Intro R Lab 3: Example Data Set

Simon Caton

In this lab we are going to do the following:

1. Read in and explore a real world dataset
2. Deal with missing values
3. Engineer some features within the dataset
4. Graphically explore the data
5. Do some elementary modelling of the data

Reading in a structured dataset

There is an interesting keywork in this header here: **structured**. Whilst so far we have mentioned data types (numerical, logicals, vectors etc.), and kinds of data (numerical and categorical), we haven't yet mentioned ways to categorise how data is represented.

The easiest data, you will have to deal with is structured data. Think back to your first databases class; if a dataset adheres to first normal form (1NF) it's structured in that there is a predefined and also constant structure. If you've forgotten your normal forms (how could you!?) the wikipedia article on database normalisation https://en.wikipedia.org/wiki/Database_normalization isn't too bad, but this would be a better read http://dl.acm.org/citation.cfm?id=358054.

At the other end of the scale, we have unstructured data, typically this will not have a standard or to be more precise easily *parsable* format. Examples here are things like images, videos, text (e.g. Facbeook posts, newspaper articles, etc.). To get a better understanding of the difference to structured data, there is a nice article here https://datascience.berkeley.edu/structured-unstructured-data/.

For today, we're focusing on structured data, a csv (comma separated value) file to be more precise, concerned with passengers of the Titanic.

Loading and Installing R packages

Today, we're going to go beyond standard built-in R functionality. So it would be worth a small disgression to explain how *packages* in R work.

All R functions and datasets are stored in packages. Only when a package is loaded are its contents available. This is done both for e ciency (the full list would take more memory and would take longer to search than a subset), and to aid package developers, who are protected from name clashes with other code.

To see which packages are installed at your site, issue the command

```
library()
```

To load a particular package use a command like

```
library(packageName)
```

When connected to the Internet you can use the *install.packages()* and *update.packages()* functions.

There are thousands of contributed packages for R, written by many different authors. Some of these packages implement specialized statistical methods, others give access to data or hardware, and others are designed to complement textbooks.

Packages have *namespaces*, which do three things: they allow the package writer to hide functions and data that are meant only for internal use, they prevent functions from breaking when a user (or other package writer) picks a name that clashes with one in the package, and they provide a way to refer to an object within a particular package.

For example, t() is the transpose function in R, but users might define their own function named t. Namespaces prevent the user's definition from taking precedence, and breaking every function that tries to transpose a matrix.

There is a useful operators that work with namespaces: The double-colon operator :: selects definitions from a particular namespace. In the example above, the transpose function will always be available as base::t, because it is defined in the base package. Only functions that are exported from the package can be retrieved in this way.

Whenever you see a package being loaded, you may find that you need to install it, at least until you have used R a fair bit!

For this lab, you will probably need to do the following:

```
install.packages(c("ggplot2", "ggthemes", "scales", "dplyr", "mice", "randomForest"))
```

Loading and preparing the Titanic dataset

Head over to https://www.kaggle.com/c/titanic and download train.csv.

A description of the dataset is as follows:

Attribute	Description
survival	(0 = No; 1 = Yes)
pclass	Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
name	Name
sex	Sex
age	Age
sibsp	Number of Siblings/Spouses Aboard
parch	Number of Parents/Children Aboard
ticket	Ticket Number
fare	Passenger Fare
cabin	Cabin
embarked	Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Now switch your working directory to where ever you have downloaded the file. For me, that would be as follows:

```
setwd("/Users/scaton/Downloads")
```

If you are a Windows user, remember to escape your \ s, as \ (as mentioned in lab 1) is a reserved character and so must be escaped. E.g.: *C:\\some directory\\some other directory*.

Now we are ready to read in our csv and put it in a data.frame, but you can also refer directly to the file too:

```
titanicData <- read.csv("titanic.csv", header=T, na.strings=c(""), stringsAsFactors = T)
```

Note that I also renamed train.csv to titanic.csv.

Tasks

1. Spend some time to understand what the different parameters passed to read.csv do
2. Encode the following attributes as factors

a. Survived, and
b. Pclass

3. Check all attributes for missing values

If you did Task 3 x12 times and didn't Google a quicker way, here are two quicker ways:

```
sapply(titanicData,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         687           2
```

Here we use another of R's loop functions *sapply*, which applies a function (*sum(is.na(x))*) over a list or vector. We'll come back to defining functions later.

