Ensemble Modelling with Caret (top 3% score)

David Marshall

Introduction

This is an attempt to use machine learning on the Titanic dataset, which classifies passenegers on the titanic by whether they died or survived. The script uses ensemble modelling and currently has a score in the top 3%.

```
# Load the data
library(readr)
titanic <- read_csv("train.csv", col_types =
                    cols(Embarked = col_factor(levels = c("S", "C", "Q")),
                         PassengerId = col_character(),
                         Pclass = col_factor(levels = c("1", "2", "3")),
                         Sex = col_factor(levels = c("male","female")),
                         Survived = col_factor(levels = c("0", "1"))))
summary(titanic)
```

I will first change the survived variable to make it clear which is which, and convert to factor.

```
# Change the survived variable
titanic$Survived <- ifelse(titanic$Survived == 1, "Survived", "Died")
titanic$Survived <- as.factor(titanic$Survived)
```

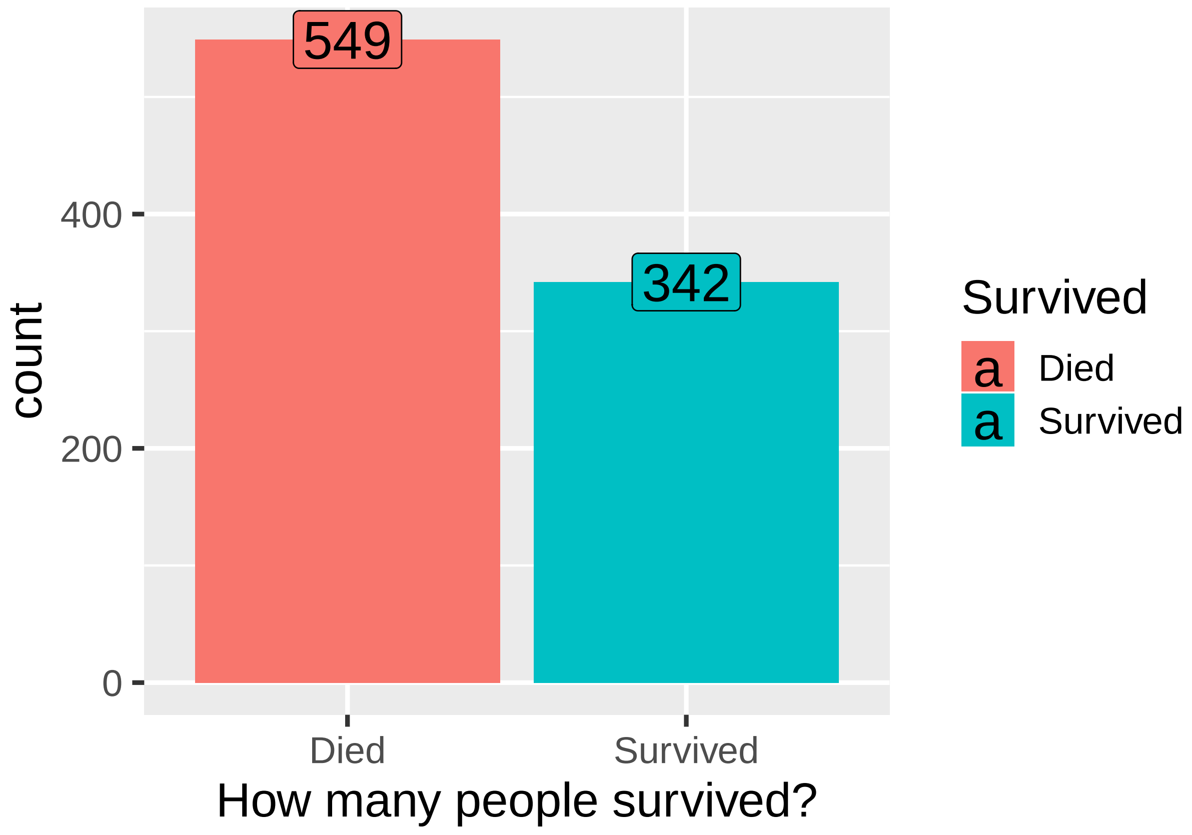
Data Exploration

Now I will begin to explore the data starting with the dependent survived variable. I will use ggplot2 from the tidyverse to explore the data.

```
# Let's start by exploring the data
library(tidyverse)
```

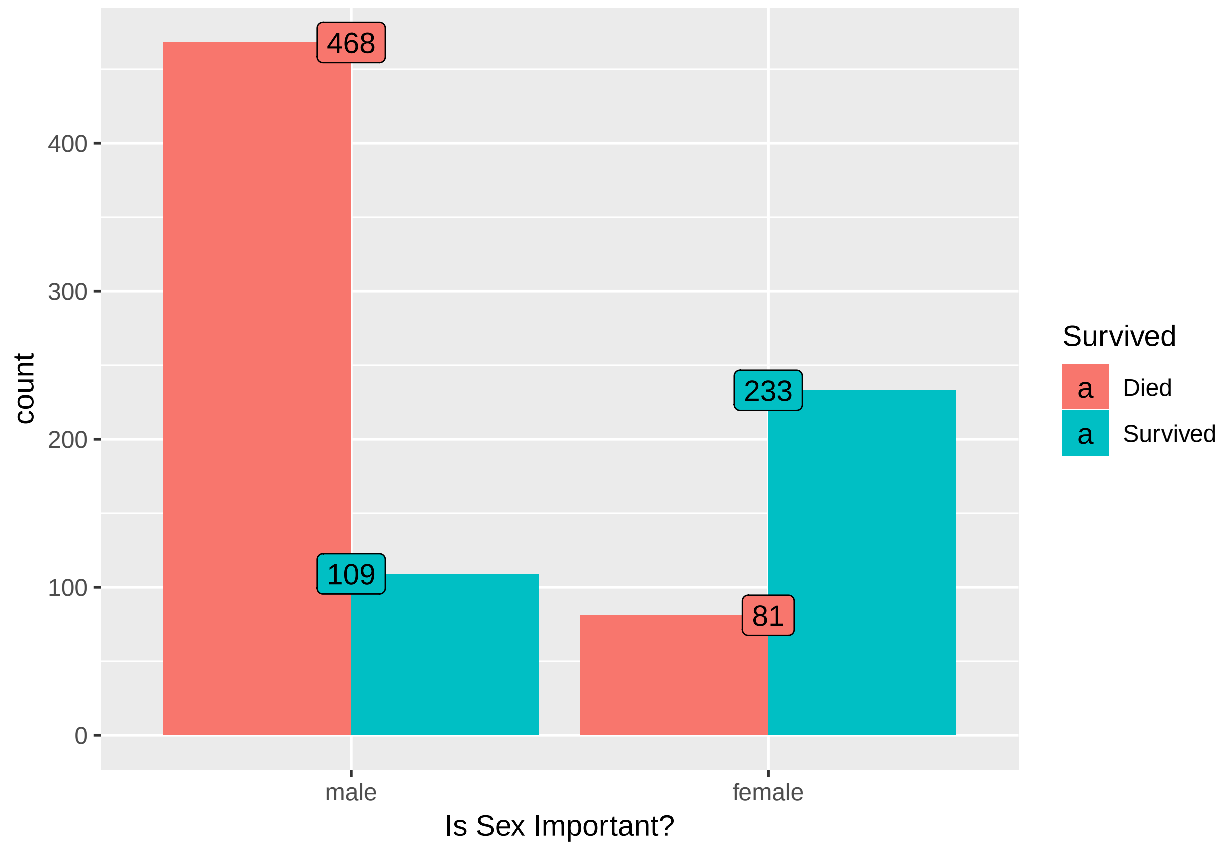
I will generate a plot and add a label, changing the theme:

```
p1 <- ggplot(titanic, aes(Survived, fill = Survived)) +
  geom_bar(stat = 'count') +
  labs(x = "How many people survived?") +
  geom_label(stat = 'count', aes(label = ..count..), size = 7) +
  theme_grey(base_size = 18)
p1
```



There is a slight imbalance in the classes in that more people died than survived. I will address this later. Perhaps if we look at sex a a predictor.

```r
# Now let's explore sex
p2 <- ggplot(titanic, aes(x = Sex, fill = Survived)) +
  geom_bar(stat = 'count', position = 'dodge') + theme_grey() +
  labs(x = "Is Sex Important?") +
  geom_label(stat = 'count', aes(label = ..count..))
p2
```

