

```
# Fitting Logistic Regression to the Training set again without the factor Sex
classifier = glm(Survived ~ . -Sex, family = binomial(link='logit'), data = train)

# Choosing the best model by AIC in a Stepwise Algorithm
# The step() function iteratively removes insignificant features from the model.
classifier <- step(classifier)
```

```
## Start:  AIC=614.37
## Survived ~ (Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
##     Title + FamilySize) - Sex
##
##              Df Deviance    AIC
## - SibSp       1   584.37 612.37
## - Embarked    2   586.52 612.52
## - Fare        1   584.83 612.83
## - Parch       1   585.58 613.58
## <none>            584.37 614.37
## - Age         1   589.95 617.95
## - FamilySize  2   594.60 620.60
## - Pclass      2   629.59 655.59
## - Title       4   772.00 794.00
##
## Step:  AIC=612.37
## Survived ~ Pclass + Age + Parch + Fare + Embarked + Title + FamilySize
##
##              Df Deviance    AIC
## - Embarked    2   586.56 610.56
## - Fare        1   584.84 610.84
## - Parch       1   586.10 612.10
## <none>            584.37 612.37
## - Age         1   589.95 615.95
## - FamilySize  2   613.53 637.53
## - Pclass      2   629.78 653.78
## - Title       4   772.02 792.02
##
## Step:  AIC=610.56
## Survived ~ Pclass + Age + Parch + Fare + Title + FamilySize
##
##              Df Deviance    AIC
## - Fare        1   587.31 609.31
## - Parch       1   588.22 610.22
## <none>            586.56 610.56
## - Age         1   592.09 614.09
## - FamilySize  2   618.78 638.78
## - Pclass      2   632.93 652.93
## - Title       4   783.98 799.98
##
## Step:  AIC=609.31
## Survived ~ Pclass + Age + Parch + Title + FamilySize
##
##              Df Deviance    AIC
## - Parch       1   589.15 609.15
## <none>            587.31 609.31
## - Age         1   593.14 613.14
## - FamilySize  2   618.78 636.78
## - Pclass      2   667.53 685.53
## - Title       4   785.52 799.52
##
## Step:  AIC=609.15
## Survived ~ Pclass + Age + Title + FamilySize
##
```

```
##              Df Deviance    AIC
## <none>            589.15 609.15
## - Age         1   595.17 613.17
## - FamilySize  2   626.64 642.64
## - Pclass      2   669.92 685.92
## - Title       4   793.80 805.80
```

```
summary(classifier)
```

```
##
## Call:
## glm(formula = Survived ~ Pclass + Age + Title + FamilySize, family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6304  -0.5750  -0.3792   0.5637   2.4607
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      4.11423    0.69575   5.913 3.35e-09 ***
## Pclass2         -1.46793    0.32032  -4.583 4.59e-06 ***
## Pclass3         -2.58021    0.31125  -8.290  < 2e-16 ***
## Age             -0.02543    0.01059  -2.403  0.01627 *
## TitleMiss       -0.40382    0.55940  -0.722  0.47037
## TitleMr         -3.36730    0.60132  -5.600 2.15e-08 ***
## TitleMrs         0.05208    0.63190   0.082  0.93431
## TitleOther      -2.56210    0.81122  -3.158  0.00159 **
## FamilySizeSmall -0.23202    0.25621  -0.906  0.36516
## FamilySizeLarge -2.65769    0.50374  -5.276 1.32e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 949.90  on 712  degrees of freedom
## Residual deviance: 589.15  on 703  degrees of freedom
## AIC: 609.15
##
## Number of Fisher Scoring iterations: 5
```

```
vif(classifier)
```

```
##                GVIF Df GVIF^(1/(2*Df))
## Pclass     1.688094  2        1.139854
## Age        1.909003  1        1.381667
## Title      2.618396  4        1.127858
## FamilySize 1.805038  2        1.159102
```

```
durbinWatsonTest(classifier)
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      0.02148659      1.956557   0.576
##  Alternative hypothesis: rho != 0
```

The model looks good. The standard errors are in a reasonable range. GVIF values are all less than 5. Furthermore, since Durbin-Watson test results with D-W Statistic 1.96 and p-value > 0.05, we do not reject

Ho:Residuals are not autocorrelated. Hence, we can conclude there is sufficient evidence to say residuals are not autocorrelated. Hooray! The assumptions are checked and they are passed.