We can also graphically inspect the data

```
install.packages("Amelia")
```

```
library(Amelia)
```

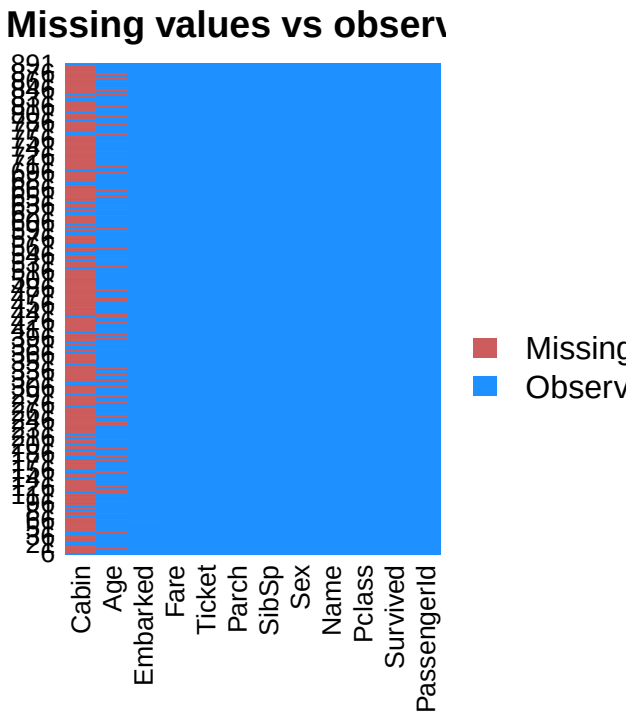
```
## Warning: package 'Amelia' was built under R version 3.4.4
```

```
## Loading required package: Rcpp
```

```
## Warning: package 'Rcpp' was built under R version 3.4.4
```

```
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.5, built: 2018-05-07)
## ## Copyright (C) 2005-2018 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```
#Visual representation of missing data
missmap(titanicData, main = "Missing values vs observed")
```



So we can see, this is no longer a toy dataset perfectly prepared for you to use.

Dealing with missing values

There are a number of different ways we could go about doing this. Given the small size of the dataset, we probably should not opt for deleting either entire observations (rows) or variables (columns) containing missing values. We're left with the option of either replacing missing values with a sensible values given the distribution of the data, e.g., the mean, median or mode. Finally, we could go with prediction. We'll use both of the two latter methods and rely on some data visualization to guide some decisions.

Embarked

We know that there are 2 passengers where Embarked is missing, this corresponds to 0.2244669% of our data. The simple option would be to delete these rows:

```
titanicData <- titanicData[!is.na(titanicData$Embarked), ]
```

but that's no fun. So let's see who is missing this data:

```
paste("PassengerId: ", titanicData[is.na(titanicData$Embarked), 1], " needs to be corrected")
```

```
## [1] "PassengerId:  62  needs to be corrected"
## [2] "PassengerId:  830  needs to be corrected"
```

For the moment let's temporarily remove these two. One of the nice things with R is that there is always a lot of different ways to do perform simple as well as complex tasks.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
embarked <- titanicData %>%
  filter(PassengerId != 62 & PassengerId != 830)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

Ok, so this is where the need for some independent and critical thinking comes in. There are different ways, to solve the challenge of missing data, but it starts the same way every time: explore.

There are really quite a few options. Pclass, fare, and name are obvious options. However, the latter is quite complex (at the moment). Let's start with fare, and use some simple visualisation to help us out.

Task Get these three details (class, fare, and name) of the two passengers.

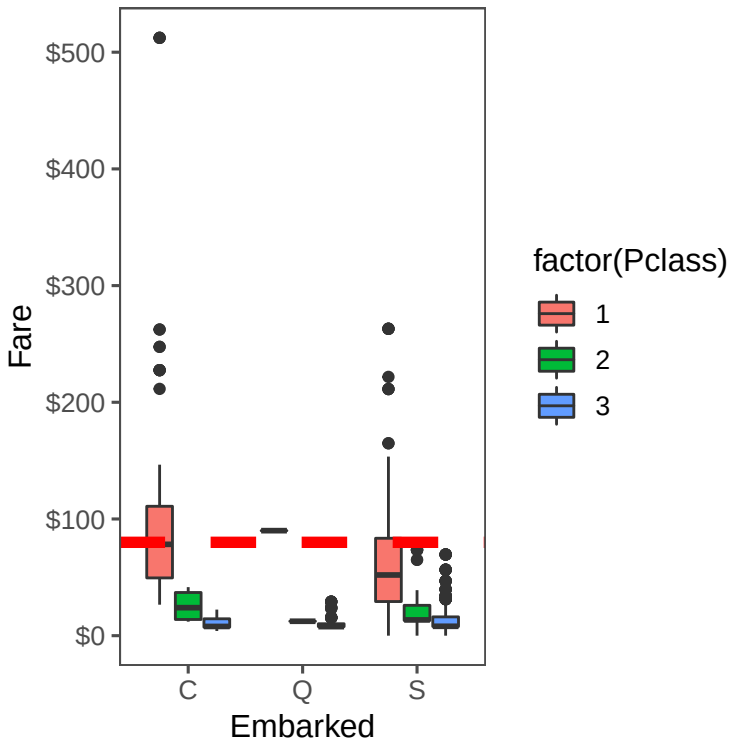
```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 3.4.4
```

```
library(scales)
# Use ggplot2 to visualize embarkment, passenger class, & median fare
ggplot(embarked, aes(x = Embarked, y = Fare, fill = factor(Pclass))) +
  geom_boxplot() +
  geom_hline(aes(yintercept=80), #<--- the price we know for one passenger
    colour='red', linetype='dashed', lwd=2) +
  scale_y_continuous(labels=dollar_format()) +
  theme_few()
```



