Data preparation and feature engineering

Titanic data set

For this Lab, we will use the Titanic data set, available from Kaggle.com.

Load the data (training and test sets):

```
# load Titanic train ("data/train.csv") and test sets ("data/test.csv")
titanic.train <- read.csv("data/train.csv", stringsAsFactors = F)
titanic.test <- read.csv("data/test.csv", stringsAsFactors = F)
```

Let's start by examining the structure of the data sets.

Note: description of all the variables is available at the Kaggle website.

```
# print the structure of the train set
str(titanic.train)
```

```
## 'data.frame':    891 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        : chr  "male" "female" "female" "female" ...
##  $ 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" ...
```

```
# print the structure of the test set
str(titanic.test)
```

```
## 'data.frame':    418 obs. of  11 variables:
##  $ PassengerId: int  892 893 894 895 896 897 898 899 900 901 ...
##  $ Pclass     : int  3 3 2 3 3 3 3 2 3 3 ...
##  $ Name       : chr  "Kelly, Mr. James" "Wilkes, Mrs. James (Ellen Needs)" "Myles, Mr. Thomas Francis
##  $ Sex        : chr  "male" "female" "male" "male" ...
##  $ Age        : num  34.5 47 62 27 22 14 30 26 18 21 ...
##  $ SibSp      : int  0 1 0 0 1 0 0 1 0 2 ...
##  $ Parch      : int  0 0 0 0 1 0 0 1 0 0 ...
##  $ Ticket     : chr  "330911" "363272" "240276" "315154" ...
##  $ Fare       : num  7.83 7 9.69 8.66 12.29 ...
##  $ Cabin      : chr  "" "" "" "" ...
##  $ Embarked   : chr  "Q" "S" "Q" "S" ...
```

The structure of the training and test sets is almost exactly the same (as expected). In fact, the only difference is the *Survived* column that is present in the training, but absent in the test set - it is the response (outcome) variable.

Detecting missing values

Let's start by checking if the data is complete, that is if there are some missing values.

One way to do that is through the *summary* f. which will let us know if a variable has NA values.

```
# print the summary of the train set
summary(titanic.train)
```

```
##   PassengerId       Survived          Pclass          Name          
##  Min.   :  1.0   Min.   :0.0000   Min.   :1.000   Length:891        
##  1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000   Class :character  
##  Median :446.0   Median :0.0000   Median :3.000   Mode  :character  
##  Mean   :446.0   Mean   :0.3838   Mean   :2.309                     
##  3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000                     
##  Max.   :891.0   Max.   :1.0000   Max.   :3.000                     
##                                                                     
##      Sex                 Age            SibSp           Parch       
##  Length:891         Min.   : 0.42   Min.   :0.000   Min.   :0.0000  
##  Class :character   1st Qu.:20.12   1st Qu.:0.000   1st Qu.:0.0000  
##  Mode  :character   Median :28.00   Median :0.000   Median :0.0000  
##                     Mean   :29.70   Mean   :0.523   Mean   :0.3816  
##                     3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000  
##                     Max.   :80.00   Max.   :8.000   Max.   :6.0000  
##                     NA's   :177                                     
##     Ticket               Fare           Cabin             Embarked        
##  Length:891         Min.   :  0.00   Length:891         Length:891        
##  Class :character   1st Qu.:  7.91   Class :character   Class :character  
##  Mode  :character   Median : 14.45   Mode  :character   Mode  :character  
##                     Mean   : 32.20                                        
##                     3rd Qu.: 31.00                                        
##                     Max.   :512.33                                        
## 
```

It seems that in the training set only *Age* has missing values, and quite a number of them (177).

```
# print the summary of the test set
summary(titanic.test)
```

```
##   PassengerId         Pclass          Name               Sex           
##  Min.   : 892.0   Min.   :1.000   Length:418         Length:418        
##  1st Qu.: 996.2   1st Qu.:1.000   Class :character   Class :character  
##  Median :1100.5   Median :3.000   Mode  :character   Mode  :character  
##  Mean   :1100.5   Mean   :2.266                                        
##  3rd Qu.:1204.8   3rd Qu.:3.000                                        
##  Max.   :1309.0   Max.   :3.000                                        
##                                                                        
##       Age            SibSp            Parch            Ticket         
```

```
##  Min.   : 0.17   Min.   :0.0000   Min.   :0.0000   Length:418
##  1st Qu.:21.00   1st Qu.:0.0000   1st Qu.:0.0000   Class :character
##  Median :27.00   Median :0.0000   Median :0.0000   Mode  :character
##  Mean   :30.27   Mean   :0.4474   Mean   :0.3923
##  3rd Qu.:39.00   3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   :76.00   Max.   :8.0000   Max.   :9.0000
##  NA's   :86
##       Fare            Cabin             Embarked
##  Min.   :  0.000   Length:418         Length:418
##  1st Qu.:  7.896   Class :character   Class :character
##  Median : 14.454   Mode  :character   Mode  :character
##  Mean   : 35.627
##  3rd Qu.: 31.500
##  Max.   :512.329
##  NA's   :1
```

In the test set, in addition to the 86 NAs for *Age*, there is also one missing value for the *Fare* variable.

So, based on the NA values, it seems that only *Age* variable has a serious issue with missing values.

However, if you take a closer look at the output of the *str* function, you'll notice that for some observations (passengers) the value for *Cabin* seems to be missing, that is, *Cabin* value is equal to an empty string ("“). Let's inspect this more closely by checking how many”" values we have for the *Cabin* variable in both datasets.

```
# number of observations with empty Cabin variable
sum(titanic.train$Cabin=="")
```

```
## [1] 687
```

```
sum(titanic.test$Cabin=="")
```

```
## [1] 327
```

So, for 687 passengers in the training set and 327 passengers in the test, we have "" as the *Cabin* value. Should we consider these as missing values?

Recall that on Titanic, there were three classes of passengers, and only those from the 1st class were offered a cabin. So, some of the empty string values we have observed are due to the fact that passengers were from the 2nd or the 3rd class, meaning that they really didn't have a cabin. In those cases, an empty string is not a missing value, but “not applicable” value.

However, passengers from the 1st class should have had a cabin. So, an empty string for the *Cabin* value of a 1st class passenger is a ‘real' missing value. Let's check how many such values we have in the training set.

```
# get indices of observations with no Cabin value from the first class, in the train set
train.class1.no.cabin <- which(titanic.train$Pclass==1 & titanic.train$Cabin=="")
length(train.class1.no.cabin)
```

```
## [1] 40
```

Also, on the test set:

```r
# get indices of observations with no Cabin value from the first class, in the test set
test.class1.no.cabin <- which(titanic.test$Pclass==1 & titanic.test$Cabin=="")
length(test.class1.no.cabin)
```

```
## [1] 27
```

So, for 40 1st class passengers in the training set and 27 1st class passengers in the test set, the *Cabin* value is missing. To make this explicit, let's replace the missing *Cabin* values for 1st class passengers with NAs.

```r
# set the Cabin value for identified passangers to NA in the train and test sets
titanic.train$Cabin[train.class1.no.cabin] <- NA
titanic.test$Cabin[test.class1.no.cabin] <- NA
```

We can check the results of this transformation:

```r
# print the number of missing Cabin values in the train and test sets
sum(is.na(titanic.train$Cabin))
```

```
## [1] 40
```

```r
sum(is.na(titanic.test$Cabin))
```

```
## [1] 27
```

Note that we have discovered missing values of the *Cabin* variable by spotting a few empty strings in the output of the *str* function. However, if those values were not amongst the first couple of values listed by *str*, they would have passed unnoticed. So, let's check other string variables for missing values 'hidden' as empty strings.

```r
# check for the presence of empty strings in character variables in the train set
apply(X = titanic.train[,c("Name","Sex","Ticket","Embarked")],
      MARGIN = 2,
      FUN = function(x) sum(trimws(x)==""))
```

```
##     Name      Sex   Ticket Embarked
##        0        0        0        2
```

In the training set, only for the Embarked variable, we have 2 missing values.

```r
# check for the presence of empty strings in character variables in the test set
apply(X = titanic.test[,c("Name","Sex","Ticket","Embarked")],
      MARGIN = 2,
      FUN = function(x) sum(trimws(x)==""))
```

```
##     Name      Sex   Ticket Embarked
##        0        0        0        0
```

In the test set, none of the examined variables has missing values.

We'll set the two missing values of *Embarked* to NA, as we did with the Cabin.

```
# set the empty Embarked values to NA in the train set
titanic.train$Embarked[titanic.train$Embarked==""] <- NA
```

We have now examined all the variables for the missing values. Before proceeding with 'fixing' the missing values, let's see how we can make use of visualizations to more easily spot missing values.