According to the best model, the features **Pclass**, **Age**, **Title** and **FamilySize** significantly contribute to the model in predicting survival. We will see how well the model predicts on new data in the validation set.

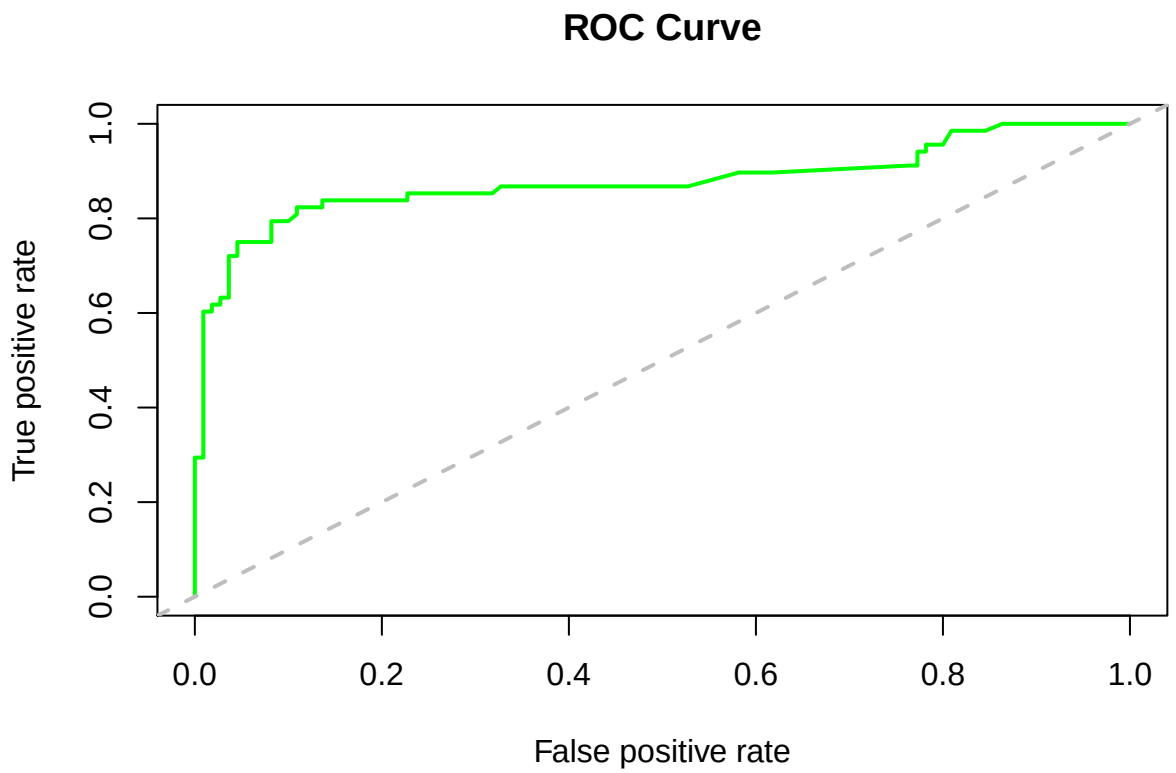
```
# Predicting the Validation set results
prob_pred = predict(classifier, type = 'response', newdata = test)
y_pred = ifelse(prob_pred > 0.5, 1, 0)

# Checking the prediction accuracy
table(test$Survived, y_pred > 0.5) # Confusion matrix
```

```
##
##     FALSE TRUE
##   0   101    9
##   1    17   51
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.8539"
```

```
# Use the predictions to build a ROC curve to assess the performance of our model
fitpred = prediction(prob_pred, test$Survived)
fitperf = performance(fitpred,"tpr","fpr")
plot(fitperf,col="green",lwd=2,main="ROC Curve")
abline(a=0,b=1,lwd=2,lty=2,col="gray")
```



The ROC (Receiver Operating Characteristics) curve is a graphical representation of the performnace of the classifier and it shows the performance of our model rises well above the diagonal line. This indicates that our logistic regression model performs better than just a random guess. The logistic regression model delivers a whooping 0.8539 accuracy interms of predicting the survival.