Passenger 62 we know was in class 1 (at least if you did the above task correctly!). So from this figure, we can suppose that they probably bordered borded in C (Cherbourg), as the price they paid is almost the mean price of 1st class passengers that boarded here. We can dismiss Queenstown simply because there are so few Passengers boarding here. We can dismiss S (Southampton) as the price of 80 would approaching the third quartile of the prices (or it would be more expensive than almost 75% of passengers that boarded here). There is absolutely no guarantee that we are right, we are just trying to select the most likely value.

As both passengers are in 1st class, and both paid the same fare (80), we can assign both of them the same embarked value of C.

```
titanicData$Embarked[c(62, 830)] <- 'C'
```

Let's check:

```
sum(is.na(titanicData$Embarked))
```

```
## [1] 0
```

Age

Ok so Embarked was only a handful of missing values. This is not the case for Age, where we have 19.8653199% missing values.

We have dealt with a numeric missing value before, we set them to be the mean. However, there are also other options.

Before we start, let's look at how to make a new Age vector, so that we can experiment a little with different options. Otherwise, if we modify *titanicData$Age* and make a mistake or are not happy with the results we have to read the data in again, and rerun at least some of the of the code above.

```
ImputedAgeMean <- titanicData$Age
```

Task Impute the missing values of ImputedAgeMean to be the mean age (make sure that you **don't** keep missing values in the computation of the mean!).

So a good question to be asking right now, is how do I know that my imputation is good or not? Providing a comprehensive answer to this question is not something for right now, so we'll address this qualitatively.

Task Use histograms to inspect the distribution of *titanicData$Age* compared to *ImputedAgeMean*. Would you happy with the result? If you are, you probably left the missing values in!!

Imputation via Linear Regression

Next, a fairly advanced technique (you probably won't see the theory behind this method for a few weeks yet); we're going to build a linear model to predict the age of every passenger for which we do not know the age.

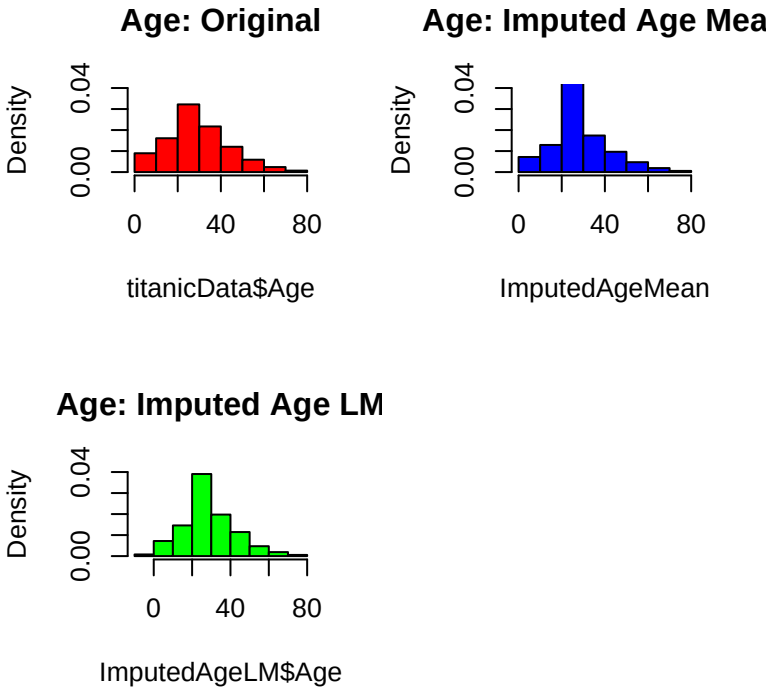
So a small bit of vocabulary, Age is referred to as a dependent variable (it's the one we want to predict, so it is *dependent* on a number of other variables). To predict Age, we will use a certain number of explanatory variables (sometimes also called independent, or predictor variables). This, by the way, is the same way we plotted the line of best fit in Lab 2. Essentially, we are doing the same thing now, but with more than two explanatory variables (or dimensions). Once we have derived this line (which is actually a hyperplane in this case) we use it to fill in the gaps (or missing values). We do this by selecting the point on the plane, i.e. Age, that corresponds to values in the explanatory variables used to build the linear model. If you are interested in a quick explanation of this works: click here.