An easy way to get a high-level view on the data completeness is to visualize the data using some functions from the **Amelia** R package.

```
#install.packages('Amelia')
# load Amelia library
library(Amelia)
```

We will use the *missmap* function to plot the missing data from the training and test sets.

```
# set the display area to show two plots in the same row
par(mfrow=c(1,2))

# use the missmap f. to visualise the missing data in the train set
missmap(obj = titanic.train, main = "Training set", legend = FALSE)

# use the missmap f. to visualise the missing data in the test set
missmap(obj = titanic.test, main = "Test set", legend = FALSE)
```

```r
# revert the plotting area to the default (one plot per row)
par(mfrow=c(1,1))
```

Note: the detection of missing values in the *missmap* function is based on the NA values; so, if we hadn't transformed empty strings (in *Cabin* and *Embarked* columns) into NAs, they wouldn't be visualized as missing.

Handling missing values

Let's now deal with missing values. We'll start with those cases that are easier to deal with, that is, variables where we have just a few missing values.

Categorical variables with a small number of missing values

In our datasets, *Embarked* variable is a categorical (factor) variable.

```r
# get the number of unique values for the Embarked variable in both sets
unique(titanic.train$Embarked)
```

```
## [1] "S" "C" "Q" NA
```

```r
unique(titanic.test$Embarked)
```

```
## [1] "Q" "S" "C"
```

So, as we see, *Embarked* is essentially a nominal (categorical) variable with 3 possible values ('S', 'C', and 'Q'). And, we have seen that it has 2 missing values (in the train set).

In a situation like this, the missing values are replaced by the 'majority class', that is, the most dominant value.

```r
# create the contingency table for the values of the Embarked variable
xtabs(~Embarked, data = titanic.train)
```

```
## Embarked
##   C   Q   S
## 168  77 644
```

So, "S" is the dominant value, and it will be used as a replacement for NAs.

```r
# replace all NA values for the Embarked variable with 'S' in the train set
titanic.train$Embarked[is.na(titanic.train$Embarked)] <- 'S'

# print the contingency table for the values of the Embarked variable
xtabs(~Embarked, data = titanic.train)
```

```
## Embarked
##   C   Q   S
## 168  77 646
```

Let's also make *Embarked* a 'true' categorical variable by transforming it into a factor variable.

```
# transform the Embarked variable into a factor in both sets
titanic.train$Embarked <- as.factor(titanic.train$Embarked)
titanic.test$Embarked <- as.factor(titanic.test$Embarked)
```

Numerical variables with a small number of missing values

In our data set, *Fare* variable belongs to this category - it is a numerical variable with 1 missing value (in the test set).

A typical way to deal with missing values in situations like this is to replace them with the average value of the variable on a subset of observations that are the closest (most similar) to the observation(s) with the missing value. One way to do find the most similar observations is to apply the **kNN** method.

However, we will opt here for a simpler approach: we will replace the missing *Fare* value with the average *Fare* value for the passengers of the same class (*Pclass*).

First, we need to check the distribution of the *Fare* variable, to decide if we should use mean or median as the average value.

```
# test the Fare variable for normality
shapiro.test(titanic.test$Fare)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  titanic.test$Fare
## W = 0.5393, p-value < 2.2e-16
```

The variable is not normally distributed -> use median.

Now, identify the passenger class (*Pclass*) of the passenger whose *Fare* is missing.

```
# get the class of the observation with missing Fare variable
missing.fare.pclass <- titanic.test$Pclass[is.na(titanic.test$Fare)]
```

The passenger with missing Fare value comes from the 3rd class. Compute median *Fare* for all other passengers of the same class.

```
# calculate the median value for the Fare variable of all passangers from the 3rd class
median.fare <- median(x = titanic.test$Fare[titanic.test$Pclass == missing.fare.pclass],
                      na.rm = T) # we have to set this to true as Fare has one NA value
```

Set the missing *Fare* value to the computed median value.

```
# set the median value to the Fare variable of the passanger with a missing Fare
titanic.test$Fare[is.na(titanic.test$Fare)] <- median.fare
```

Check if the NA value was really replaced.

```
# print the summary of the test set
summary(titanic.test$Fare)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   7.896  14.454  35.561  31.472 512.329
```

Variables with many missing values and/or missing values that are difficult to replace

The *Age* variable is an example of the first type: variable with many missing values; *Cabin* is an example of the second type, as it is a categorical variable with many different values (~150)

For such variables the replacement of missing values is done through a process known as *imputation* - the process of replacing missing values with substituted (predicted) values. It is, in fact, the task of predicting (good substitutes for) the missing values. R has several packages for imputation: MICE, Amelia, HMisc,…

We are not going to do imputation (out of the scope of this course), but will instead create new variables (features) that will, in a way, serve as substitutes or proxies for *Age* and *Cabin*. This will be covered a bit later in the section on *Feature engineering*.

Feature selection

To select features to be used for creating a prediction model, we have to examine if and to what extent they are associated with the response (outcome) variable.

If we are familiar with the domain of the problem (prediction task), we can start from the knowledge and/or intuition about the predictors. Otherwise, that is, if the domain is unknown to us (for example, predictions related to some chemical reactions) or the real names (labels) of the variables are withdrawn (e.g. for privacy reasons), we have to rely on some well establish general methods for feature selection (such as forward or backward selection).

Since the Titanic data set is associated with a familiar domain, we can start from some intuition about potential predictors.

Examining the predictive power of variables from the data set

It's well-known that in disasters woman and children are often the first to be rescued. Let's check if that was the case in the Titanic case. We'll start by looking at the survival based on gender.

First, let's see the proportion of males and females in the dataset.

```
# transofrm the Sex variable into factor
titanic.train$Sex <- factor(titanic.train$Sex)

# get the summary of the Sex variable
summary(titanic.train$Sex)
```

```
## female   male
##    314    577
```

```r
# compute the proportions table of the Sex variable
prop.table(summary( titanic.train$Sex ))
```

```
##   female     male
## 0.352413 0.647587
```

Now, examine the survival counts based on the sex.

```r
# create a contingency table for Sex vs. Survived
sex.survived.counts <- xtabs(~Sex + Survived, data = titanic.train)
sex.survived.counts
```

```
##         Survived
## Sex        0   1
##   female  81 233
##   male   468 109
```

Get the proportions.

```r
# compute the proportions for Sex vs. Survived
sex.surv.tbl <- prop.table(sex.survived.counts,
                           margin = 1) # proportions are computed at the row level (each row sums to 1)
sex.surv.tbl
```

```
##         Survived
## Sex              0         1
##   female 0.2579618 0.7420382
##   male   0.8110919 0.1889081
```

Obviously, gender is highly associated with survival.

Before inspecting if/how age group has affected the chances for survival, let's quickly take a look at the potential impact of the passenger class (1st, 2nd or 3rd), as it is reasonable to expect that those from a higher class would have had higher chances of survival.

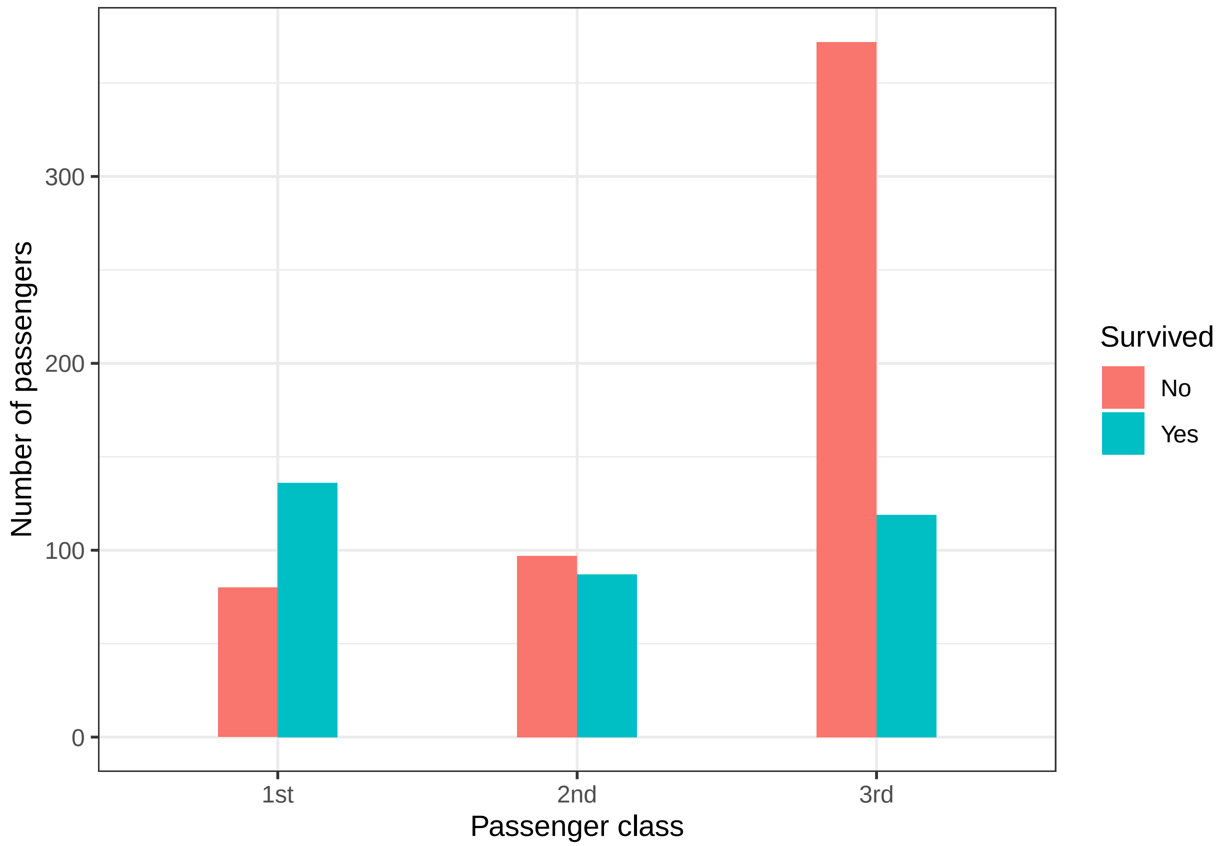
We can do that again using tables, but it might be more effective to examine it visually, using the ggplot2 package.