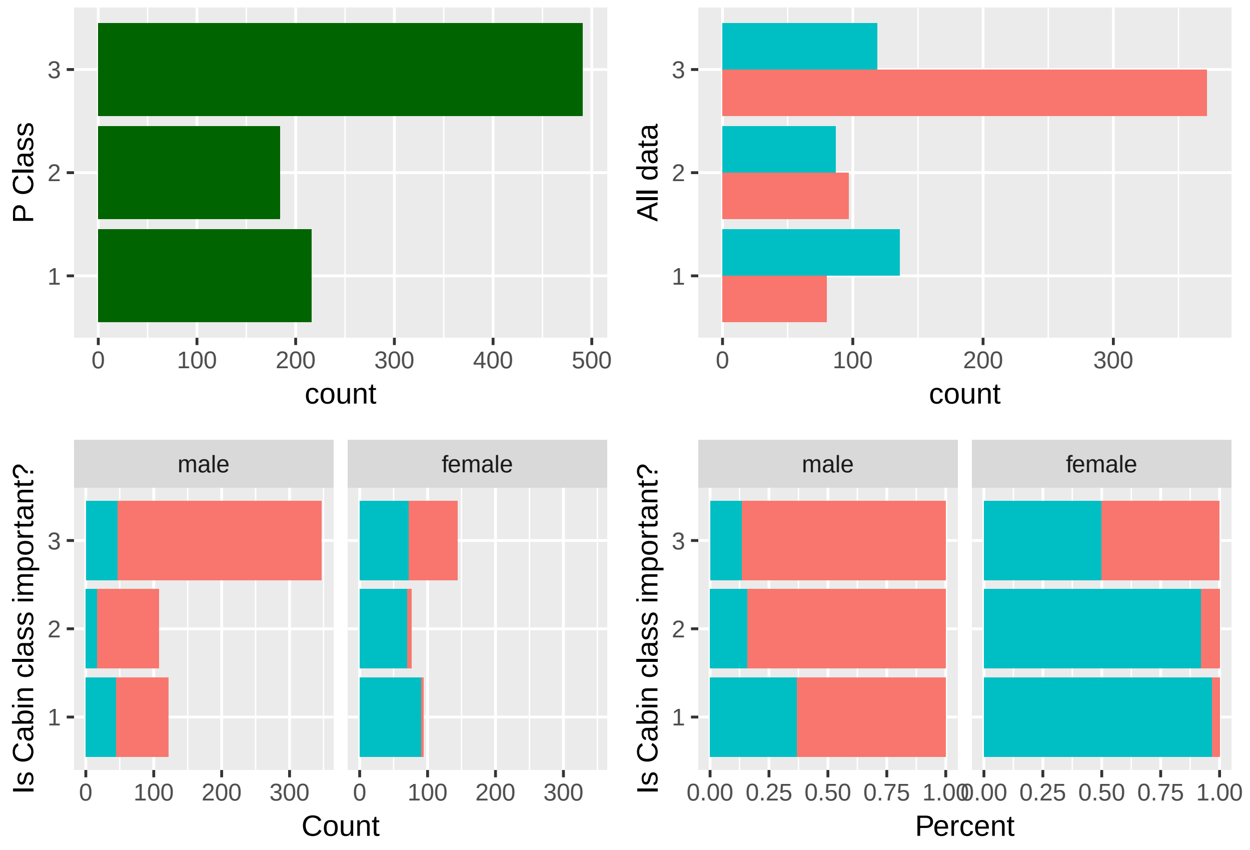


It appears that sex is important. Women and children were prioritised and it is these people who were more liekly to survive. We can also examine the class of passenger (ticket) and its impact on survival. I will create four plots and put them together with *gridExtra*.

```
# How about class?
p3 <- ggplot(titanic, aes(x = Pclass, fill = "green")) +
  geom_bar(stat='count', position='dodge', show.legend = FALSE, fill = "darkgreen") +
  labs(x = 'P Class') +
  theme(legend.position="none") + coord_flip()
p4 <- ggplot(titanic, aes(x = Pclass, fill = Survived)) +
  geom_bar(stat='count', position='dodge', show.legend = FALSE) + labs(x = 'All data') +
  theme(legend.position="none") + theme_grey()+ coord_flip()
p5 <- ggplot(titanic, aes(x = Pclass, fill = Survived)) +
  geom_bar(stat='count', position='stack', show.legend = FALSE) +
  labs(x = 'Is Cabin class important?', y= "Count") + facet_grid(.~Sex) +
  theme(legend.position="pclass") + theme_grey()+ coord_flip()
p6 <- ggplot(titanic, aes(x = Pclass, fill = Survived)) +
  geom_bar(stat='count', position='fill', show.legend = FALSE) +
  labs(x = 'Is Cabin class important?', y= "Percent") + facet_grid(.~Sex) +
  theme(legend.position="none") + theme_grey()+ coord_flip()
```

I can use the function *grid.arrange to display 4 functions at once:

```
library(gridExtra)
grid.arrange(p3, p4, p5, p6, ncol=2)
```



It appears that class is important. People in first class were less likely to die. For that reason I will combine these 2 variables into another separate variable (it seems common to do this on other kernels)

```r
# We can combine class and sex
titanic$PclassSex[titanic$Pclass== 1  & titanic$Sex=='male'] <- 'P1Male'
titanic$PclassSex[titanic$Pclass== 2  & titanic$Sex=='male'] <- 'P2Male'
titanic$PclassSex[titanic$Pclass== 3  & titanic$Sex=='male'] <- 'P3Male'
titanic$PclassSex[titanic$Pclass== 1  & titanic$Sex=='female'] <- 'P1Female'
titanic$PclassSex[titanic$Pclass== 2  & titanic$Sex=='female'] <- 'P2Female'
titanic$PclassSex[titanic$Pclass== 3  & titanic$Sex=='female'] <- 'P3Female'
titanic$PclassSex <- as.factor(titanic$PclassSex)
```

Feature Engineering

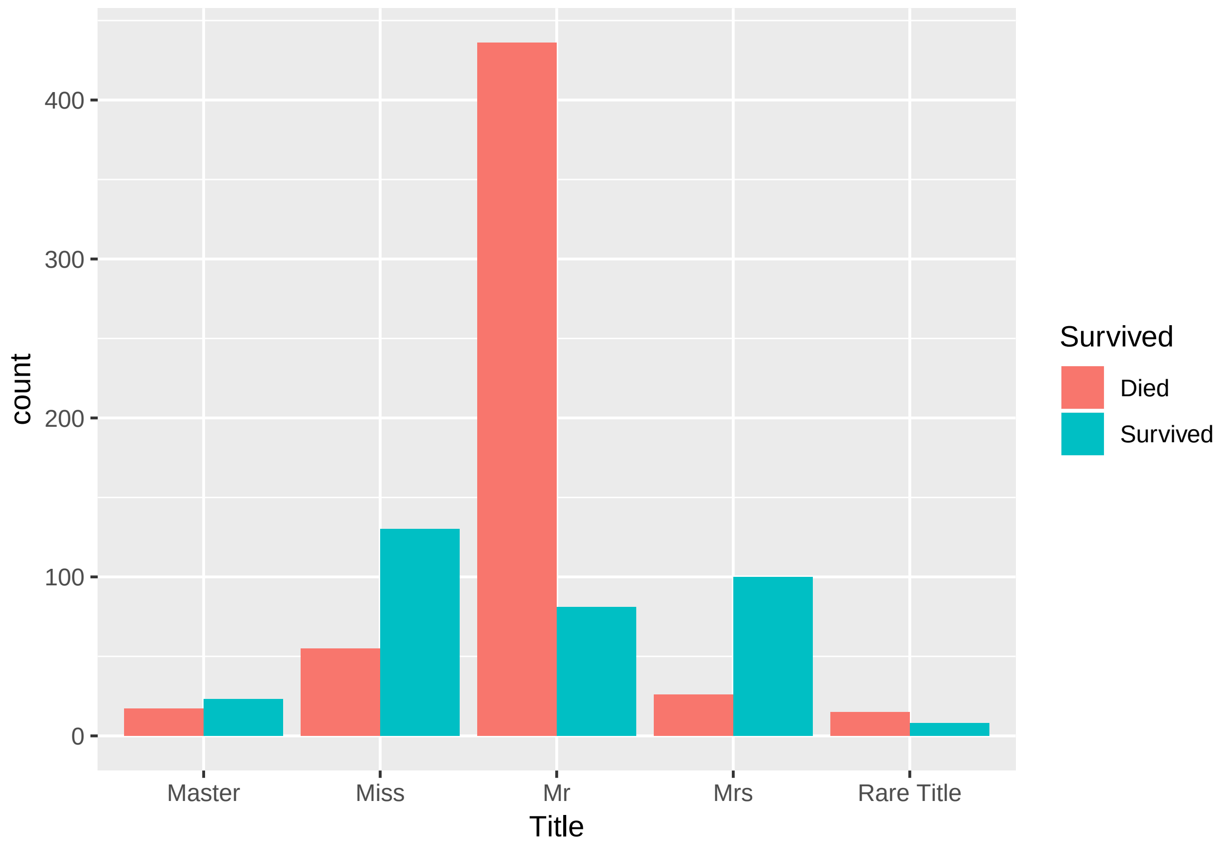
It is also helpful to do some feature engineering before plotting the models in order to get more out of the data. The title of the person from name might be important, since women and children were prioritised. I will first take out the surname, then the title, then reduce the number of titles by combining ones we might consider similar.

```r
# We can also do feature engineering, first taking the title and surname
titanic$Surname <- sapply(titanic$Name, function(x) {strsplit(x, split='[,.]')[[1]][1]})

#correcting some surnames that also include a maiden name
titanic$Surname <- sapply(titanic$Surname, function(x) {strsplit(x, split='[-]')[[1]][1]})

# Now to get the title of each person
```

```r
titanic$Title <- sapply(titanic$Name, function(x) {strsplit(x, split='[,.]')[[1]][2]})
titanic$Title <- sub(' ', '', titanic$Title) #removing spaces before title

# Now to reduce the number of titles
titanic$Title[titanic$Title %in% c("Mlle", "Ms")] <- "Miss"
titanic$Title[titanic$Title== "Mme"] <- "Mrs"
titanic$Title[!(titanic$Title %in% c('Master', 'Miss', 'Mr', 'Mrs'))] <- "Rare Title"
titanic$Title <- as.factor(titanic$Title)
```