Support Vector Machines

Secondly, I use Support Vector Machines (SVM) for classification. In order to use SVM, we need to remember to do one thing - Feature Scaling! Because the SVM classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters.

```
# Checking the variance of numeric features
paste('Age variance: ',var(train$Age),', SibSp variance: ',var(train$SibSp),', Parch variance: ',var(tra
```

```
## [1] "Age variance:  173.992892791181 , SibSp variance:  1.26512441495816 , Parch variance:  0.604594
```

The variances of **Age** and **Fare** seem very high. Let's do feature scaling to standardize these features so that all features are on the same scale and no feature is dominated by the other.

```
# Feature Scaling - use scale() to standardize the feature columns
standardized.train = cbind(select(train, Survived, Pclass, Sex, SibSp, Parch, Embarked, Title, FamilySi
paste('Age variance: ',var(standardized.train$Age),', Fare variance: ',var(standardized.train$Fare))
```

```
## [1] "Age variance:  1 , Fare variance:  1"
```

```
standardized.test = cbind(select(test, Survived, Pclass, Sex, SibSp, Parch, Embarked, Title, FamilySize
paste('Age variance: ',var(standardized.test$Age),', Fare variance: ',var(standardized.test$Fare))
```

```
## [1] "Age variance:  1 , Fare variance:  1"
```

Now that we have done the feature scaling, we can fit SVM to predict survival. First I fit a linear SVM.

```
# Fitting Linear SVM to the Training set
classifier = svm(Survived ~ .,
                 data = standardized.train,
                 type = 'C-classification',
                 kernel = 'linear')

# Predicting the Validation set results
y_pred = predict(classifier, newdata = standardized.test[,-which(names(standardized.test)=="Survived")]

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

```
##    y_pred
##       0   1
##   0 106   4
##   1  18  50
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.8764"
```

Wow! The linear model accuracy is 0.8764. That is great and I am happy with the result. Let's fit a non-linear radial kernel and see whether the accuracy will be improved.

```
# Fitting Non-linear SVM to the Training set
classifier = svm(Survived ~ .,
```

```
                 data = standardized.train,
                 type = 'C-classification',
                 kernel = 'radial')

# Predicting the Validation set results
y_pred = predict(classifier, newdata = standardized.test[,-which(names(standardized.test)=="Survived")])

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

```
##    y_pred
##       0   1
##   0 105   5
##   1  16  52
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.882"
```

Yes! The accuracy has been improved by 1% to 0.8820. Also, note that both linear SVM and non-linear SVM accuracies are higher than the accuracy for logistic regression model.

How about Parameter Tuning?

The best non-linear SVM performance occurs with cost=1 and gamma=0.0625. Now I tune these parameters to attempt to improve our model (remember, our model is already a good model).

```
# Tuning the model
# Applying Grid Search to find the best parameters
tune.results <- tune(svm,
                     Survived ~ .,
                     data = standardized.train,
                     kernel='radial',
                     ranges=list(cost=2^(-2:2), gamma=2^(-6:-2)))
summary(tune.results)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     4 0.125
##
## - best performance: 0.1765258
##
## - Detailed performance results:
##    cost    gamma     error dispersion
## 1  0.25 0.015625 0.1976135 0.04893833
## 2  0.50 0.015625 0.1934077 0.05127991
## 3  1.00 0.015625 0.1905908 0.04961131
## 4  2.00 0.015625 0.1835876 0.04623116
## 5  4.00 0.015625 0.1849961 0.04642873
## 6  0.25 0.031250 0.1905712 0.05042353
## 7  0.50 0.031250 0.1905908 0.04602550
```

```
## 8  1.00 0.031250 0.1863654 0.05205927
## 9  2.00 0.031250 0.1849765 0.04587358
## 10 4.00 0.031250 0.1863850 0.04917736
## 11 0.25 0.062500 0.1877543 0.04725138
## 12 0.50 0.062500 0.1891823 0.04201056
## 13 1.00 0.062500 0.1863850 0.04917736
## 14 2.00 0.062500 0.1863654 0.04809803
## 15 4.00 0.062500 0.1891823 0.05460939
## 16 0.25 0.125000 0.1919601 0.04479708
## 17 0.50 0.125000 0.1863654 0.04537236
## 18 1.00 0.125000 0.1864045 0.04872506
## 19 2.00 0.125000 0.1821596 0.05043104
## 20 4.00 0.125000 0.1765258 0.04637780
## 21 0.25 0.250000 0.1975939 0.03596626
## 22 0.50 0.250000 0.1877347 0.04984047
## 23 1.00 0.250000 0.1779147 0.04983886
## 24 2.00 0.250000 0.1821205 0.05190484
## 25 4.00 0.250000 0.1905516 0.04678710
```

```
# The best non-linear SVM performance occurs with cost=4 and gamma=0.125

# Fitting Non-linear SVM to the Training set
classifier = svm(Survived ~ .,
                 data = standardized.train,
                 type = 'C-classification',
                 kernel = 'radial',
                 cost = 4,
                 gamma = 0.125)

# Predicting the Validation set results
y_pred = predict(classifier, newdata = standardized.test[,-which(names(standardized.test)=="Survived")])

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