For plotting the survival against the passenger class, we need to transform both variables into factor variables (they are given as variables of type int).

```r
# transform the Survived variable into factor
titanic.train$Survived <- factor(titanic.train$Survived,
                                 levels = c(0,1), labels = c('No','Yes'))

# transform the Pclass variable into factor
titanic.train$Pclass <- factor(titanic.train$Pclass,
                               levels = 1:3,
                               labels = c("1st", "2nd", "3rd"))
```

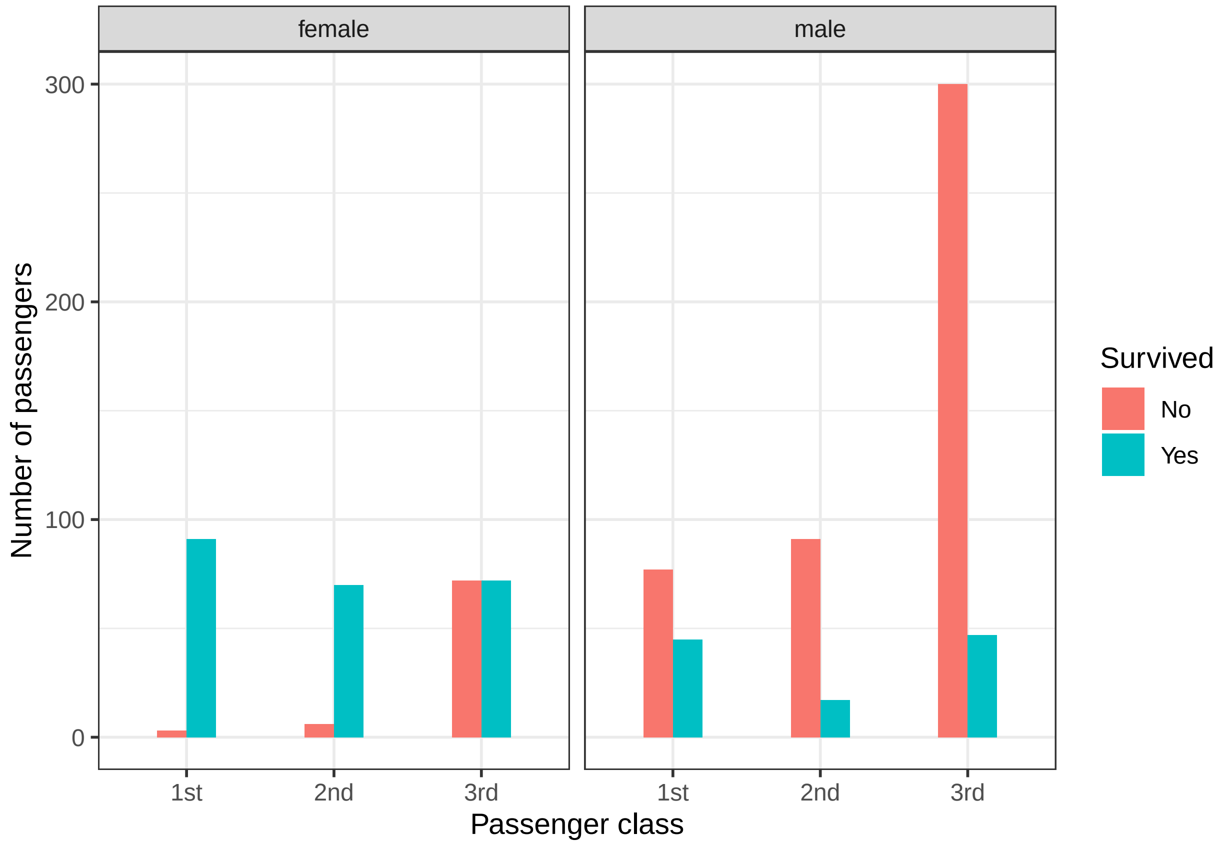
```r
# plot the number of passengers in different classes and Survived values
gp1 <- ggplot(titanic.train, aes(x = Pclass, fill=Survived)) +
  geom_bar(position = "dodge", width = 0.4) +
  ylab("Number of passengers") +
  xlab("Passenger class") +
  theme_bw()
gp1
```



The chart suggests that passenger class is another relevant predictor.

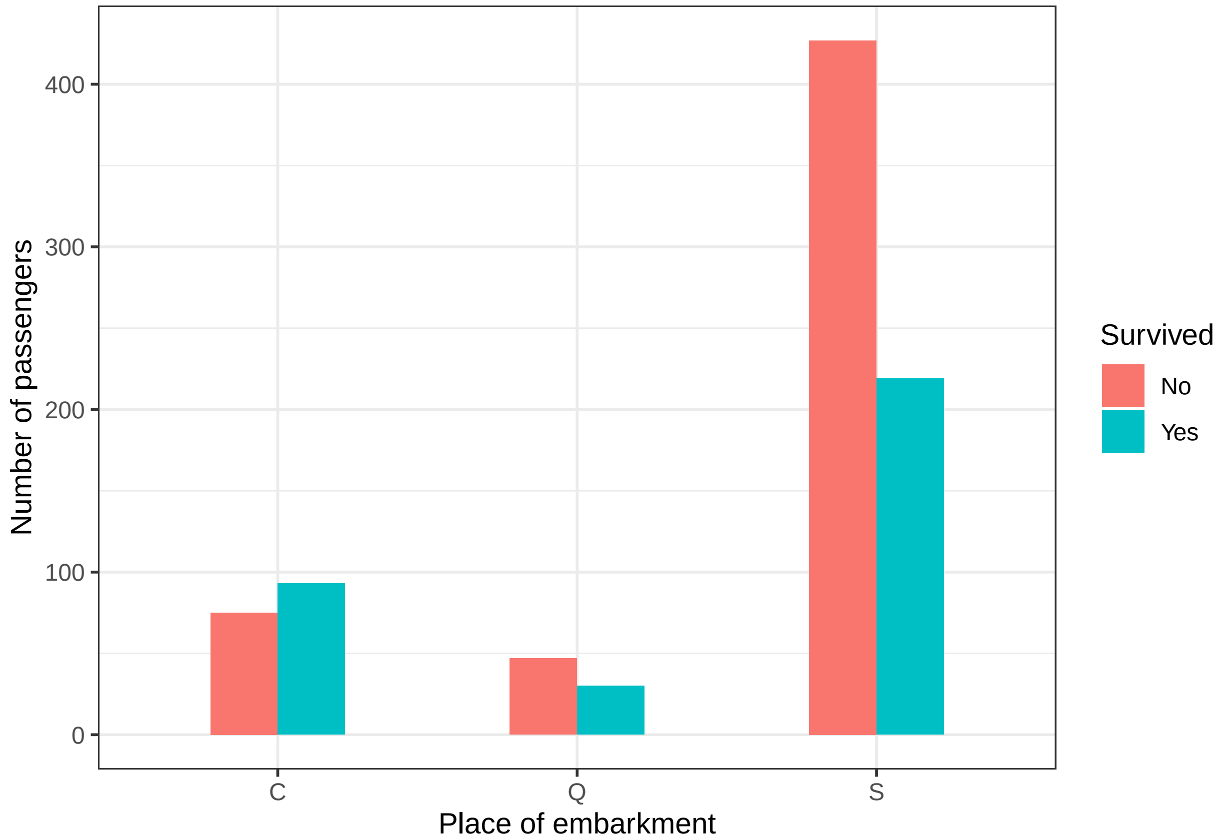
Let's examine passenger class and gender together.

```r
# add the Sex facet to the plot
gp2 <- gp1 + facet_wrap(~Sex)
gp2
```



Let's also inspect if the place of embarkment (the *Embarked* variable) affected the survival.

```
# plot the number of passengers for different embarkment places and Survived values
gp3 <- ggplot(titanic.train, aes(x = Embarked, fill = Survived)) +
  geom_bar(position = "dodge", width = 0.45) +
  ylab("Number of passengers") +
  xlab("Place of embarkment") +
  theme_bw()
gp3
```



It seems that those who embarked in Cherbourg had a higher chance of surviving than the passengers who embarked in the other two ports. Though not as strong as *Sex* and *Pclass*, this variable seems to be a viable candidate for a predictor.