After a long description, the code is quite short. We're chosing Pclass, Survived and SibSp as our explanatory variables. The tilda (~) indicates that Age should be modelled using the variables that follow. The + is used to itemise each of the explanatory variables.

```
idx_na <- is.na(titanicData$Age)
age_train <- titanicData[-idx_na, ]
age_test <- titanicData[idx_na, ]

ageModel <-lm(Age~Pclass + Survived + SibSp, data = age_train)
age_test$Age <- predict(ageModel, newdata = age_test)

ImputedAgeLM <- titanicData

ImputedAgeLM[ImputedAgeLM$PassengerId %in% age_test$PassengerId, "Age"] <- age_test$Age
```

So we could certainly tweak this approach (adding / removing explanatory variables), however, you **cannot** use categorical data in a linear regression. So let's have a look at how we are doing.

```
ImputedAgeMean[is.na(ImputedAgeMean)] <- mean(ImputedAgeMean, na.rm = T)

par(mfrow=c(2,2)) # allows us to put plots into a 2 x 2 grid -- makes it easier to compare
hist(titanicData$Age, freq=F, main='Age: Original', col='red', ylim=c(0,0.04))
hist(ImputedAgeMean, freq=F, main='Age: Imputed Age Mean', col='blue', ylim=c(0,0.04))
hist(ImputedAgeLM$Age, freq=F, main='Age: Imputed Age LM', col='green', ylim=c(0,0.04))
```



Imputation via Machine Learning

We have a space in our graph, so let's use a package (*mice*) that will do most of the work for us using machine learning. In this example, we are building a fairly complex machine learning model called a random forest to predict the missing values. Look at the mice help, to see the other methods available. The random forest is a pretty good go to machine learning model for prediction. You'll come back to this model when you start the data mining module.

```
library(mice)
```

```
## Warning: package 'mice' was built under R version 3.4.4
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'mice'
```

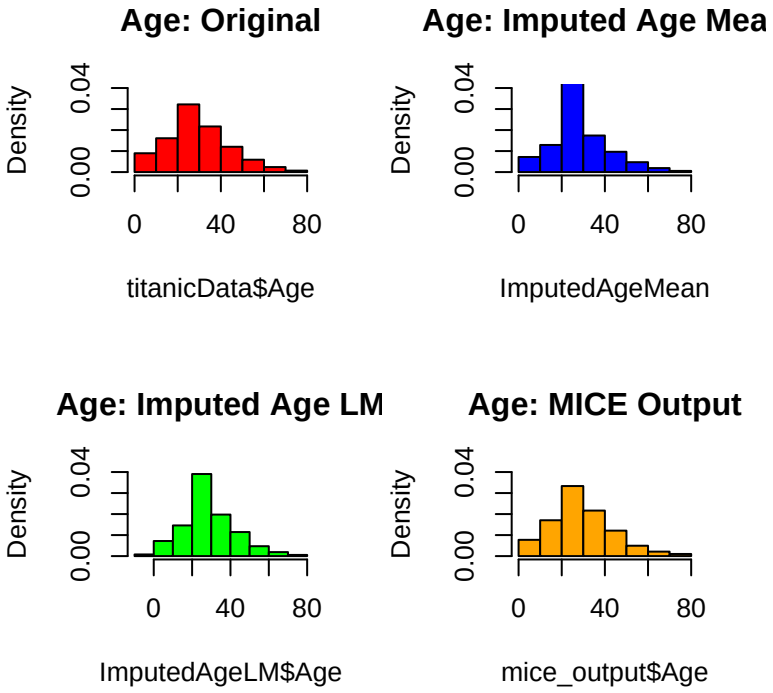
```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
# Perform mice imputation, excluding some variables that probably won't help:
mice_mod <- mice(titanicData[, !names(titanicData) %in%
     c('PassengerId','Name','Ticket','Cabin','Survived')], method='rf')
```

```
##
##  iter imp variable
##   1   1  Age
##   1   2  Age
##   1   3  Age
##   1   4  Age
```

```
##   1   5  Age
##   2   1  Age
##   2   2  Age
##   2   3  Age
##   2   4  Age
##   2   5  Age
##   3   1  Age
##   3   2  Age
##   3   3  Age
##   3   4  Age
##   3   5  Age
##   4   1  Age
##   4   2  Age
##   4   3  Age
##   4   4  Age
##   4   5  Age
##   5   1  Age
##   5   2  Age
##   5   3  Age
##   5   4  Age
##   5   5  Age
```

```r
# Complete the missing values
mice_output <- complete(mice_mod)

#Plot all the results to compare
par(mfrow=c(2,2)) # allows us to put plots into a 2 x 2 grid -- makes it easier to compare
hist(titanicData$Age, freq=F, main='Age: Original', col='red', ylim=c(0,0.04))
hist(ImputedAgeMean, freq=F, main='Age: Imputed Age Mean', col='blue', ylim=c(0,0.04))
hist(ImputedAgeLM$Age, freq=F, main='Age: Imputed Age LM', col='green', ylim=c(0,0.04))
hist(mice_output$Age, freq=F, main='Age: MICE Output',  col='orange', ylim=c(0,0.04))
```