```
##    y_pred
##       0   1
##   0 104   6
##   1  24  44
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.8315"
```

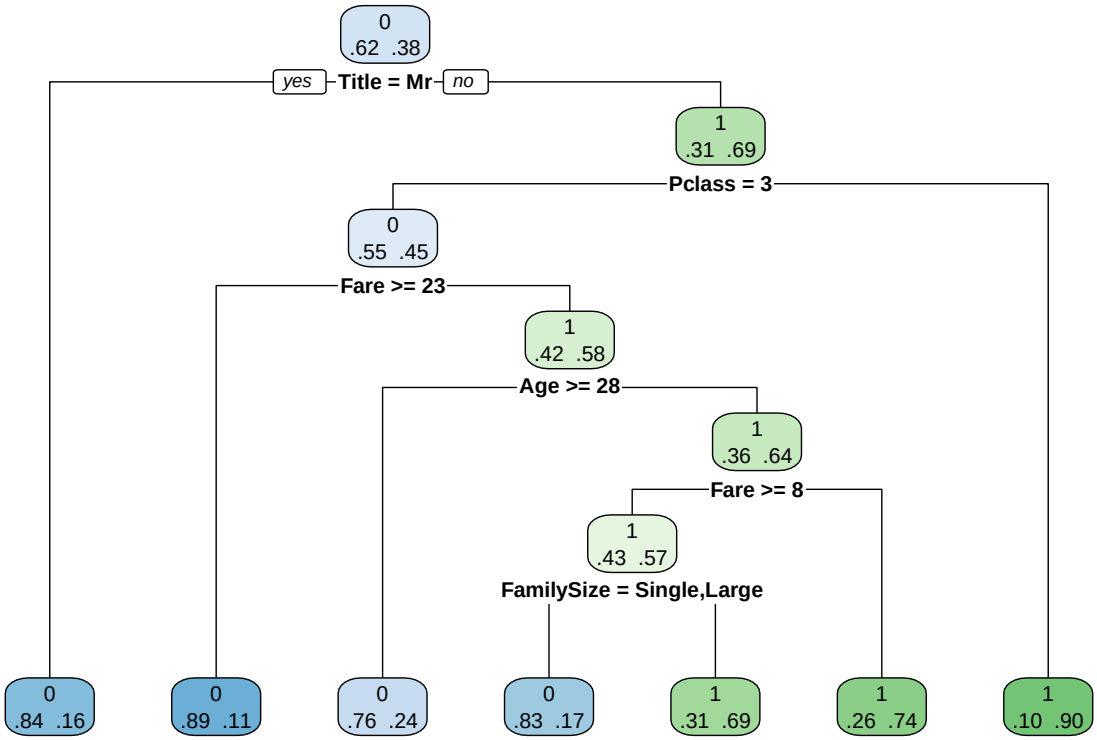
The accuracy went down to 0.8315. We were not able to improve our model. Infact, the best non-linear SVM was already a good model with accuracy 0.8820. I retain that model as my best non-linear SVM model with cost=1 and gamma=0.0625.

Decision Tree

As we all know, Random Forest is a more powerful algorithm over just a single tree. However, the Decision Tree classification preserve the interpretability which the random forest algorithm lacks.

The Decision Tree does not require feature scaling. Let's fit a decision tree model to our training data.

```
# Fitting Decision Tree Classification Model to the Training set
classifier = rpart(Survived ~ ., data = train, method = 'class')

# Tree Visualization
rpart.plot(classifier, extra=4)
```



The single tree uses five features **Title**, **Pclass**, **Fare**, **Age** and **FamilySize** for classification. Let's see how well our model performs with the data in the validation set.