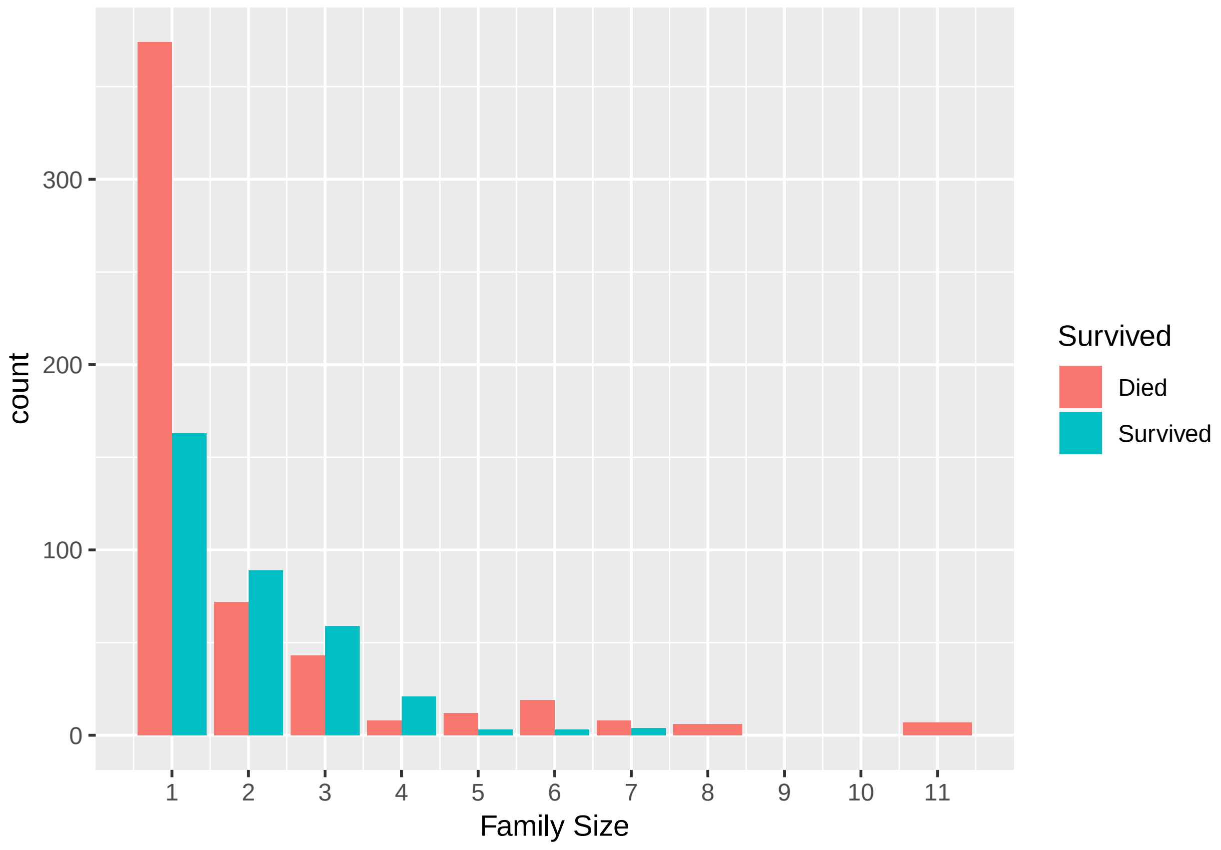
After the data is split we can again use *ggplot* to look at the number in each title, and whether they survived.

```r
# And plot the result
p7 <- ggplot(titanic, aes(x = Title, fill = Survived)) +
  geom_bar(stat='count', position='dodge') +
  labs(x = 'Title') + theme_grey()
p7
```



We can also create and plot the family size. We do this by adding the number of siblings, parents, children and the person themself.

```r
# A next step is to create a family size for each person
titanic$Fsize <- titanic$SibSp + titanic$Parch + 1

# And plot
p8 <- ggplot(titanic, aes(x = Fsize, fill = Survived)) +
  geom_bar(stat='count', position='dodge') +
  scale_x_continuous(breaks=c(1:11)) +
  labs(x = 'Family Size') + theme_grey()
p8
```



We can summarise the data using the package *skimr*:

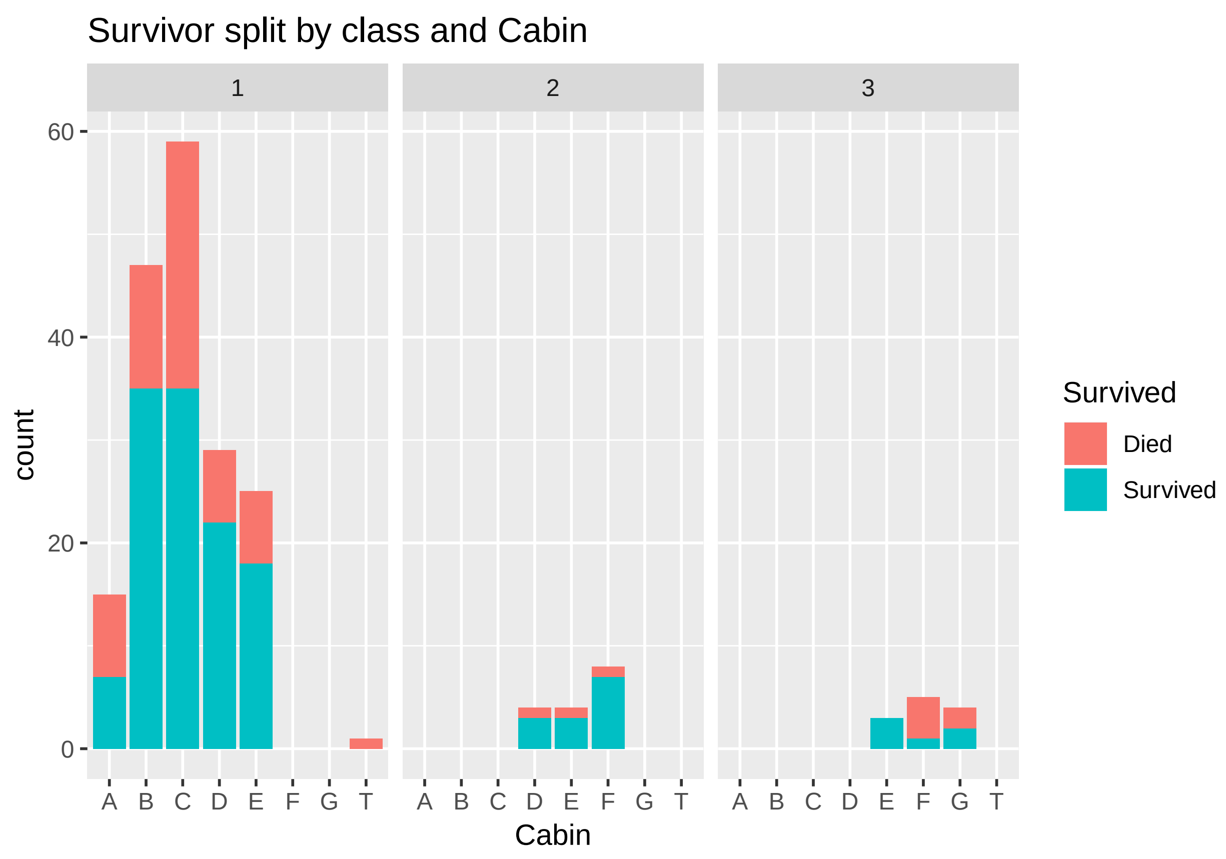
```
# Perhaps if we summarise the data
library(skimr)
```

```
skimmed <- skim(titanic)
skimmed
```

```
## Skim summary statistics
##  n obs: 891
##  n variables: 16
##
## -- Variable type:character -----------------------------------------------------------------
##     variable missing complete   n min max empty n_unique
##        Cabin     687      204 891   1  15     0      147
##         Name       0      891 891  12  82     0      891
##  PassengerId       0      891 891   1   3     0      891
##      Surname       0      891 891   3  20     0      664
##       Ticket       0      891 891   3  18     0      681
##
## -- Variable type:factor --------------------------------------------------------------------
##   variable missing complete   n n_unique
##   Embarked       2      889 891        3
##     Pclass       0      891 891        3
##  PclassSex       0      891 891        6
##        Sex       0      891 891        2
##   Survived       0      891 891        2
##      Title       0      891 891        5
##                               top_counts ordered
##           S: 644, C: 168, Q: 77, NA: 2   FALSE
##          3: 491, 1: 216, 2: 184, NA: 0   FALSE
##  P3M: 347, P3F: 144, P1M: 122, P2M: 108   FALSE
##              mal: 577, fem: 314, NA: 0   FALSE
##              Die: 549, Sur: 342, NA: 0   FALSE
##    Mr: 517, Mis: 185, Mrs: 126, Mas: 40   FALSE
##
## -- Variable type:integer -------------------------------------------------------------------
##  variable missing complete   n mean   sd p0 p25 p50 p75 p100     hist
##     Parch       0      891 891 0.38 0.81  0   0   0   0    6 <U+2587><U+2582><U+2581><U+2581><U+2581
##     SibSp       0      891 891 0.52 1.1   0   0   0   1    8 <U+2587><U+2581><U+2581><U+2581><U+2581
##
## -- Variable type:numeric -------------------------------------------------------------------
##  variable missing complete   n mean    sd   p0   p25   p50 p75   p100
##       Age     177      714 891 29.7 14.53 0.42 20.12 28     38  80
##      Fare       0      891 891 32.2 49.69 0     7.91 14.45  31 512.33
##     Fsize       0      891 891  1.9  1.61 1     1     1      2  11
##      hist
##  <U+2582><U+2583><U+2587><U+2586><U+2583><U+2582><U+2581><U+2581>
##  <U+2587><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581>
##  <U+2587><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581><U+2581>
```

For the cabin variable, we there is rather a lot to imupte but we could change all the missing observations to 'u' and then extract the first letter from the remaining observations.The first letter represents the deck so might be important for survival if lifeboats were all on the same deck. We can then plot the result

```
# Examine Canin variable
titanic$Cabin[is.na(titanic$Cabin)] <- "U"
titanic$Cabin <- substring(titanic$Cabin, 1, 1)
titanic$Cabin <- as.factor(titanic$Cabin)

# We caan generate a plot
p9 <- ggplot(titanic[(!is.na(titanic$Survived)& titanic$Cabin!='U'),],
             aes(x=Cabin, fill=Survived)) +
  geom_bar(stat='count') + theme_grey() + facet_grid(.~Pclass) +
  labs(title="Survivor split by class and Cabin")
p9
```



It seems that cabin is unimportant for predicting survival. There is a 50:50 split for each deck. I will therefore ignore this variable.