So what we can see is that the mean is worst (in this case), the linear model improves on the results, but still isn't that great, and the mice model seems to be best.

Tasks

1. Replace mean with median and see what difference it makes. There are occasions where the median can be a better choice.
2. Try a few different combinations of explanatory variables for the LM (not categorical variables!) and see if you can improve it.

Cabin

Ok so Age was somewhat involved, but that's because we explored a few different ways. When it comes to Cabin, where we have 77.1043771% missing values, we have to rely on external sources like, for example: https://www.encyclopedia-titanica.org/cabins.html.

We'll come back to dealing with this kind of data later. For now, we're just going to remove the Cabin variable, as it's going to be quite complicated (at this stage) to deal with.

```
titanicData <- titanicData[, -11]
```

Simple Exploration of the Target Variable

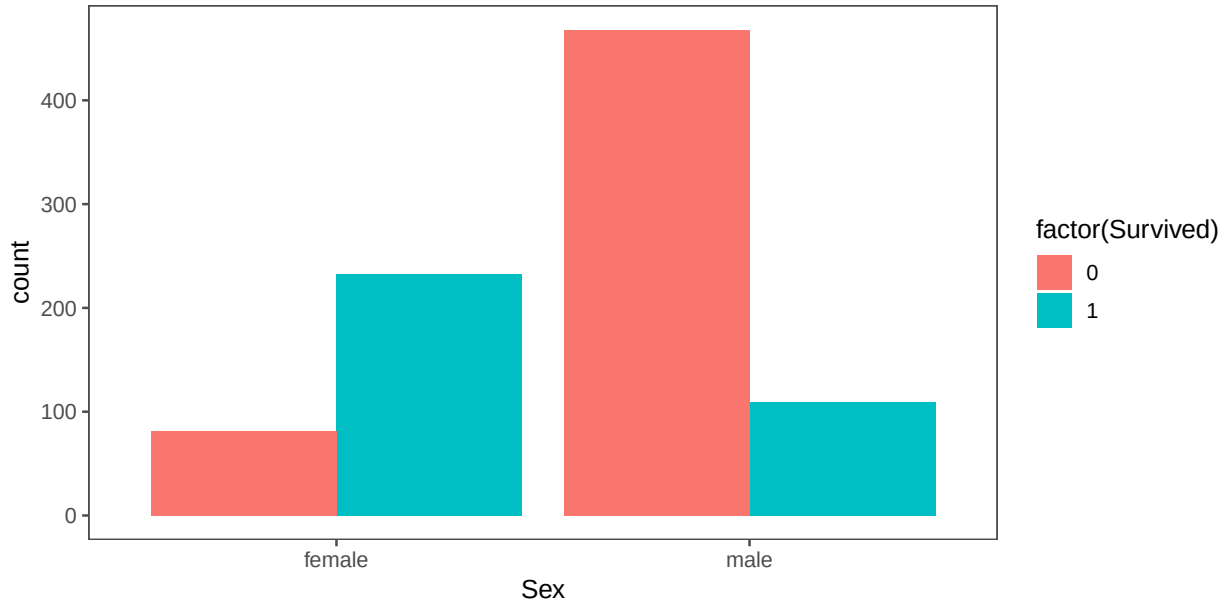
Whenever we analyse a dataset we should have some sort of objective in mind. For the titantic, people are usually interested in understanding why some people survived and others did not. So most studies will resolve around the *Survived* column.

Tasks

1. Make a barplot of the survived variable, to gauge the survival rate.
2. Make a barplot of the Sex variable, to see the distribution of gender.