```
# Predicting the Validation set results
y_pred = predict(classifier, newdata = test[,-which(names(test)=="Survived")], type='class')

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

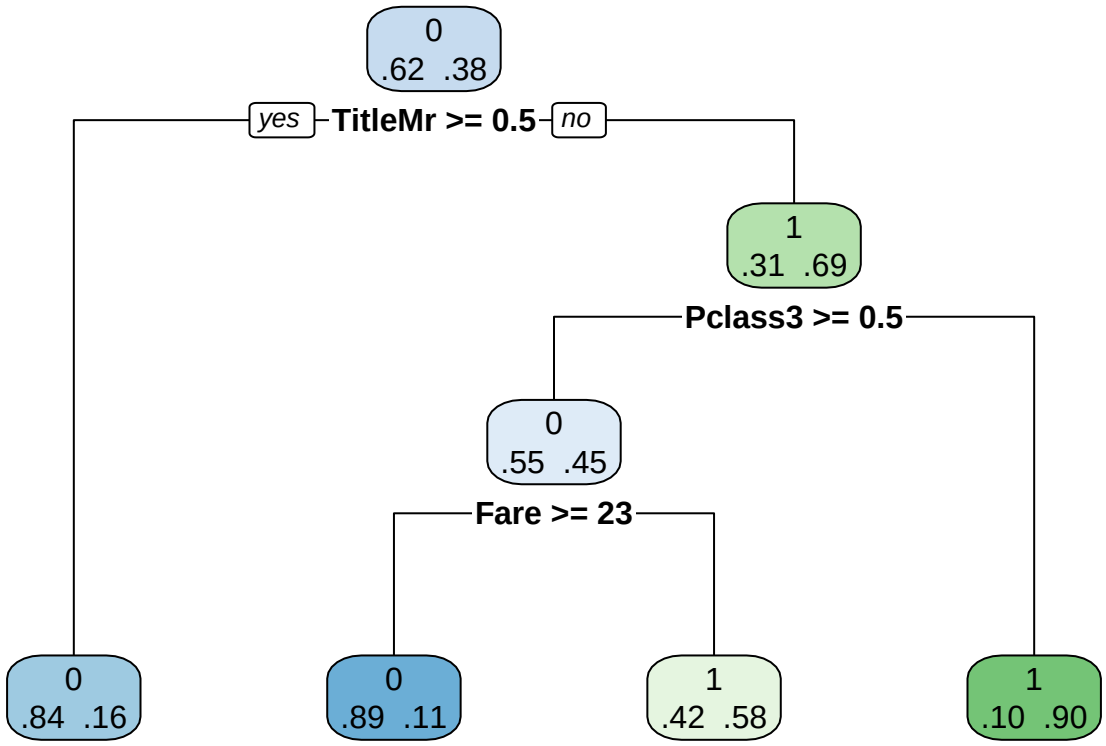
```
##    y_pred
##       0   1
##   0 102   8
##   1  21  47
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.8371"
```

Accuracy of a single tree is 0.8371. Overfitting can easily occur in Decision Tree classification. We can idenfity that evaluating the model using **k-Fold Cross Validation**. Or we might be able to improve the model. Let's do 10-fold cross validation to find out whether we could improve the model.

```
# Applying k-Fold Cross Validation
set.seed(789)
folds = createMultiFolds(train$Survived, k = 10, times = 5)
control <- trainControl(method = "repeatedcv", index = folds)
classifier_cv <- train(Survived ~ ., data = train, method = "rpart", trControl = control)

# Tree Visualization
rpart.plot(classifier_cv$finalModel, extra=4)
```




```
# Predicting the Validation set results
y_pred = predict(classifier_cv, newdata = test[,-which(names(test)=="Survived")])

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

```
##    y_pred
##      0  1
##   0 99 11
##   1 17 51
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