Dealing with Missing Data

We can find the missing data like this:

```
# Find NAs
sapply(titanic, function(x) {sum(is.na(x))})
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##           0           0           0           0           0         177
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##           0           0           0           0           0           2
##   PclassSex     Surname       Title       Fsize
##           0           0           0           0
```

For age, I'll code for the missingness before imputation.

```
# Code age missingess
titanic$age_present <- ifelse(is.na(titanic$Age),"No", "Yes")
titanic$age_present <- as.factor(titanic$age_present)
```

Now we can remove the variables we don't need because they are are useless of unusable:

```
# Remove the remaining unhelpful variables
titanic$Name <- NULL
titanic$Ticket <- NULL
```

```r
titanic$Parch <- NULL
titanic$Surname <- NULL
titanic$Cabin <- NULL
titanic$PassengerId <- NULL
```

And change embarked also to a factor. We can then view a summary of the data again using skimmed.

```r
# And change embarked to a factor
titanic$Embarked <- as.factor(titanic$Embarked)
skim_to_wide(titanic)
```

```
## # A tibble: 11 x 16
##    type  variable missing complete n     n_unique top_counts ordered mean
##    <chr> <chr>    <chr>   <chr>    <chr> <chr>    <chr>      <chr>   <chr>
##  1 fact~ age_pre~ 0       891      891   2        Yes: 714,~ FALSE   <NA>
##  2 fact~ Embarked 2       889      891   3        S: 644, C~ FALSE   <NA>
##  3 fact~ Pclass   0       891      891   3        3: 491, 1~ FALSE   <NA>
##  4 fact~ PclassS~ 0       891      891   6        P3M: 347,~ FALSE   <NA>
##  5 fact~ Sex      0       891      891   2        mal: 577,~ FALSE   <NA>
##  6 fact~ Survived 0       891      891   2        Die: 549,~ FALSE   <NA>
##  7 fact~ Title    0       891      891   5        Mr: 517, ~ FALSE   <NA>
##  8 inte~ SibSp    0       891      891   <NA>     <NA>       <NA>    0.52
##  9 nume~ Age      177     714      891   <NA>     <NA>       <NA>    29.7
## 10 nume~ Fare     0       891      891   <NA>     <NA>       <NA>    32.2
## 11 nume~ Fsize    0       891      891   <NA>     <NA>       <NA>    " 1.~
## # ... with 7 more variables: sd <chr>, p0 <chr>, p25 <chr>, p50 <chr>,
## #   p75 <chr>, p100 <chr>, hist <chr>
```

For the remaining data I will impute it using the *mice* package.

```r
# Now lets attempt to impute the age, fare and embarked
library(mice)
tempData <- mice(titanic,m=5,maxit=50,meth='pmm',seed=500)
```

```
## Warning: Number of logged events: 857
```

```r
titanic <- complete(tempData,1)
skim_to_wide(titanic)
```

```
## # A tibble: 11 x 16
##    type  variable missing complete n     n_unique top_counts ordered mean
##    <chr> <chr>    <chr>   <chr>    <chr> <chr>    <chr>      <chr>   <chr>
##  1 fact~ age_pre~ 0       891      891   2        Yes: 714,~ FALSE   <NA>
##  2 fact~ Embarked 0       891      891   3        S: 645, C~ FALSE   <NA>
##  3 fact~ Pclass   0       891      891   3        3: 491, 1~ FALSE   <NA>
##  4 fact~ PclassS~ 0       891      891   6        P3M: 347,~ FALSE   <NA>
##  5 fact~ Sex      0       891      891   2        mal: 577,~ FALSE   <NA>
##  6 fact~ Survived 0       891      891   2        Die: 549,~ FALSE   <NA>
##  7 fact~ Title    0       891      891   5        Mr: 517, ~ FALSE   <NA>
##  8 inte~ SibSp    0       891      891   <NA>     <NA>       <NA>    0.52
##  9 nume~ Age      0       891      891   <NA>     <NA>       <NA>    29.89
## 10 nume~ Fare     0       891      891   <NA>     <NA>       <NA>    "32.~
## 11 nume~ Fsize    0       891      891   <NA>     <NA>       <NA>    " 1.~
## # ... with 7 more variables: sd <chr>, p0 <chr>, p25 <chr>, p50 <chr>,
## #   p75 <chr>, p100 <chr>, hist <chr>
```

```r
rm(tempData)
```

Model Setup

I am now ready to run the models. I will start by defining a formula for the models:

Add the formula

```r
form <- as.formula(survived ~ pclass + age + fare + Fsize + sibsp + Title +
                     sex + PclassSex + embarked +
                     age_present + cabin_present)
```

The next step is to define the controls for each model. From p1 it's clear there is an imbalance in the data. There are several ways to deal with this. The models will be built using the Caret package so I will focus on Caret's built in methods. Briefly, they are downsampling the major class, upsampling the minor class, or using a hybrid of the 2 (SMOTE and ROSE). I will use the normal data, as well as 'down' and 'smote' methods for dealing with the imbalance.

I will define train control with normal, down and smote, and assess which method is best. These algorithms are implemented within the repeated cross validation. Repeated cross validation avoids the need for manually splitting a training and test set. It divides the data into k folds, holding 1 set for testing and the others for training. The resampling using "up" or "smote" is done to the training data only. This is repeated and the result is an average. This avoids over fitting.

I will repeat the 5 fold cross validation 5 times. This is standard. I will focus on the ROC curve with the two class summary.

```r
# And train control with "normal", "down" and "smote"
library(caret)
library(DMwR)
ctrl <- trainControl(method= "repeatedcv",
                     number = 5,
                     repeats= 5,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)
ctrl_down <- trainControl(method= "repeatedcv",
                          number = 5,
                          repeats= 5,
                          summaryFunction = twoClassSummary,
                          classProbs = TRUE,
                          sampling = "down")
ctrl_smote <- trainControl(method= "repeatedcv",
                           number = 5,
                           repeats= 5,
                           summaryFunction = twoClassSummary,
                           classProbs = TRUE,
                           sampling = "smote")
```

I will configure multicore to allow extra cores to be used. This is possible because each of the cross validated models is analysed separately.

```r
# Configure multicore
library(doParallel)
```

```r
cl <- makeCluster(detectCores()-1)
registerDoParallel(cl)
```

Now I can define a function which will train the models. The maxit isn't relevant for every model but will work for all of them. Each model has it's own tuning parameters which I will define individually. The data is preprocessed using range, which puts all the variables between 0 and 1. This makes the variables comparable with the dummy variables, which will be generated automatically by the models for every variable which is a factor.

```r
# Define a function for training the models
train_models <- function(method, ctrl, tuning) {
  model.train <- train(form,
                       data = titanic,
                       method = method,
                       trControl = ctrl,
                       tuneGrid = tuning,
                       maxit = 1000000,
                       preProcess = c('range'))
  return(model.train)
}
```

Training the Models

Each model has its own tuning defined by me. The algorithm selects the best model based on optimal parameters. I will train 3 models for each algorithm, and evaluate them together at the end. Firstly, the extreme gradient boosting algorithm:

```r
# 1. Extreme gradient boosting model
set.seed(2018)
tune.grid.xgb <- expand.grid(eta = c(0.05, 0.075, 0.1),
                             nrounds = c(50, 75, 100),
                             max_depth = 6:8,
                             min_child_weight = c(2.0, 2.25, 2.5),
                             colsample_bytree = c(0.3, 0.4, 0.5),
                             gamma = 0,
                             subsample = 1)
# Normal non-sampled
model_xgb <- train_models("xgbTree", ctrl, tune.grid.xgb)
# Down Sampled
model_xgb_down <- train_models("xgbTree", ctrl_down, tune.grid.xgb)
# Smote Sampled
model_xgb_smote <- train_models("xgbTree", ctrl_smote, tune.grid.xgb)
```

The random forest:

```r
# 2. Random Forest
set.seed(2018)
tune.grid.rf <- expand.grid(.mtry = sqrt(ncol(titanic)))
# Normal non-sampled
model_rf <- train_models("rf", ctrl, tune.grid.rf)
# Down Sampled
model_rf_down <- train_models("rf", ctrl_down, tune.grid.rf)
# Smote Sampled
model_rf_smote <- train_models("rf", ctrl_smote, tune.grid.rf)
```

A neural network:

```r
# 3. Model Averaged Neural Network
set.seed(2018)
library(nnet)
tune.grid.nn <- expand.grid(size=c(10), decay=c(0.1))
# Normal non-sampled
model_net <- train_models("nnet", ctrl, tune.grid.nn)
# Down Sampled
model_net_down <- train_models("nnet", ctrl_down, tune.grid.nn)
# Smote Sampled
model_net_smote <- train_models("nnet", ctrl_smote, tune.grid.nn)
```

Logistic Regression with elastic net regularisation

```r
# 4. Logistic Regression
set.seed(2018)
lamda.grid <- 10^seq(2, -2, length = 100)
alpha.grid <- seq(0,1, length = 10)
srchGrd <- expand.grid(.alpha = alpha.grid,
                       .lambda = lamda.grid)
# Normal non-sampled
model_glm <- train_models("glmnet", ctrl, srchGrd)
# Down Sampled
model_glm_down <- train_models("glmnet", ctrl_down, srchGrd)
# Smote Sampled
model_glm_smote <- train_models("glmnet", ctrl_smote, srchGrd)
```