Feature engineering

When creating new features (attributes) to be used for prediction purposes, we need to base those features on the data from both the training and the test sets, so that the features are available both for training the prediction model and making predictions on the unseen test data.

Hence, we will merge the training and the test sets and develop new features on the merged data.

But before we do that, we need to assure that the training and the test sets have exactly the same structure. To that end, we will first add the *Survived* column to the test data, with all NA values (as these values are unknown).

```
# add the Survived variable to the test set
titanic.test$Survived <- NA
```

Next, we need to transform the *Pclass*, *Sex*, and *Embarked* variables in the test set into factors, since we've done that in the training set (the structure should be exactly the same).

```
# transform the Pclass variable into factor (in the test set)
titanic.test$Pclass <- factor(x = titanic.test$Pclass,
                              levels = c(1,2,3), labels = c("1st", "2nd", "3rd"))
```

```r
# transform the Sex variable into factor (in the test set)
titanic.test$Sex <- as.factor(titanic.test$Sex)

# transform the Embarked variable into factor (in the test set)
titanic.test$Embarked <- as.factor(titanic.test$Embarked)
```

Now, we can merge the two datasets.

```r
# merge train and test sets
titanic.all <- rbind(titanic.train, titanic.test)
```

Creating an age proxy variable

Recall that the *Age* variable has a lot of missing values, and simple imputation methods we considered cannot be used in such cases. So, we will create a new variable that approximates the passengers' age group. We'll do that by making use of the *Name* variable.

To start, let's first inspect values of the Name variable.

```r
# print a sample of the Name variable
titanic.all$Name[1:10]
```

```
##  [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"
##  [7] "McCarthy, Mr. Timothy J"
##  [8] "Palsson, Master. Gosta Leonard"
##  [9] "Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)"
## [10] "Nasser, Mrs. Nicholas (Adele Achem)"
```

We can observe that the *Name* variable consists of surname, title, first name, and in some cases additional name (maiden name of a married woman).

The idea is to use the title of a person as a rough proxy for his/her age.

First, we need to extract the title from the *Name* variable; to that end, we'll split the Name string using "," or "." as delimiters; lets' try it first. Note that a dot (".") is a special character in regular expressions and has to 'escaped', that is, preceded with a slash; since a slash is also a special character that has to be 'escaped', we end up with a double slash followed by a dot (\\.)

```r
# split the name of the first observation on , or . characters
strsplit(x = titanic.all$Name[1], split = ",|\\.")
```

```
## [[1]]
## [1] "Braund"       " Mr"          " Owen Harris"
```

We get a list of vectors, where each vector consists of pieces of a person's name. To extract the title, we need to simplify the output, so that instead of a list, we get a vector (with the elements of a person's name).

```r
# split the name of the first observation on , or . characters and unlist
unlist(strsplit(x = titanic.all$Name[1], split = ",|\\."))
```

```
## [1] "Braund"       " Mr"          " Owen Harris"
```

and then, take the second element of that vector:

```r
# split the name of the first observation on , or . characters, unlist and take the 2nd elem.
unlist(strsplit(x = titanic.all$Name[1], split = ",|\\."))[2]
```

```
## [1] " Mr"
```

You might have noticed a space before the title, we'll remove that quickly, but before that, we'll apply this procedure to all the rows in the titanic.all dataset to create a new feature:

```r
# create a new variable - Title - based on the value of the Name variable
titanic.all$Title <- sapply(titanic.all$Name,
                            FUN = function(x) unlist(strsplit(x, split = ",|\\."))[2] )
```

Now, let's remove that leading blank space.

```r
# remove the leading space character from the Title
titanic.all$Title <- trimws(titanic.all$Title, which = "left")
```

We can now inspect different kinds of titles we have in the dataset.

```r
# print the contingency table for the Title values
table(titanic.all$Title)
```

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

There are some rarely occurring titles that won't be useful for creating a model; so, we'll aggregate those titles into broader categories that represent some basic age-gender groups:

```r
# create a vector of all women (adult female) titles
adult.women <- c("Dona", "Lady", "Mme", "Mrs", "the Countess")

# create a vector of all girl (young female) titles
girls <- c("Ms", "Mlle", "Miss")

# create a vector of all men (adult male) titles
adult.men <- c("Capt", "Col", "Don", "Dr", "Major", "Mr", "Rev", "Sir")

# create a vector of all boy (young male) titles
boys <- c("Master", "Jonkheer")
```

First, we'll introduce a new variable (feature) to represent the age-gender group.

```
# introduce the AgeGender variable
titanic.all$AgeGender <- NA
```

and, now define each age-gender group using the Title groupings we defined above.

```
# set the AgeGender value based on the vector the Title value belongs to
titanic.all$AgeGender[ titanic.all$Title %in% adult.women ] <- "Adult_Female"
titanic.all$AgeGender[ titanic.all$Title %in% adult.men ] <- "Adult_Male"
titanic.all$AgeGender[ titanic.all$Title %in% girls ] <- "Young_Female"
titanic.all$AgeGender[ titanic.all$Title %in% boys ] <- "Young_Male"
```

Note: the *%in%* operator checks to see if a value is an element of the given vector.

Let's see how passengers are distributed across our age-gender groups:

```
# print the contingency table for the AgeGender values
table(titanic.all$AgeGender)
```

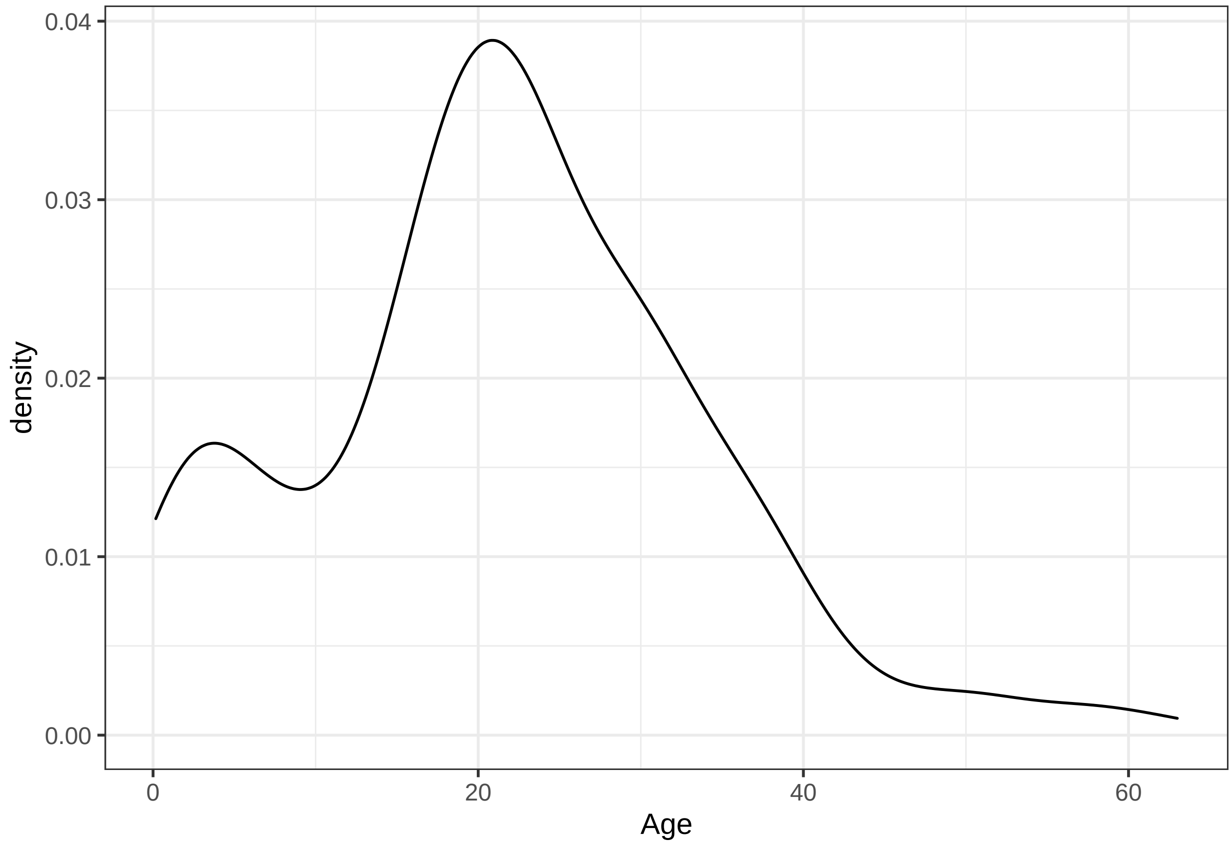
```
##
## Adult_Female   Adult_Male Young_Female   Young_Male
##          201          782          264           62
```

We observe a high disproportion in the number of boys and girls, and man and woman. Let's take a closer look at the groups with an unexpectedly high number of passengers, namely *Young_Female* and *Adult_Male* groups.