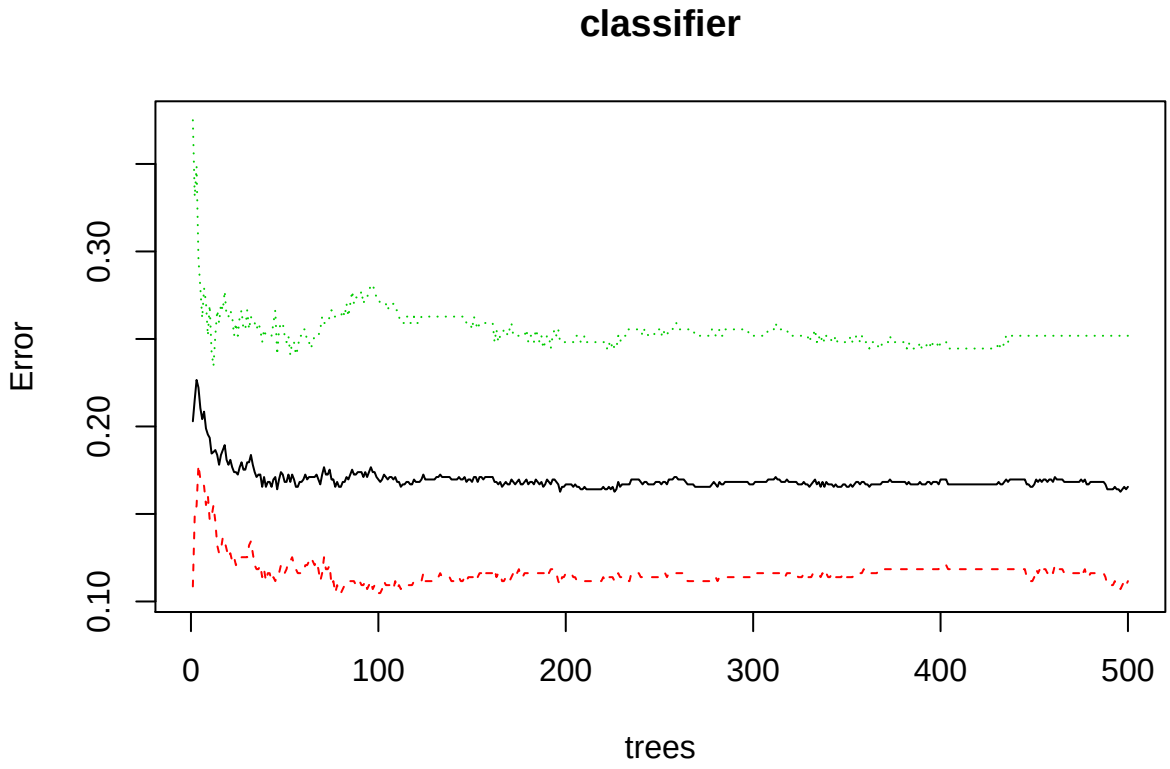
```
## [1] "Accuracy 0.8427"
```

We were able to improve the model after 10-fold cross validation. The accuracy has been improved to 0.8427 but note the improved model uses only three features **Title**, **Pclass** and **Fare** for classification.

Random Forests

Random forests improve predictive accuracy by generating a large number of bootstrapped trees (based on random samples of variables). Random Forest is a prowerful machine learning algorithm which holds a relatively high classification accuracy.

```
# Fitting Random Forest Classification to the Training set
set.seed(432)
classifier = randomForest(Survived ~ ., data = train)

# Choosing the number of trees
plot(classifier)
```



The green, black and red lines represent error rate for death, overall and survival, respectively. The overall error rate converges to around 17%. Interestingly, our model predicts death better than survival. Since the overall error rate converges to a constant and does not seem to further decrease, our choice of default 500 trees in the `randomForest` function is a good choice.

```
# Predicting the Validation set results
y_pred = predict(classifier, newdata = test[,-which(names(test)=="Survived")])

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

```
##    y_pred
##       0   1
##   0 100  10
##   1  18  50
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.8427"
```

The accuracy is 0.8427 and which is greater than the accuracy of just a single tree 0.8371 (without 10-fold cross validation). Let's see if 10-fold cross validation can improve our model as it did for the Decision Tree classification.