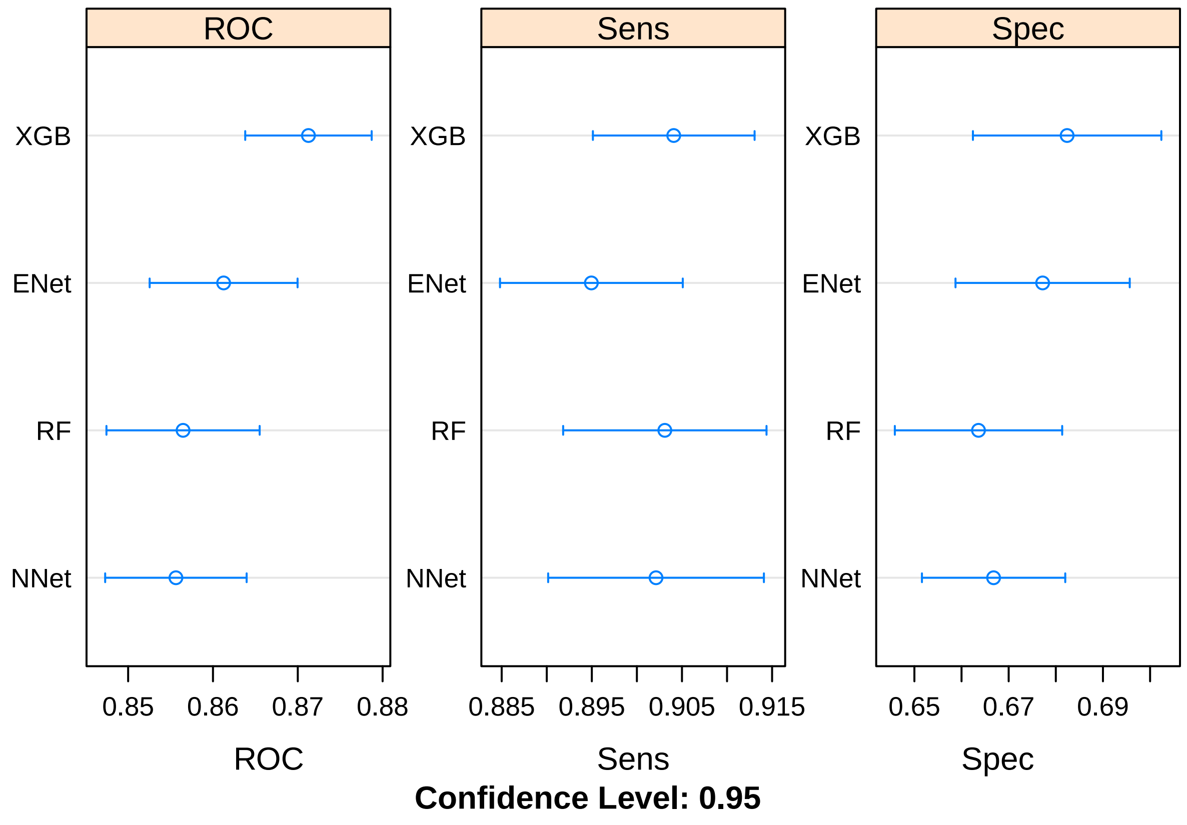
Evaluating Model Performance

The next step is to evaluate how the models perform. I will group the models together and then use a simple dot plot with 95% confidence intervals to examine the result.

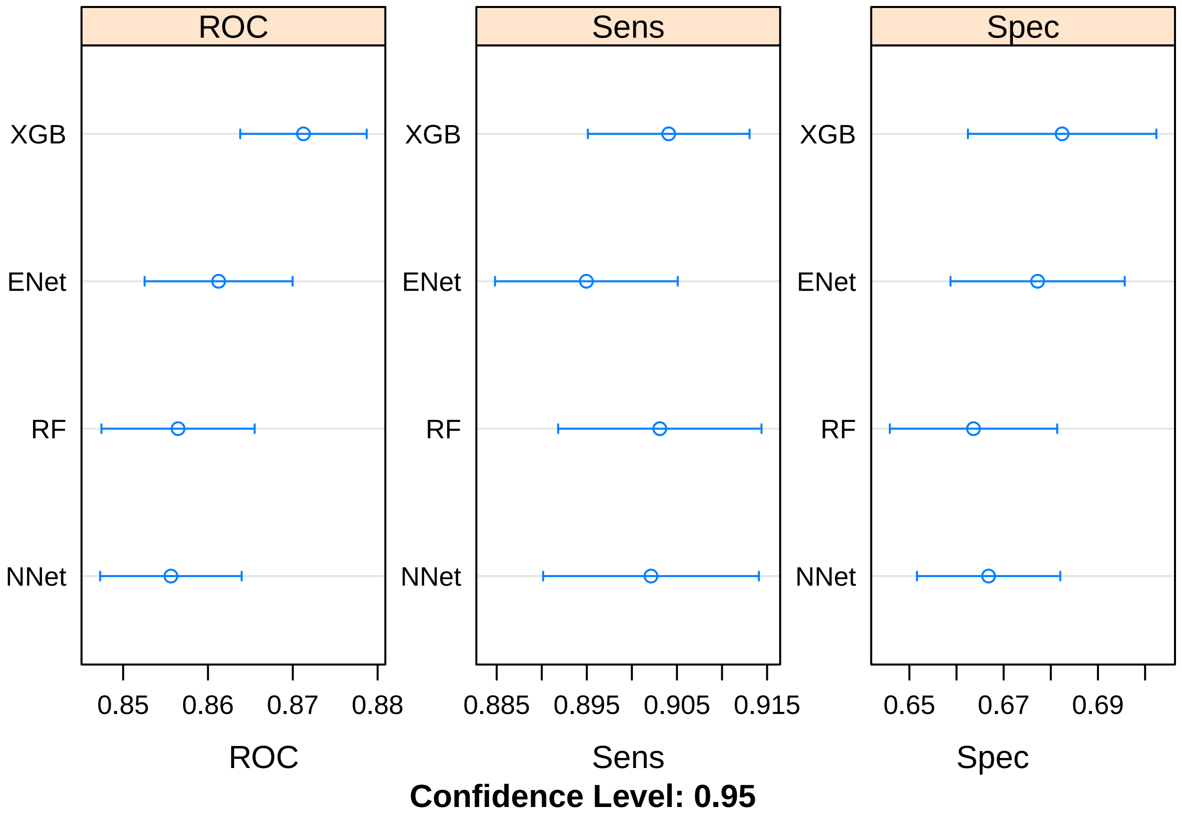
```r
# Group the models
models_compare <- resamples(list(RF = model_rf,
                                 XGB = model_xgb,
                                 ENet = model_glm,
                                 NNet = model_net))
models_compare_down <- resamples(list(RF = model_rf_down,
                                      XGB = model_xgb_down,
                                      ENet = model_glm_down,
                                      NNet = model_net_down))
models_compare_smote <- resamples(list(RF = model_rf_smote,
                                       XGB = model_xgb_smote,
                                       ENet = model_glm_smote,
                                       NNet = model_net_smote))
# Create the scales for the plotting
scales <- list(x=list(relation="free"), y=list(relation="free"))
```

Next, I will display a dotplot for each set of models:

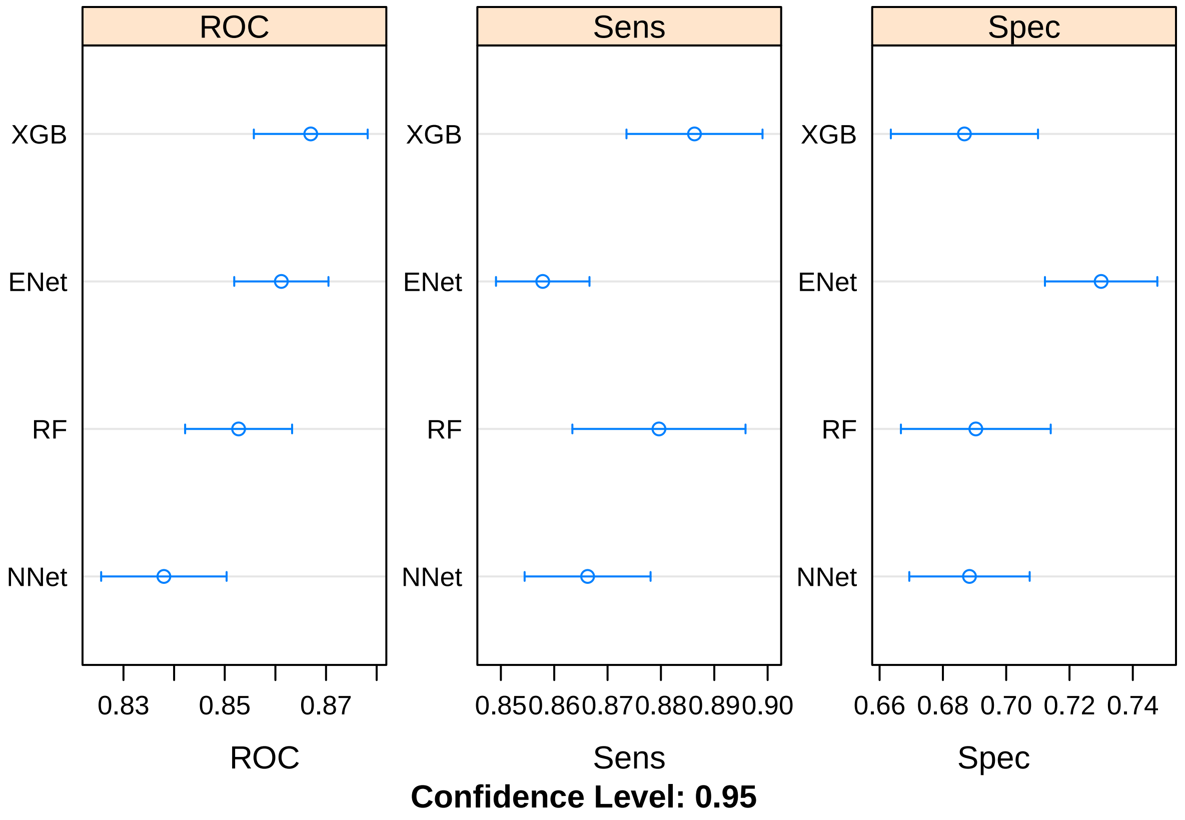
```r
# Dot plot each one so we can compare the models smote, down and normal sampling
dotplot <- dotplot(models_compare, scales = scales)
dotplot_down <- dotplot(models_compare_down, scales=scales)
dotplot_smote <- dotplot(models_compare_smote, scales=scales)
dotplot
```



```
dotplot_down
```