We'll make use of the available values of the *Age* variable to see how our *Young_Female* group is distributed with respect age.

```
# plot the distribution of the Age attribute in the Young_Female group
ggplot(titanic.all[titanic.all$AgeGender=="Young_Female",], aes(x = Age)) +
  geom_density() +
  theme_bw()
```

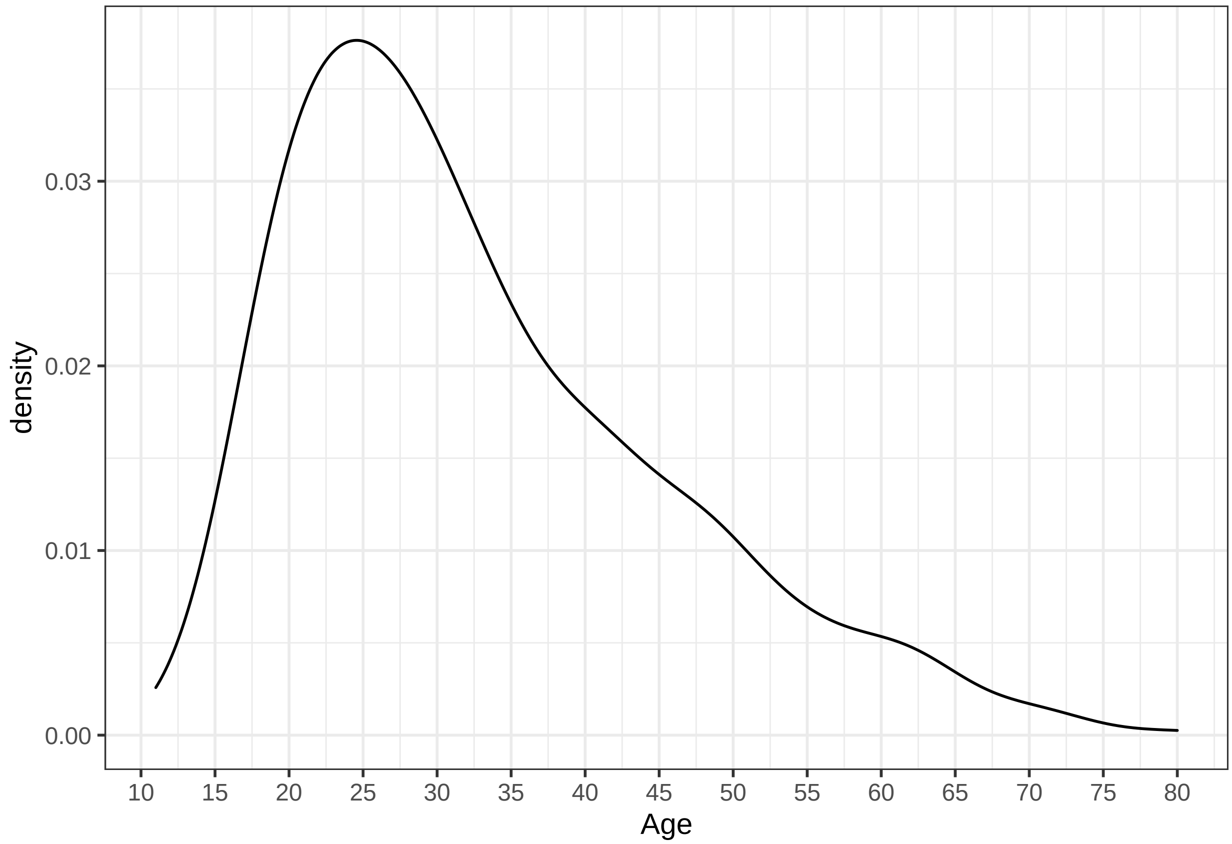
```
## Warning: Removed 51 rows containing non-finite values (stat_density).
```



It is obvious from the graph that the *Young_Female* group includes a considerable number of adult women. We'll need to fix this. But before that, let's also inspect the *Adult_Male* group.

```
# plot the distribution of the Age attribute in the Adult_Male group
ggplot(titanic.all[titanic.all$AgeGender=="Adult_Male", ], aes(x = Age)) +
  geom_density() +
  scale_x_continuous(breaks = seq(5,80,5)) +
  theme_bw()
```

```
## Warning: Removed 177 rows containing non-finite values (stat_density).
```



From this plot, we can see that the *Adult_Male* group also includes some males who cannot be qualified as adults.

We will try to fix both problems using the available values of the *Age* variable.

First, let's check for how many passengers in the 'Young_Female' group the *Age* value is available.

```
# print the number of young females who has the Age value set
nrow(titanic.all[titanic.all$AgeGender=="Young_Female" & !is.na(titanic.all$Age),])
```

```
## [1] 213
```

So, we have *Age* value for 213 out of 264 Girls, which is not bad at all (80%). We'll make use of these available *Age* values to move some *Young_Female* to *Adult_Female* group, using 18 years of age as the threshold.

```
# set the AgeGender to 'Adult_Female' for all 'girls' with age over 18
titanic.all$AgeGender[titanic.all$AgeGender=="Young_Female" &
                        !is.na(titanic.all$Age) &
                        titanic.all$Age >= 18] <- "Adult_Female"
```

We'll do a similar thing for the *Adult_Male* group. First, check the number of *Adult_Male* passengers for whom age is available.

```
# print the number of adult males who has the Age value set
nrow(titanic.all[titanic.all$AgeGender=="Adult_Male" & !is.na(titanic.all$Age),])
```

```
## [1] 605
```

We have *Age* value for 605 out of 782 *AdultMen* passengers (77%). Let's make use of those values to move some passengers from *Adult_Male* to *Young_Male* group using, again, the 18 year threshold.

```
# set the AgeGender to 'Young_Male' for all 'Adult_Male' with age under 18
titanic.all$AgeGender[titanic.all$AgeGender=="Adult_Male" &
                        !is.na(titanic.all$Age) &
                        titanic.all$Age < 18] <- "Young_Male"
```

Let's check the *AgeGender* proportions after these modifications.

```
# print the contingency table for the AgeGender variable
table(titanic.all$AgeGender)
```

```
##
## Adult_Female   Adult_Male Young_Female   Young_Male
##          347          753          118           91
```

```
# print the proportions table for the AgeGender variable
round(prop.table(table(titanic.all$AgeGender)), digits = 2)
```

```
##
## Adult_Female   Adult_Male Young_Female   Young_Male
##         0.27         0.58         0.09         0.07
```

This looks far more realistic.

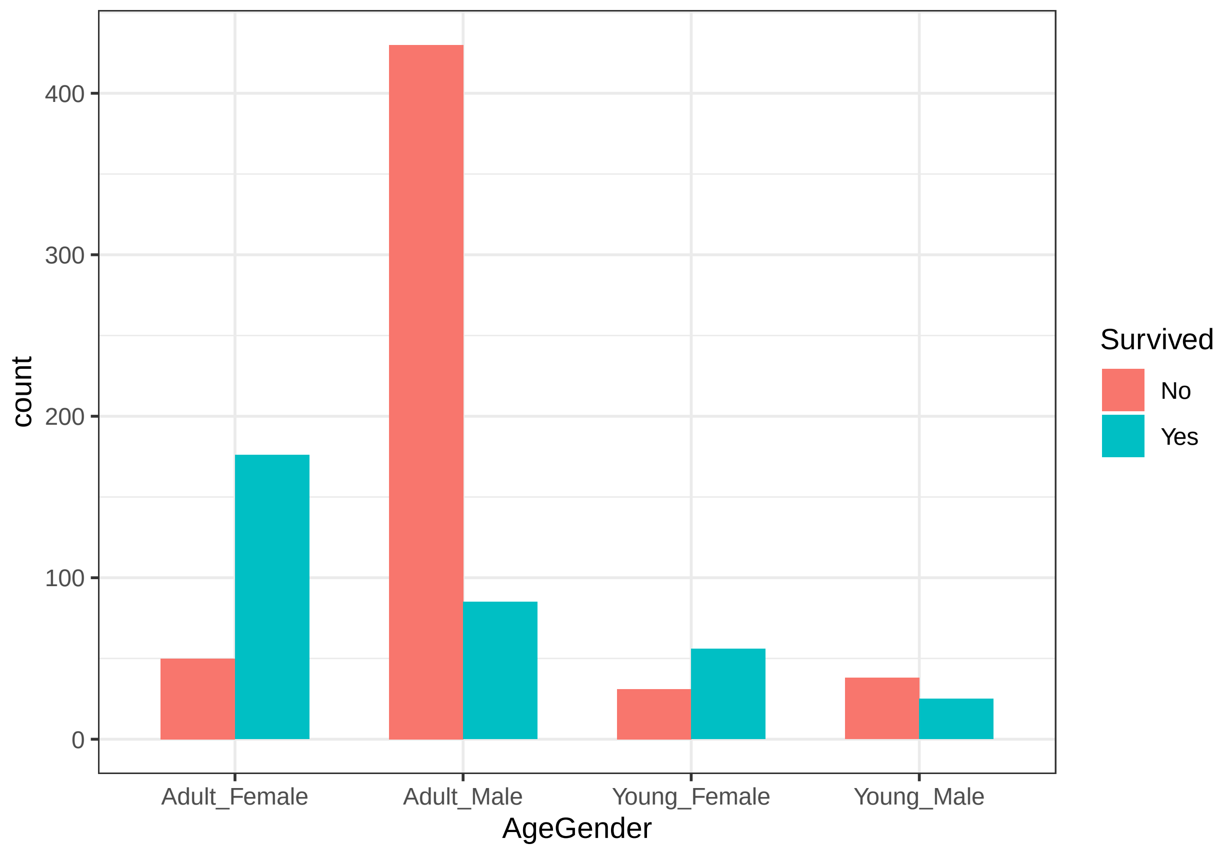
Finally, we'll transform *AgeGender* into a factor variable, so that it can be better used for data exploration and prediction purposes.

```
# transform the AgeGender to factor
titanic.all$AgeGender <- factor(titanic.all$AgeGender)
summary(titanic.all$AgeGender)
```

```
## Adult_Female   Adult_Male Young_Female   Young_Male
##          347          753          118           91
```