```
# Applying k-Fold Cross Validation
set.seed(651)
folds = createMultiFolds(train$Survived, k = 10)
control <- trainControl(method = "repeatedcv", index = folds)
classifier_cv <- train(Survived ~ ., data = train, method = "rf", trControl = control)

# Predicting the Validation set results
y_pred = predict(classifier_cv, newdata = test[,-which(names(test)=="Survived")])

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

```
##    y_pred
##      0  1
##   0 97 13
##   1 18 50
```

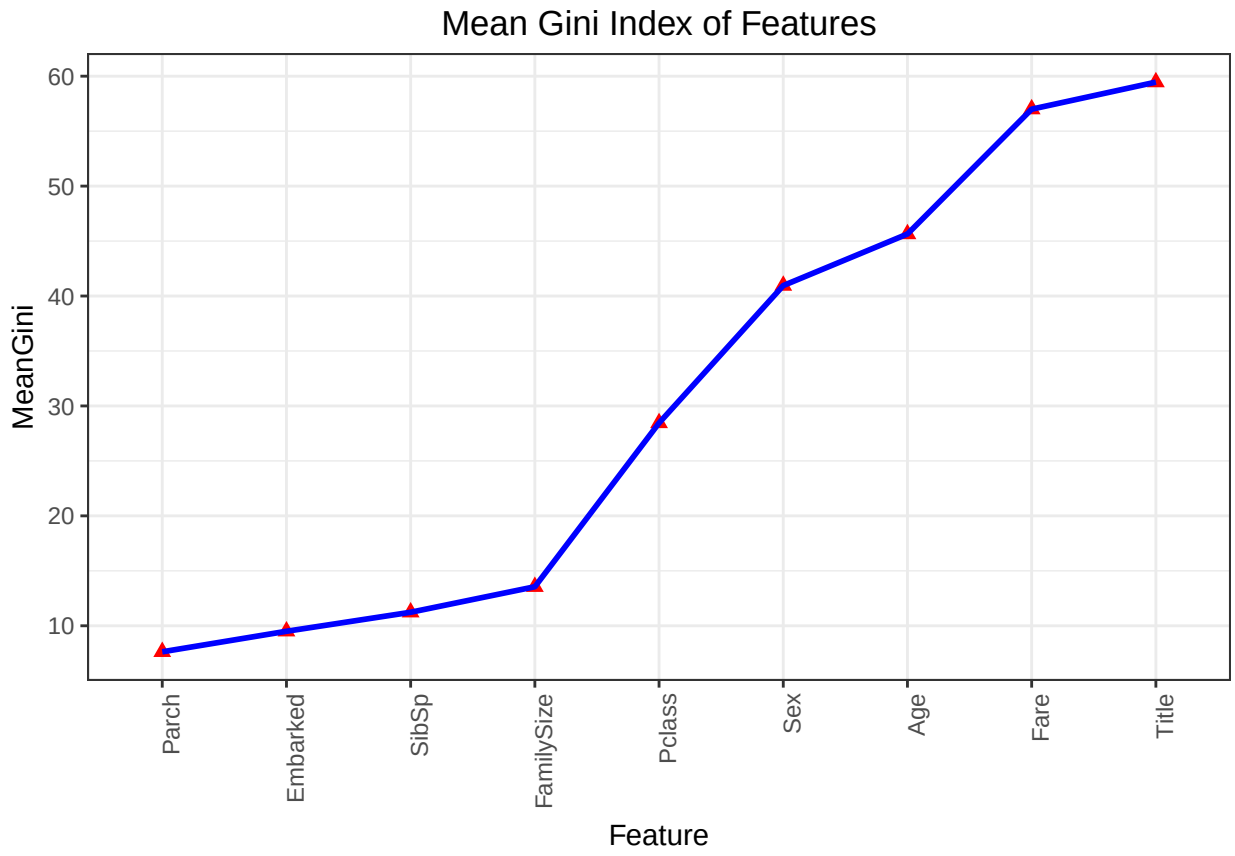
```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.8258"
```

Accuracy went down to 0.8258. We were not able to improve the random forest model using 10-fold cross validation.