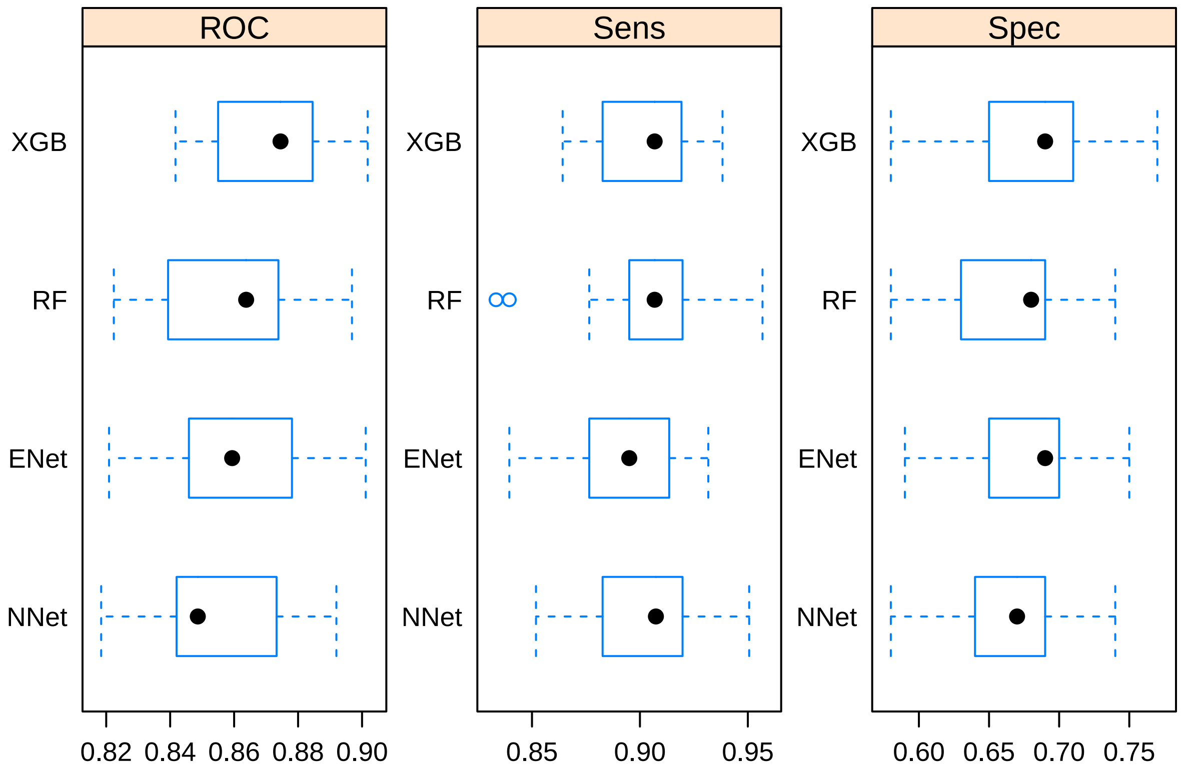


```
dotplot_smote
```

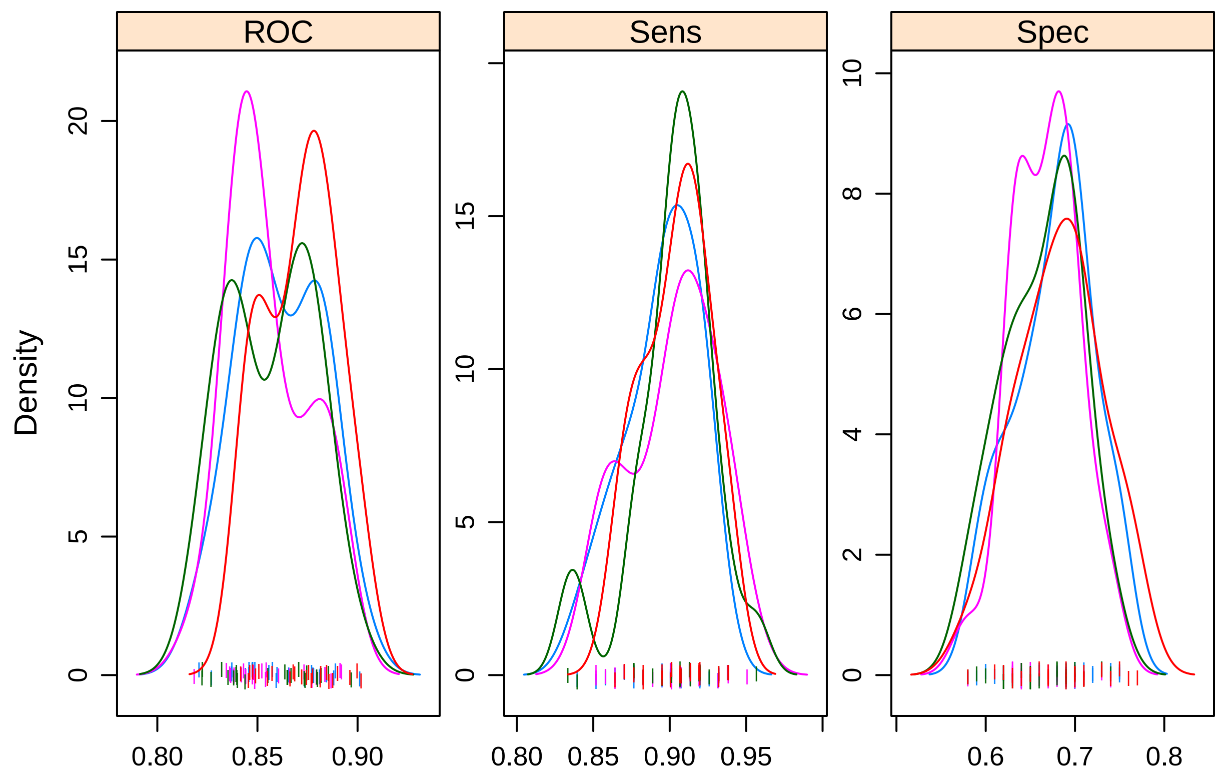


It seems that the resampling doesn't make much difference. This is unsuprising given that the resampling is for very large imbalances in the data. I will therefore do the following analysis only on the unsampled models. Firstly, I look at 5 plots of the models:

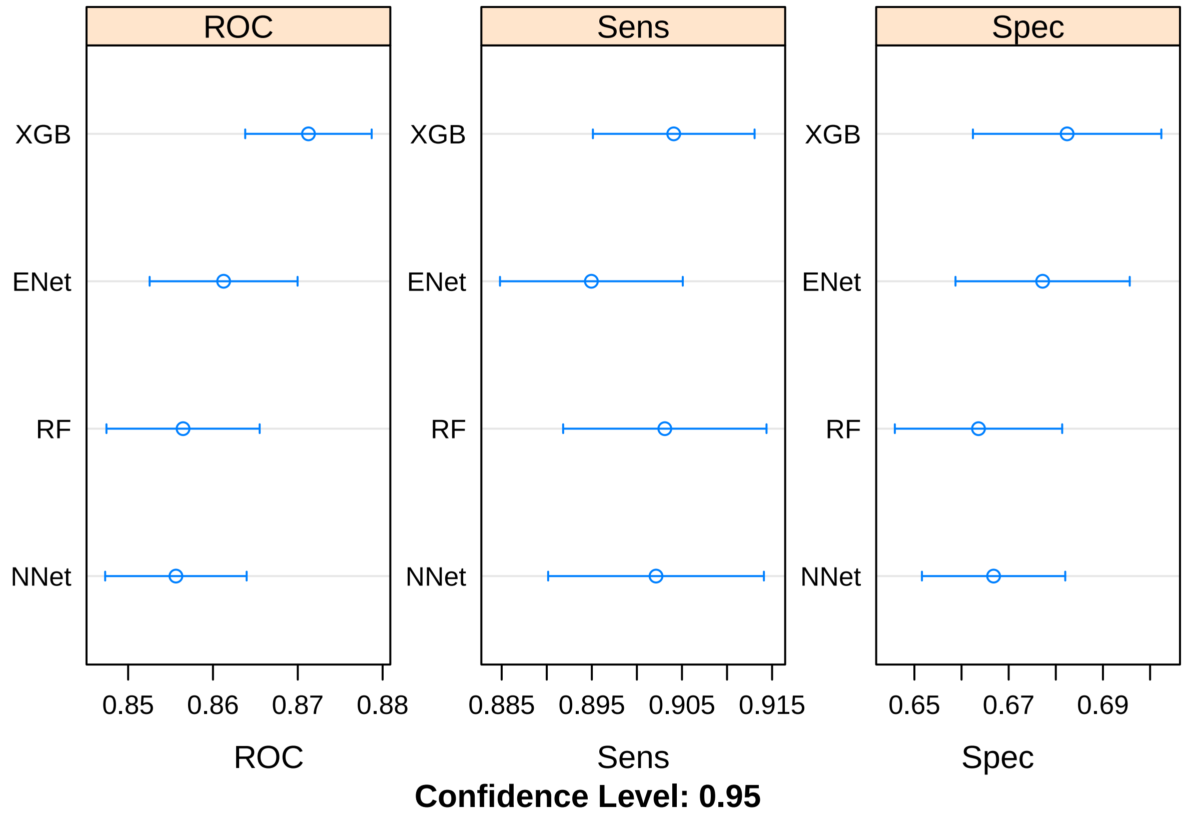
```r
# Box plot
boxplot <- bwplot(models_compare, scales=scales)
boxplot
```