Now let's combine both of these into one plot:

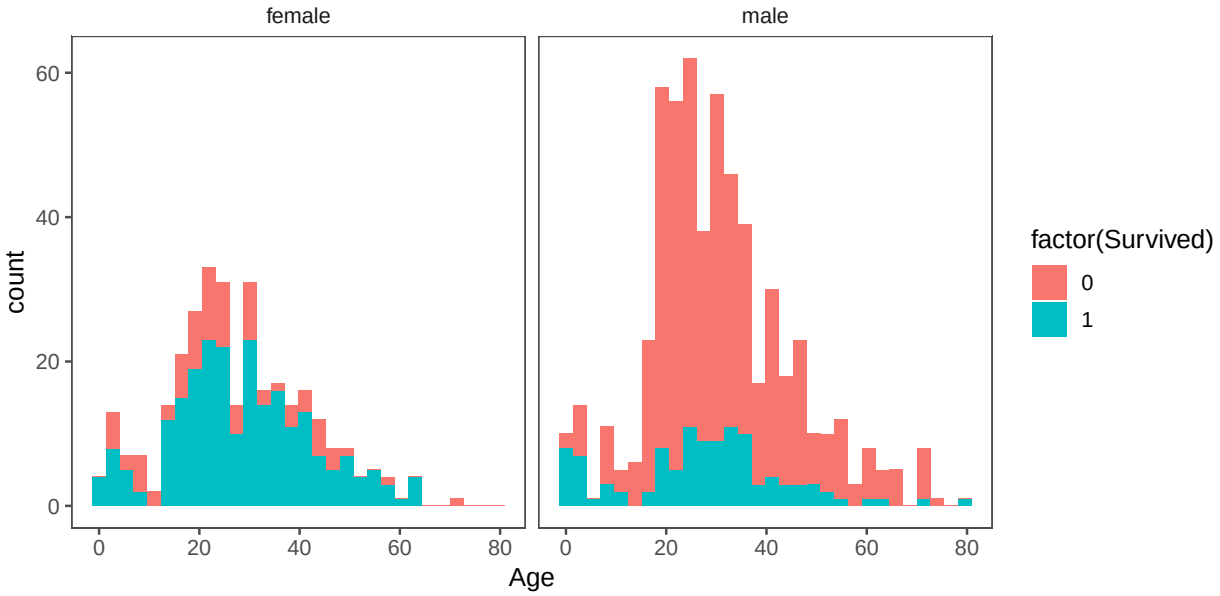
```
ggplot(titanicData, aes(x = Sex, fill = factor(Survived))) +
  geom_bar(stat='count', position='dodge') +
  labs(x = 'Sex') +
  theme_few()
```



So from this plot, we can see that there is a fair chance that if we know the gender of the passenger that we know if they survived or not. What about age? Well, we imputed Age already, so let's fill in the gaps from *mice*.

```
titanicData$Age <- mice_output$Age

ggplot(titanicData, aes(Age, fill = factor(Survived))) +
  geom_histogram() +
  # I include Sex since we know (a priori) it's a significant predictor
  facet_grid(.~Sex) +
  theme_few()
```

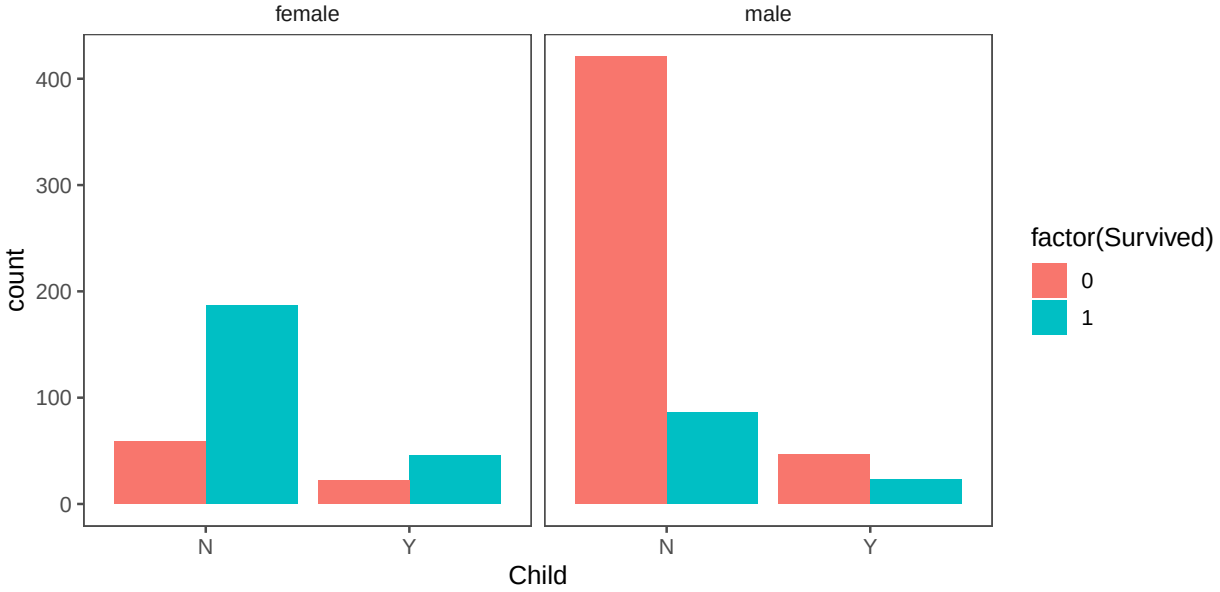
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Doesn't look as influential as knowing the Sex of a passenger.