Let's see if our efforts in creating the *AgeGender* variable were worthwhile, that is, if *AgeGender* is likely to be a significant predictor. To that end, we will plot the *AgeGender* groups against the *Survival* variable.

```
# plot the AgeGender against Survived attribute
ggplot(titanic.all[!is.na(titanic.all$Survived),],
       aes(x = AgeGender, fill=Survived)) +
  geom_bar(position = "dodge", width = 0.65) +
  theme_bw()
```



Note: we are using only the first 891 observations in the merged dataset as these are observations from the training set for which we know the outcome (i.e., survival).

Let's examine this also as percentages. First, we need to compute the percentages.

```
# calculate the proportions for AgeGender and Survived values
age.gen.surv.tbl <- prop.table(table(AgeGender = titanic.all$AgeGender,
                                     Survived = titanic.all$Survived,
                                     useNA = 'no'),
                               margin = 1)
age.gen.surv.tbl
```

```
##                Survived
## AgeGender              No       Yes
##   Adult_Female 0.2212389 0.7787611
##   Adult_Male   0.8349515 0.1650485
##   Young_Female 0.3563218 0.6436782
##   Young_Male   0.6031746 0.3968254
```

Note that we are setting the margin parameter to 1 as we want to have proportions of survived and not-survived (column values) computed for each AgeGender group (row) individually. Try setting the margin to 2 and not setting it at all to observe the effect.

For plotting, we'll transform the table into a data frame.

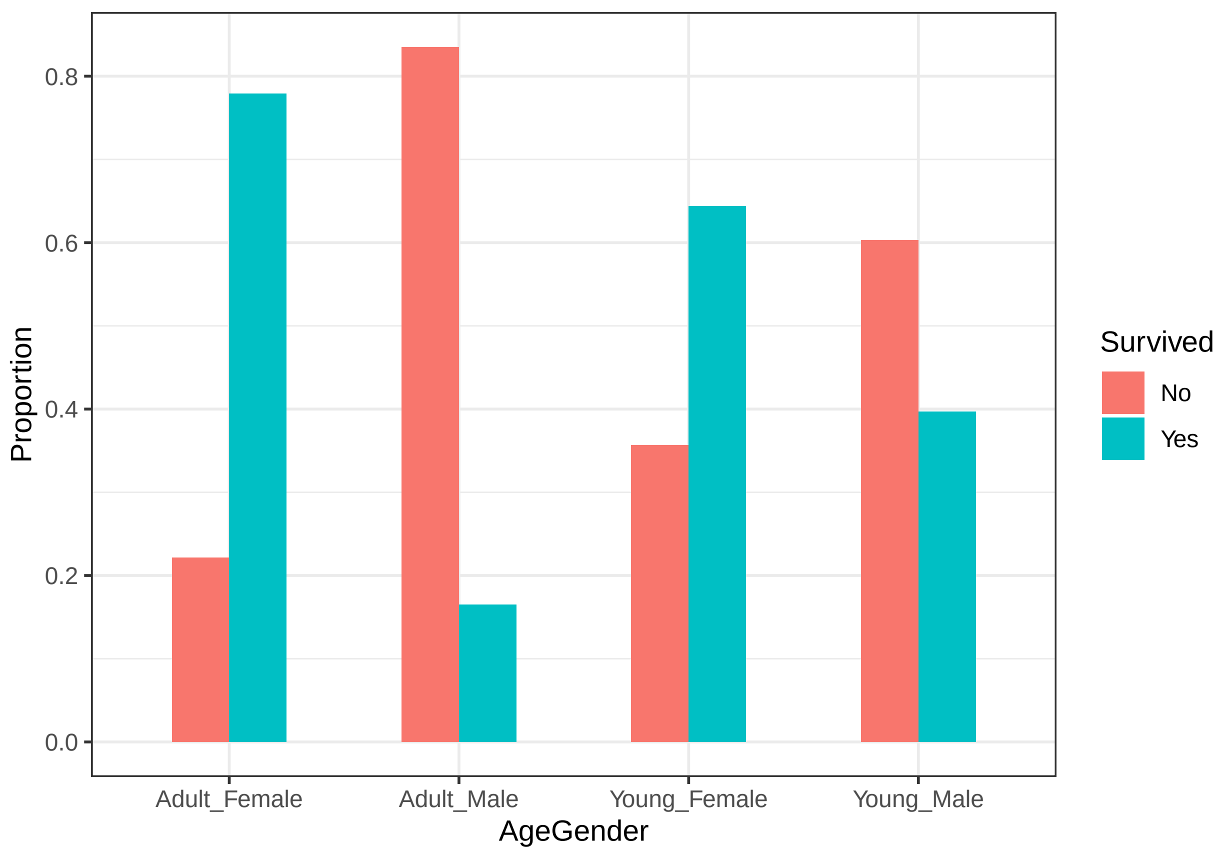
```r
# transform the proportions table in a dataframe
age.gen.surv.df <- as.data.frame(age.gen.surv.tbl)
age.gen.surv.df
```

```
##     AgeGender Survived      Freq
## 1 Adult_Female       No 0.2212389
## 2   Adult_Male       No 0.8349515
## 3 Young_Female       No 0.3563218
## 4   Young_Male       No 0.6031746
## 5 Adult_Female      Yes 0.7787611
## 6   Adult_Male      Yes 0.1650485
## 7 Young_Female      Yes 0.6436782
## 8   Young_Male      Yes 0.3968254
```

Note the difference in the structure of the table and the data frame.

```r
# change the name of the last column to better reflect its meaning
colnames(age.gen.surv.df)[3] <- "Proportion"
```

```r
# plot the AgeGender vs. Proportion vs. Survived
ggplot(age.gen.surv.df, aes(x = AgeGender, y = Proportion, fill=Survived)) +
  geom_col(position = "dodge", width = 0.5) +
  theme_bw()
```



Obviously, the age/gender group is a strong predictor of survival.

Creating the FamilySize variable

Recall that we have two variable related to the number of family members one is traveling with:

- *SibSp* - the number of siblings and spouses a passenger is traveling with
- *Parch* - the number of parents and children one is traveling with

```
# examine the values of the SibSp variable
summary(titanic.all$SibSp)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.4989  1.0000  8.0000
```

```
table(titanic.all$SibSp)
```

```
##
##   0   1   2   3   4   5   8
## 891 319  42  20  22   6   9
```

```
# examine the values of the Parch variable
summary(titanic.all$Parch)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   0.000   0.385   0.000   9.000
```

```
table(titanic.all$Parch)
```

```
##
##    0    1    2    3    4    5    6    9
## 1002  170  113    8    6    6    2    2
```

To make it easier to keep track of the number of family members one was traveling with, we'll create a new variable *FamilySize* by simply adding the value of the *SibSp* and *Parch* variables.

```
# create a new variable FamilySize based on the SibSp and Parch values
titanic.all$FamilySize <- titanic.all$SibSp + titanic.all$Parch
summary(titanic.all$FamilySize)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.8839  1.0000 10.0000
```

We can observe that a large majority of passengers didn't travel with family members.

```
# print the contingency table for the FamilySize
table(titanic.all$FamilySize)
```

```
##
##   0   1   2   3   4   5   6   7  10
## 790 235 159  43  22  25  16   8  11
```

It can be also observed that those who traveled with 3+ family members were not that numerous.

```r
# compute the proportion of FamilySize >= 3 in all passangers
sum(titanic.all$FamilySize>=3)/length(titanic.all$FamilySize)
```

```
## [1] 0.09549274
```

Less than 10% of passengers traveled with 3+ family members.