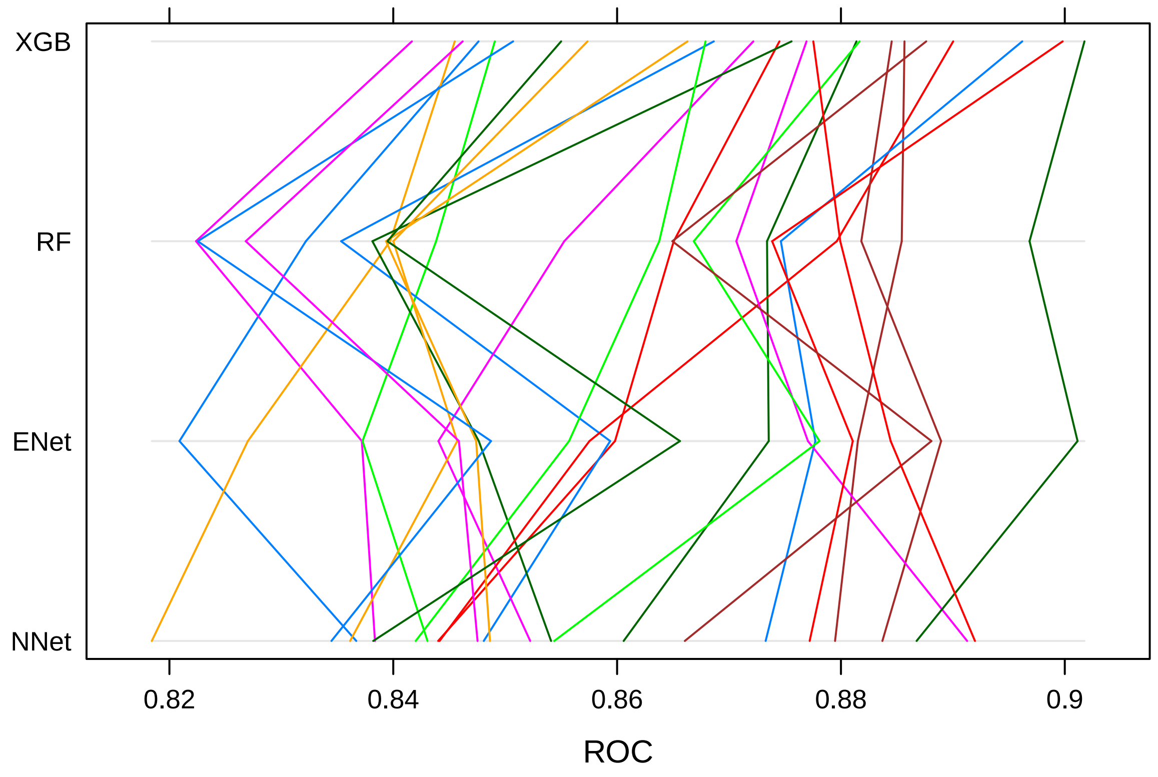
```
# Density Plot
denplot <- densityplot(models_compare, scales=scales, pch = "|")
denplot
```



```
# Dotplot
dotplot <- dotplot(models_compare, scales=scales)
dotplot
```

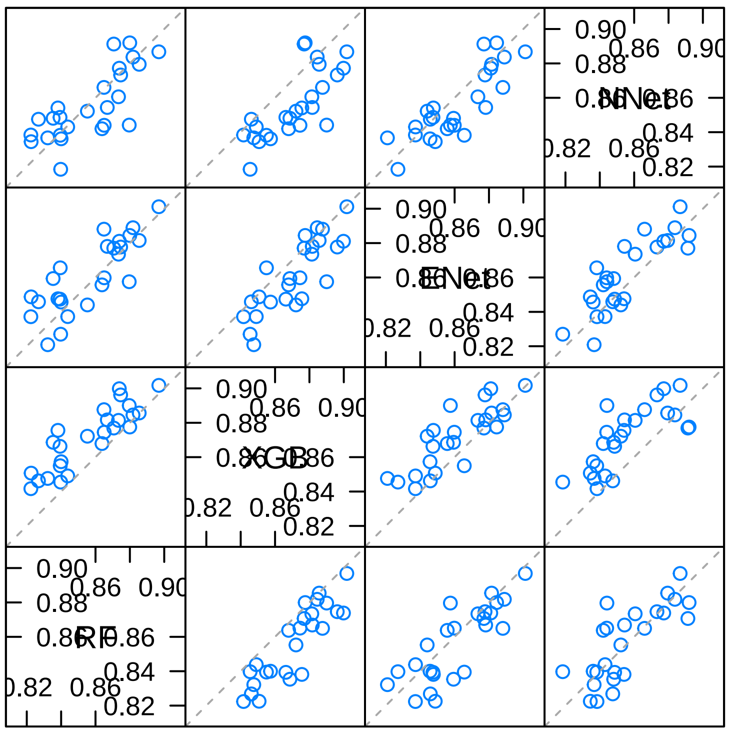


```
# Parallel plot
paraplot <- parallelplot(models_compare)
paraplot
```



```r
# Scatterplot
scatter <- splom(models_compare)
scatter
```

ROC



Scatter Plot Matrix

The models are achieving between 85 and 87% accuracy, which is reasonably impressive given we would expect the data to feature some randomness. We would expect many of the people who survived to have ended up on the lifeboat by chance. The scatterplot shows that ensembling the models may not be suitable because there is a high correlation across each of them. The parallel plot is colourful but not particularly helpful, only for showing that the xgb is consistently the best performiong model.

Feature Importance

Finally, I will do some feature importance, which is to look which factors were most important in the models using the gini impurity. More details can be found here. Firstly, I will define a function which takes each model and its importance, then plots them.

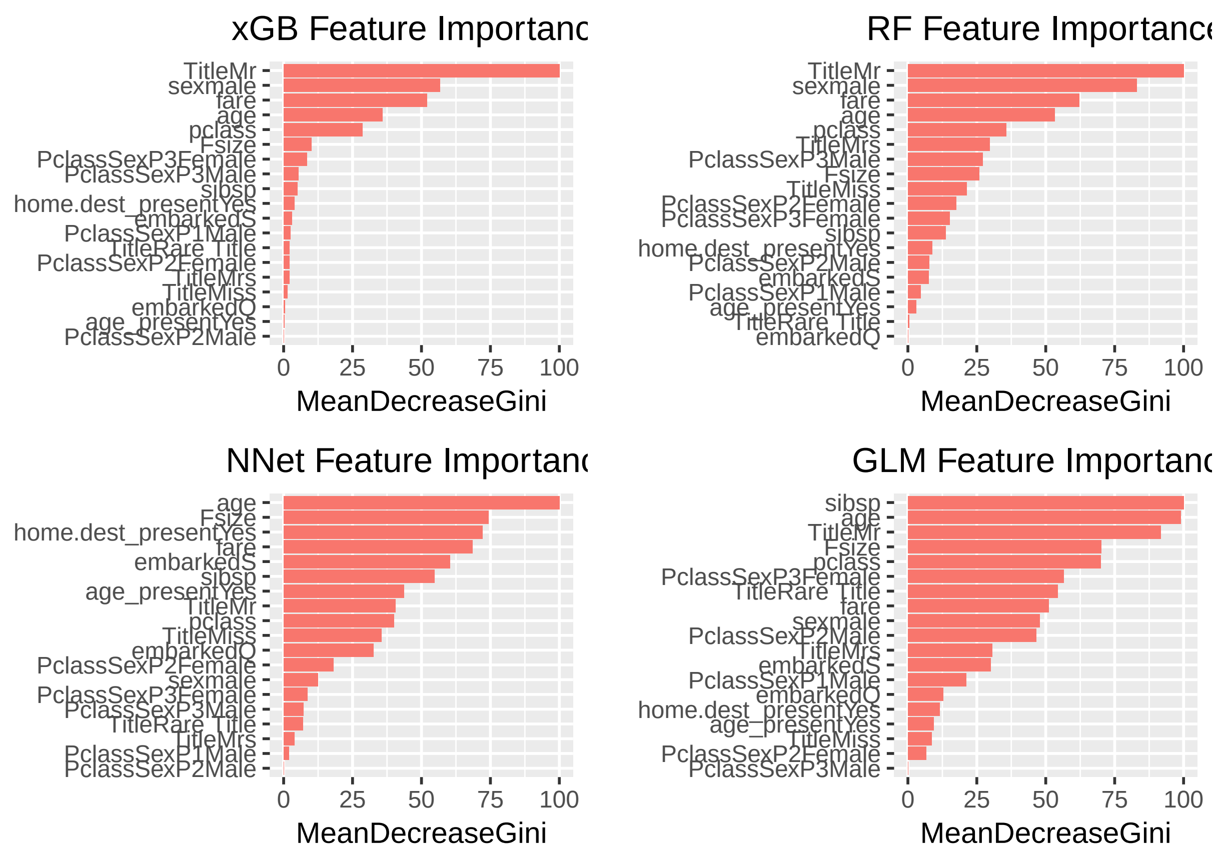
```r
# Define Function
plot_imp <- function(model, title) {
  FI_mod <- varImp(model) # Takes variable Importance
  FI_dat <- FI_mod$importance # Puts into data frame
  FI_gini <- data.frame(Variables = row.names(FI_dat),
                        MeanDecreaseGini = FI_dat$Overall) # Renames
  # Plots:
  FI_plot <- ggplot(FI_gini, aes(x=reorder(Variables, MeanDecreaseGini),
                              y=MeanDecreaseGini, fill= "green")) +
    geom_bar(stat='identity') + coord_flip() +
    theme(legend.position="none") + labs(x="") +
    ggtitle(title) + theme(plot.title = element_text(hjust = 0.5))
return(FI_plot)
}
```

Now we run the function for each of the models:

```r
# Now to plot the feature importance of the best group of models
xgb_imp_plot <- plot_imp(model_xgb, "xGB Feature Importance")
glm_imp_plot <- plot_imp(model_glm, "GLM Feature Importance")
rf_imp_plot <- plot_imp(model_rf, "RF Feature Importance")
net_imp_plot <- plot_imp(model_net, "NNet Feature Importance")
```