Task

1. Try out different combinations of *Survived* with other explanatory variables. If you use a numerical explanatory variable, you may need to add *scale_x_continuous(breaks=c(1:N))* where N is the number of values you want on the x-axis.
2. Create a new factor called *Child* that has the value "N" if the passenger is greater than or equal to 18, and "Y" otherwise.
3. Create the plot below with the new *Child* column.



Having too much variation

Where missing values cause the problem of having gaps in the data, we can also have the opposite problem: having too much information. One such example is the passenger name. Intuitively, we may be tempted to get rid of this column as every name is unique:

```
length(unique(titanicData$Name))
```

```
## [1] 891
```

But, just for fun, let's explore what we can do with a little string manipulation. First, let's extract passenger titles:

```
# Grab title from passenger names
titanicData$Title <- gsub('(.*, )|(\\..*)', '', titanicData$Name)

# Show title counts by sex
table(titanicData$Sex, titanicData$Title)
```

```
##
##          Capt Col Don  Dr Jonkheer Lady Major Master Miss Mlle Mme  Mr Mrs
##   female    0   0   0   1        0    1     0      0  182    2   1   0 125
##   male      1   2   1   6        1    0     2     40    0    0   0 517   0
##
##           Ms Rev Sir the Countess
##   female   1   0   0            1
##   male     0   6   1            0
```

So what we've done here is use a regular expression: $(.,)|(\backslash\backslash..)$ to replace everything after a "." with an empty string, thus isolating the title. So what we see is that we arguably (and obviously this is dependent on the analysis task to hand!!) may have too much information about our passengers.

NOTE

What we are about to do will result in some information loss!! Basically what we are saying here, is that we are not interested in the difference between Rev, and Dr, but rather note them as infrequently observed titles that passengers have. There are many instances where we would **NOT** want to do this, but for now it's enough to show *how* this can be done.

```
# Titles with very low cell counts to be combined to "rare" level
rare_title <- c('Dona', 'Lady', 'the Countess','Capt', 'Col', 'Don',
                'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer')

# Also reassign mlle, ms, and mme accordingly
titanicData$Title[titanicData$Title == 'Mlle']        <- 'Miss'
titanicData$Title[titanicData$Title == 'Ms']          <- 'Miss'
titanicData$Title[titanicData$Title == 'Mme']         <- 'Mrs'
titanicData$Title[titanicData$Title %in% rare_title]  <- 'Rare Title'

# Show title counts by sex again
table(titanicData$Sex, titanicData$Title)
```

```
##
##          Master Miss  Mr Mrs Rare Title
##   female      0  185   0 126          3
```

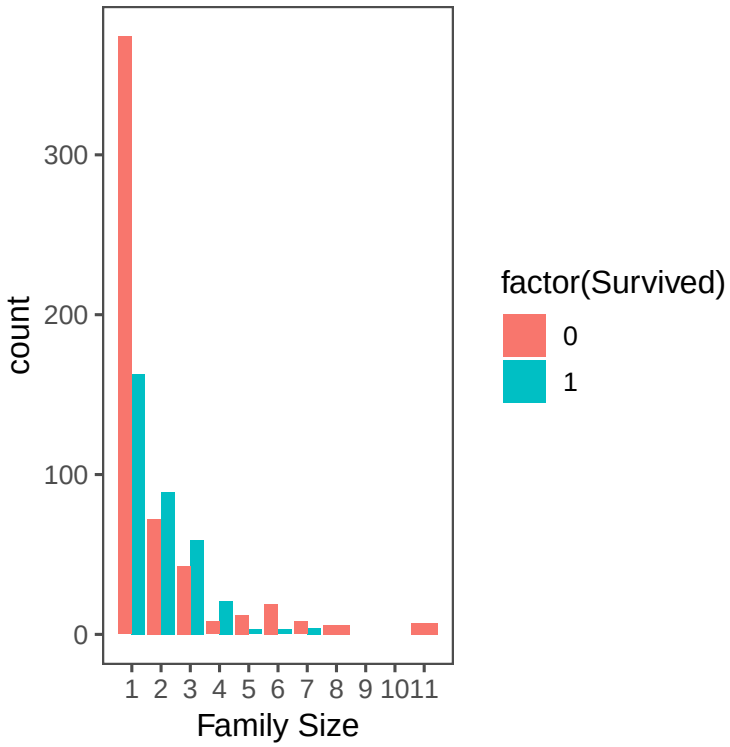
```
##   male       40    0 517   0         20
```