In situations like this - several values of a variable spread across a small proportion of the observations - it is recommended to aggregate those values. We'll apply that practice to the FamilySize variable and aggregate observations with 3+ family members.

```r
# set the FamilySize to 3 to all observations where FamilySize > 3
titanic.all$FamilySize[titanic.all$FamilySize > 3] <- 3
```

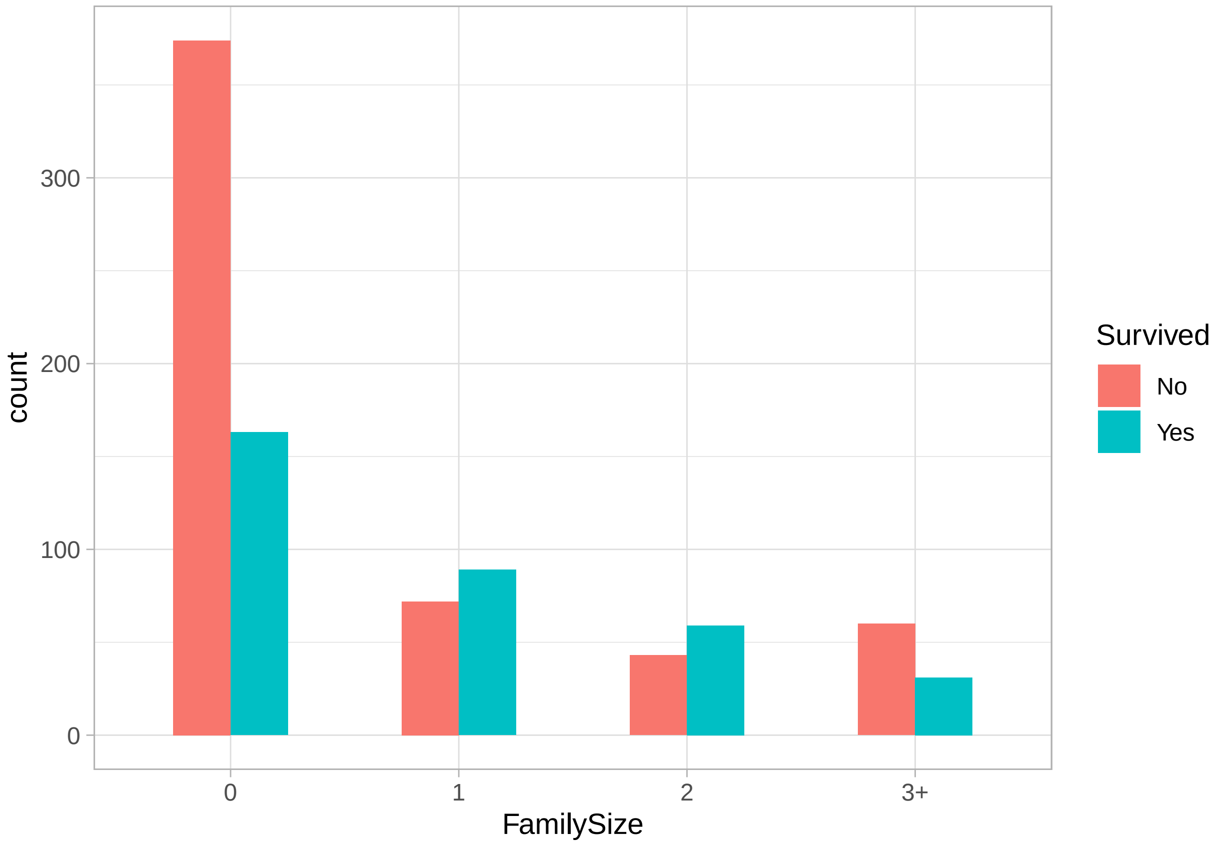
Turn *FamilySize* into a factor.

```r
# transform FamilySize into factor
titanic.all$FamilySize <- factor(titanic.all$FamilySize,
                                  levels = 0:3, labels = c(0:2, "3+"))
table(titanic.all$FamilySize)
```

```
##
##   0   1   2  3+
## 790 235 159 125
```

Let's see how this new feature affects survival prospects.

```r
# plot the FamilySize vs. Survived
ggplot(titanic.all[!is.na(titanic.all$Survived),],
       aes(x = FamilySize, fill = Survived)) +
  geom_bar(position = "dodge", width = 0.5) +
  theme_light()
```



We can see that those who traveled with 1 or 2 family members had better prospects than those who traveled without family members or with 3+ family members.

Making use of the Ticket variable

Let's examine the *Ticket* variable and see if we can make some use of it.

```
# print a sample of Ticket values
titanic.all$Ticket[1:20]
```

```
##  [1] "A/5 21171"          "PC 17599"           "STON/O2. 3101282" "113803"
##  [5] "373450"             "330877"             "17463"            "349909"
##  [9] "347742"             "237736"             "PP 9549"          "113783"
## [13] "A/5. 2151"          "347082"             "350406"           "248706"
## [17] "382652"             "244373"             "345763"           "2649"
```

We can observe that some tickets start with letters, while others consist of digits only.

```
# compute the number of distinct values of the Ticket variable
length(unique(titanic.all$Ticket))
```

```
## [1] 929
```

929 unique ticket values for 1309 passengers suggests that some passengers were traveling on the same ticket. Let's examine this further since the presence of shared tickets is an indicator that some passengers were not

traveling alone, and we saw that the number of people one was traveling with might have had effect on their survival prospects.

```r
# use tapply to compute the number of occurrences of each unique Ticket value
ticket.count <- tapply(titanic.all$Ticket,
                       INDEX = titanic.all$Ticket,
                       FUN = length)

# create a data frame with ticket name and ticket count as variables
ticket.count.df <- data.frame(ticket=names(ticket.count),
                              count=as.integer(ticket.count))

# print first few rows of the new data frame
head(ticket.count.df)
```

```
##   ticket count
## 1 110152     3
## 2 110413     3
## 3 110465     2
## 4 110469     1
## 5 110489     1
## 6 110564     1
```

Let's examine the number of passengers per single and shared tickets.

```r
# print the contingency table of the count variable
table(ticket.count.df$count)
```

```
##
##   1   2   3   4   5   6   7   8  11
## 713 132  49  16   7   4   5   2   1
```

We can see that the majority of passengers traveled on a single person ticket, a considerable number of them shared a ticket with one person, and a small number shared their ticket with 3+ people.

We'll add ticket count to each passenger by merging *titanic.all* dataset with the *ticket.count.df* based on the ticket value.

```r
# merge titanic.all and ticket.count.df datasets on the Ticket variable
titanic.all <- merge(x = titanic.all, y = ticket.count.df,
                     by.x = "Ticket", by.y = "ticket")

# change the name of the newly added column to PersonPerTicket
colnames(titanic.all)[16] <- "PersonPerTicket"
```

As we did with *FamilySize*, we'll aggregate infrequent values of *PersonPerTicket* and transform the variable into a factor.

```r
# set the PersonPerTicket to 4 to all observations where PersonPerTicket > 4
titanic.all$PersonPerTicket[titanic.all$PersonPerTicket > 4] <- 4

# convert PersonPerTicket to factor
titanic.all$PersonPerTicket <- factor(titanic.all$PersonPerTicket,
```

```
                                          levels = 1:4,
                                          labels = c(1:3, "4+"))

# print the contingency table for the PersonPerTicket
table(titanic.all$PersonPerTicket)
```

```
##
##   1   2   3  4+
## 713 264 147 185
```