Now plot the importance:

```r
# It seems 4 of the models are easily comparable:
library(gridExtra)
plot_imp <- grid.arrange(xgb_imp_plot, rf_imp_plot,
                         net_imp_plot, glm_imp_plot, ncol=2)
```



```r
plot_imp
```

```
## TableGrob (2 x 2) "arrange": 4 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (2-2,1-1) arrange gtable[layout]
## 4 4 (2-2,2-2) arrange gtable[layout]
```

Now it could be helpful to assign the variables a rank from 1 to 20 and then see which is important across the model. To do this, I start by defining a function which takes each model, gets its feature importance and ranks them leaving just the variable and rank in each data frame.

```r
rank_features <- function(model) {
  FI_mod <- varImp(model)
  FI_df <- FI_mod$importance
  FI_df <- data.frame(Variables = row.names(FI_df),
                      overall = FI_df$Overall)
  FI_df <- arrange(FI_df, desc(overall)) %>%
    mutate(rank = 1:nrow(FI_df))
  FI_df$overall <- NULL

  return(FI_df)
}
```

Now I will run the function for each model, as well as changing the rank variable name for clarity:

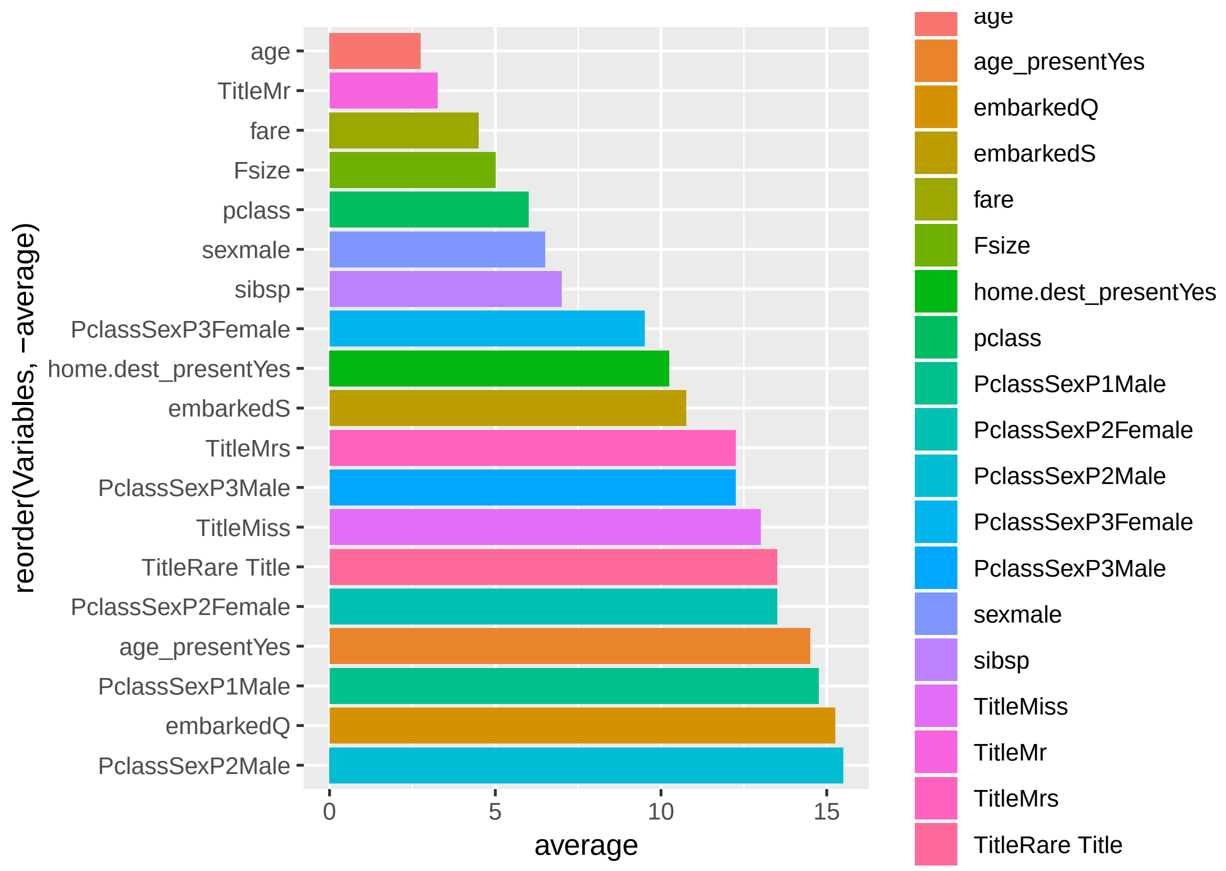
```r
rank_xgb <- rank_features(model_xgb)
rank_xgb <- rename(rank_xgb, xgb = rank)

rank_glm <- rank_features(model_glm)
rank_glm <- rename(rank_glm, glm = rank)

rank_rf <- rank_features(model_rf)
rank_rf <- rename(rank_rf, rf = rank)

rank_net <- rank_features(model_net)
rank_net <- rename(rank_net, net = rank)
```

Next, I will join the data sets using *reduce*, and add variables for average and then rank the average, using *dplyr* from the tidyverse.

```r
# Now, we can join these data sets
list_dfs <- list(rank_xgb, rank_glm, rank_net, rank_rf)
ranks <- Reduce(function(...) merge(..., all=T), list_dfs)

# And create an average for each variable
ranks <- mutate(ranks, average = (xgb + glm + rf + net)/4) %>%
  arrange(average) %>%
  mutate(rank.average = 1:nrow(ranks))
```

Finally, I will plot the average rank. There are clearly some predictors which do very well.

```r
# We can plot the average rank
plot_rank <- ggplot(ranks, aes(x = reorder(Variables, -average), y = average,
                            fill = Variables)) + geom_col() +
  coord_flip()
plot_rank
```



Age is the most important. Younger people were more likely to survive. Title3 "Mister" is important, seemingly they were more liekly to die. Family size is important, fare paid (related to pclass), sex is important, as is the dummy for whether the class was present. A logical next step would be to re run the models with a succint set of predictors.

Test Set

Finally, I want to run the same code on the test set. It is perhaps easier to combine the 2 data sets at the start, but I think this is annoying for plotting. I'm just going to run the code for the test set:

```r
# Predict
library(readr)
test <- read_csv("test.csv", col_types =
                cols(Embarked = col_factor(levels = c("S", "C", "Q")),
                PassengerId = col_character(),
                Pclass = col_factor(levels = c("1", "2", "3")),
                Sex = col_factor(levels = c("male","female"))))

test$Sex <- as.factor(test$Sex)
test$PclassSex[test$Pclass== 1  & test$Sex=='male'] <- 'P1Male'
test$PclassSex[test$Pclass== 2  & test$Sex=='male'] <- 'P2Male'
test$PclassSex[test$Pclass== 3  & test$Sex=='male'] <- 'P3Male'
test$PclassSex[test$Pclass== 1  & test$Sex=='female'] <- 'P1Female'
test$PclassSex[test$Pclass== 2  & test$Sex=='female'] <- 'P2Female'
test$PclassSex[test$Pclass== 3  & test$Sex=='female'] <- 'P3Female'
test$PclassSex <- as.factor(test$PclassSex)
```

```r
test$Surname <- sapply(test$Name, function(x) {strsplit(x, split='[,.]')[[1]][1]})
test$Surname <- sapply(test$Surname, function(x) {strsplit(x, split='[-]')[[1]][1]})
test$Title <- sapply(test$Name, function(x) {strsplit(x, split='[,.]')[[1]][2]})
test$Title <- sub(' ', '', test$Title) #removing spaces before title
test$Title[test$Title %in% c("Mlle", "Ms")] <- "Miss"
test$Title[test$Title== "Mme"] <- "Mrs"
test$Title[!(test$Title %in% c('Master', 'Miss', 'Mr', 'Mrs'))] <- "Rare Title"
test$Title <- as.factor(test$Title)
test$Fsize <- test$SibSp + test$Parch + 1
test$Cabin[is.na(titanic$Cabin)] <- "U"
test$Cabin <- substring(test$Cabin, 1, 1)
test$Cabin <- as.factor(test$Cabin)
test$age_present <- ifelse(is.na(test$Age),"No", "Yes")
test$age_present <- as.factor(test$age_present)
test$Name <- NULL
test$Ticket <- NULL
test$Parch <- NULL
test$Surname <- NULL
test$Cabin <- NULL
tempData <- mice(test,m=5,maxit=50,meth='pmm',seed=500)
test <- complete(tempData,1)
```

Finally, use all the 12 models and generate an ensemble predicion based on whichever is highest.

```r
# Make Predictions
prediction_frame <- data.frame(matrix(nrow=418, ncol=0))
prediction_frame <- as.data.frame(predict(model_xgb, test))
prediction_frame$xgb_down <- predict(model_xgb_down, test)
prediction_frame$xgb_smote <- predict(model_xgb_smote, test)

prediction_frame$rf <- predict(model_rf, test)
prediction_frame$rf_down <- predict(model_rf_down, test)
prediction_frame$rf_smote <- predict(model_rf_smote, test)

prediction_frame$net <- predict(model_net, test)
prediction_frame$net_down <- predict(model_net_down, test)
prediction_frame$net_smote <- predict(model_net_smote, test)

prediction_frame$glm <- predict(model_glm, test)
prediction_frame$glm_down <- predict(model_glm_down, test)
prediction_frame$glm_smote <- predict(model_glm_smote, test)

prediction_frame$final <-
  apply(prediction_frame,1,function(x) names(which.max(table(x))))


final_prediction <- prediction_frame %>% select(final)
passenger_ids <- test %>% select(PassengerId)
final_prediction <- cbind(passenger_ids, final_prediction)

final_prediction$final <- ifelse(final_prediction$final == "Survived",1,0)
```

Finally, save the data as a csv:

```r
write.csv(final_prediction, "final_prediction.csv")
```