As mentioned previously in the Decision Tree section, the random Forest classification suffers in terms of interpretability. We are unable to visualize the 500 trees and identify important features of the model. However, we can assess the **Feature Importance** using the Gini index measure. Let's plot mean Gini index across all trees and identify important features.

```
# Feature Importance
gini = as.data.frame(importance(classifier))
gini = data.frame(Feature = row.names(gini),
                  MeanGini = round(gini[ ,'MeanDecreaseGini'], 2))
gini = gini[order(-gini[,"MeanGini"]),]

ggplot(gini,aes(reorder(Feature,MeanGini), MeanGini, group=1)) +
  geom_point(color='red',shape=17,size=2) +
  geom_line(color='blue',size=1) +
  scale_y_continuous(breaks=seq(0,60,10)) +
  xlab("Feature") +
  ggtitle("Mean Gini Index of Features") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The feature **Title** has the highest mean gini index, hence the highest importance. **Fare** is also realtively high important and it is followed by **Age** of the passengers.

Naive Bayes

Lastly, I use Naive Bayes algorithm to predict survival. Naive Bayes classification is a simple but effective algorithm; it is faster compared to many other iterative algorithms; it does not need feature scaling; and its foundation is the Bayes Theorem.

However, Naive Bayes is based on the assumption that conditional probability of each feature given the class is independent of all the other features. The assumption of independent conditional probabilities means the features are completely independent of each other. This assumption was already checked in the Logistic Regression section and we have found that numeric features are independent to each other, however, the categorical features are not. By assuming the idependence assumption of all the features, let's fit a naive bayes model to our training data.

```
# Fitting Naive Bayes to the Training set
classifier = naiveBayes(Survived ~ ., data = train)

# Predicting the Validation set results
y_pred = predict(classifier, newdata = test[,-which(names(test)=="Survived")])

# Checking the prediction accuracy
table(test$Survived, y_pred) # Confusion matrix
```

```
##    y_pred
##      0  1
```

```
##   0 99 11
##   1 17 51
```

```
error <- mean(test$Survived != y_pred) # Misclassification error
paste('Accuracy',round(1-error,4))
```

```
## [1] "Accuracy 0.8427"
```

Wow! Naive Bayes classification performs well for our validation data with an accuracy of 0.8427. Note that the accuracy is identical to the Random Forest classification accuracy (without 10-fold cross validation).

Discussion

Comparison of models

Logistic Regression

- The assumptions were met and the accuracy of the best model is 0.8539.
- Four features, **Title**, **Pclass**, **Age** and **FamilySize**, significantly contribute to the model in predicting survival.
- Confusion matix consists of 9 false positives and 17 false negatives.

Linear SVM

- Scaled some features and gained an accuracy of 0.8764.
- Confusion matix consists of 4 false positives and 18 false negatives.

Non-linear Radial SVM

- Scaled some features and secured the highest accuracy of 0.8820.
- Parameter tuning did not improve the model accuracy.
- Confusion matix consists of 5 false positives and 16 false negatives.

Decision Tree

- Were able to improve the model using 10-fold cross validation.
- The accuracy was improved from 0.8371 to 0.8427 and the improved model uses three features, **Title**, **Fare** and **Pclass**, for classification.
- Confusion matix consists of 11 false positives and 17 false negatives.

Random Forest

- Gained an accuracy of 0.8427 and 10-fold cross validation did not improve the model accuracy.
- The first five important features based on their importance (highest to lowest) are as follows: **Title**, **Fare**, **Age**, **Sex** and **Pclass**.
- Confusion matix consists of 10 false positives and 18 false negatives.

Naive Bayes

- Gained an accuracy of 0.8427 which is identical to the Random Forest accuracy.
- The independence among features was assumed.
- Confusion matix consists of 11 false positives and 17 false negatives.

Results

```
# Predicting the Test set results
y_pred = predict(classifier, newdata = test_original)

# Save the results
results <- data.frame(PassengerID = titanic[892:1309,"PassengerId"], Survived = y_pred)

# Write the results to a csv file
write.csv(results, file = 'PredictingTitanicSurvival.csv', row.names = FALSE, quote=FALSE)
```

Once I submitted my results from above six models, suprisingly, the worst performance was given by our best performer of the Validation Set, Non-linear SVM. Logistic Regression, Linear SVM, Decision Tree and Random Forest executed roughly the same score and closely followed by the Naive Bayes score.

Conclusion

I am super happy I have finished my first kaggle kernel. With this exploration, I have acquired a great deal of knowledge about Machine Learning algorithms and their pros and cons. Also, I have developed skills on handling missing data, feature engineering and exploratory data analysis.

I thank the kaggle community for publicly sharing the scripts which greatly helps new machine learning folks like me to learn and explore the field further.

Thank you everyone for reading my first kaggle kernel. I would be very much appreciated if you could upvote if you found my kernel useful or you just liked it. That will keep me motivated :) Also, I welcome your comments and suggestions!