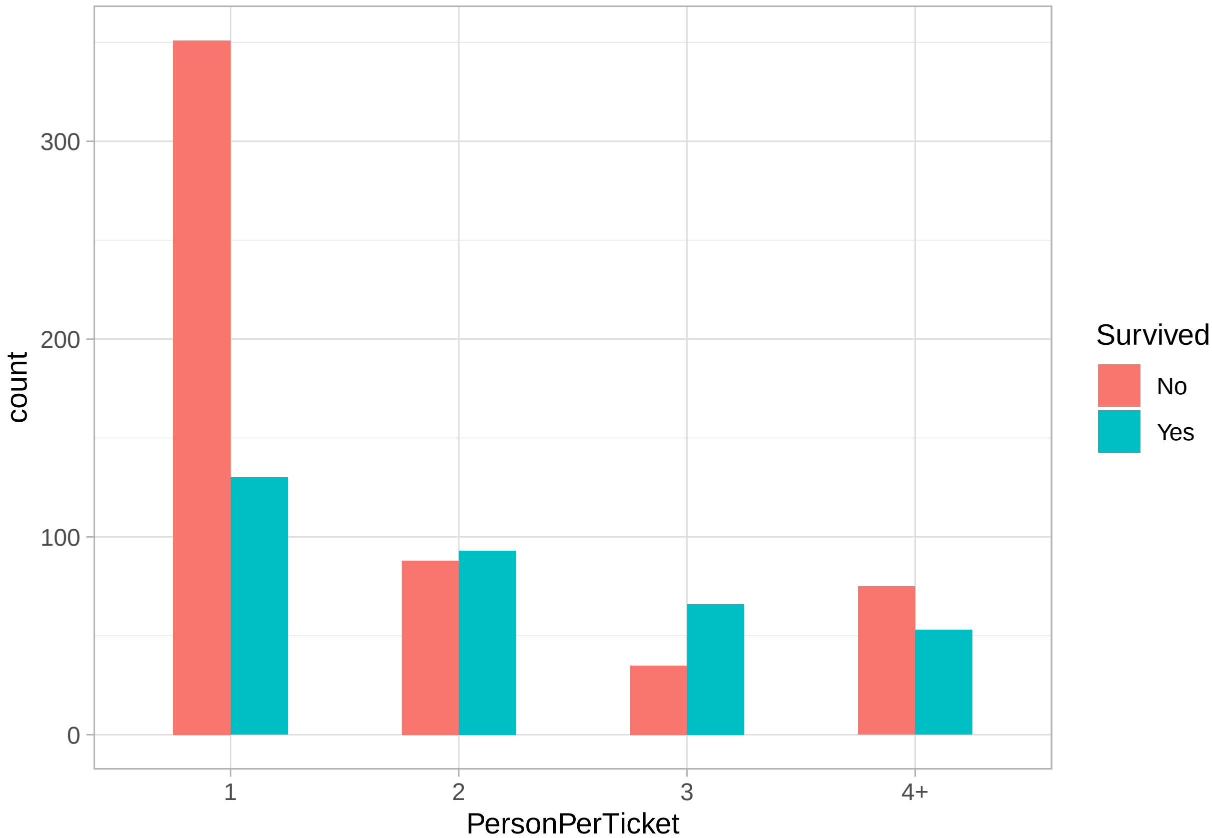
Out of curiosity, we can crosstab this variable with *FamilySize* to see if there were some passengers who were not traveling with family members but still had company, as well as those who really traveled alone.

```
# print the contingency table for the PersonPerTicket vs. FamilySize
xtabs(~ PersonPerTicket + FamilySize, data = titanic.all)
```

```
##                FamilySize
## PersonPerTicket   0   1   2  3+
##              1  663  31  16   3
##              2   62 170  25   7
##              3   21  26  98   2
##              4+  44   8  20 113
```

Let's examine the *PersonPerTicket* feature from the perspective of its relevance for a passenger's survival.

```
# plot all survived passangers (without NAs)
ggplot(titanic.all[!is.na(titanic.all$Survived),],
       aes(x = PersonPerTicket, fill=Survived)) +
  geom_bar(position = "dodge", width = 0.5) +
  theme_light()
```



It seems that this feature could be a useful predictor.

Note that when we merged the *titanic.all* and *ticket.count.df* data frames, the order of rows in the *titanic.all* changed, so it is not the case anymore that the first 891 observations are those taken from the training set and the rest are from the test set. Therefore, in the data argument (of *ggplot*) we had to select observations based on having value for the *Survived* attribute.

Let's also check what a plot based on percentages would look like.

Compute first the percentages of survived and not survived for each *PersonPerTicket* value:

```
# calculate the proportions of the PersonPerTicket vs. Survived table
tcount.surv.tbl <- prop.table(table(PersonPerTicket = titanic.all$PersonPerTicket,
                                    Survived = titanic.all$Survived,
                                    useNA = "no"),
                              margin = 1)
tcount.surv.tbl
```

```
##                Survived
## PersonPerTicket        No       Yes
##              1  0.7297297 0.2702703
##              2  0.4861878 0.5138122
##              3  0.3465347 0.6534653
##              4+ 0.5859375 0.4140625
```

In the *table* funcion we used the *useNA* argument to restrict the computations to only those observations where the *Survived* variable is not NA (that is, observations are from the training set).

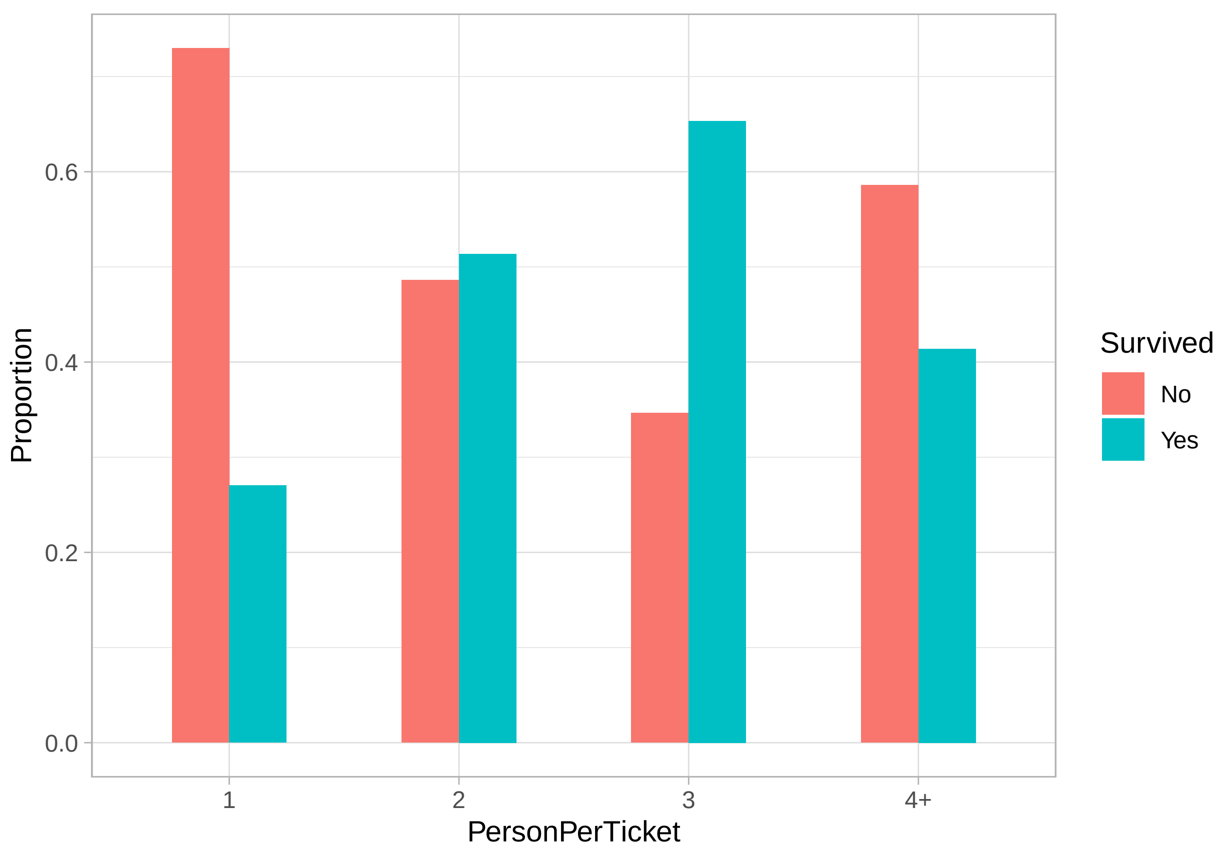
Transform the table into a data frame (required for plotting):

```r
# convert the table into a data frame
tcount.surv.df <- as.data.frame(tcount.surv.tbl)
tcount.surv.df
```

```
##   PersonPerTicket Survived      Freq
## 1               1       No 0.7297297
## 2               2       No 0.4861878
## 3               3       No 0.3465347
## 4              4+       No 0.5859375
## 5               1      Yes 0.2702703
## 6               2      Yes 0.5138122
## 7               3      Yes 0.6534653
## 8              4+      Yes 0.4140625
```

```r
# change the name of the last column to better reflect its meaning
colnames(tcount.surv.df)[3] <- "Proportion"
```

```r
# plot the PersonPerTicket vs. Proportion barchart, split based on the Survived attribute
ggplot(tcount.surv.df, aes(x = PersonPerTicket, y = Proportion, fill=Survived)) +
  geom_col(width = 0.5, position = "dodge") +
  theme_light()
```



The proportions of survived and those who did not are very similar for those who travelled on a shared ticket (values 2 and 3+ of the PersonPerTicket variable), but significantly different from those who travelled on a single ticket. So, while not effective as some of the previously considered variables, this variable would be worth including in a prediction model.

TASK: Create a binary variable TravelledAlone that would have value TRUE for those who travelled on a single ticket without family members, and FALSE otherwise. Use plots to examine its predictive power.

Save the augmented data set

Finally, let's split the augmented data set again into training and test parts and save them.

Training observations are those that have the *Survived* value set; test observations have NA value for the *Survived* attribute

```
# split into train and test sets based on whether the Survived is present
ttrain.new <- titanic.all[!is.na(titanic.all$Survived),]
ttest.new <- titanic.all[is.na(titanic.all$Survived),]
```

```
# save both data sets to a file
saveRDS(ttrain.new, file = "data/train_new.RData")
saveRDS(ttest.new, file = "data/test_new.RData")
```