Ok, so we've harvested passenger titles, but what about the surnames? Would it not perhaps be useful to see if there is a relationship between being part of a family and suriving?

```
#engineer a surname attribute
titanicData$Name <- as.character(titanicData$Name)
#R assumed it to be a factor when we read in the data

titanicData$Surname <-  sapply(titanicData$Name,
  FUN=function(x) {strsplit(x, split='[,.]')[[1]][1]})

# Create a family size variable including the passenger themselves
titanicData$Fsize <- titanicData$SibSp + titanicData$Parch + 1

# Create a family variable
titanicData$Family <- paste(titanicData$Surname, titanicData$Fsize, sep='_')
```

Task Make the plot below, to help us understand if *Fsize* has any relationship with survival

---



So it would appear (we afterall haven't run any numbers to validate this conclusion) that small families of 2-4 had a better chance of surviving. We can collapse this variable into three levels which will be helpful since there are comparatively fewer large families. Let's create a discretized family size variable.

Task Make a variable called *FsizeD* in accordance to the following rules:

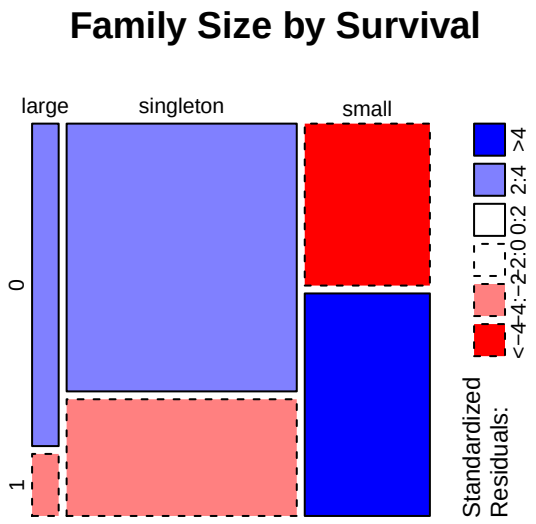
- Takes value of 'singleton' an individual is in a family of size one
- Takes value of 'small' if an individual is in a family of size between 2, 3, or 4
- Takes value of 'large' if an individual is in a family of 5 of greater

You should have a result like:

```
##
##     large singleton     small
##        62       537       292
```

Now let's explore our new variable:

```
# Show family size by survival using a mosaic plot
mosaicplot(table(titanicData$FsizeD, titanicData$Survived),
  main='Family Size by Survival', shade=TRUE)
```



The plot seems to validate our assuption, that there's a survival penalty among singletons and large families, but a benefit for passengers in small families.

Prediction

We can't spend all this time exploring this dataset without doing some sort of prediction. We are going to revisit the random forest model that we used in the Age imputation.

However, before we do, a few notes on this particular dataset: 1- the dataset is imbalanced: more people perished than survived; this means that notions of accuracy will be biased by this imbalance. 2- we know from earlier analysis (above) that if we know the Sex of a passenger, we have a good idea if they should have survived or not. This is quite convenient as it can act as a baseline performance benchmark for any predictive model we build. 3- We have a pretty small dataset; this fundamentally rules out the more advanced predictive

models like Artificial Neural Networks, Deep Learning, etc. 4- Our attributes are not that conveniently represented, e.g. Age is scewed, all of our numeric attributes have different value ranges (are not normalised), we have both categorical and numeric data. For these reasons (and several others that are similar) we need a pretty robust form of machine learning: the Random Forest is exactly that.

First a little bit of clean up:

```
titanicData$Pclass <- as.factor(titanicData$Pclass)
titanicData$Survived <- as.factor(titanicData$Survived)
titanicData$Title <- as.factor(titanicData$Title)
```

Let's subset our data into 2 portions: the first to build a model, and the second to test its performance.

```
index <- sample(1:nrow(titanicData), nrow(titanicData) * .75, replace=FALSE)

train <- titanicData[index, ]
test <- titanicData[-index, ]
```

So let's cut to the chase, and build our random forest, how this model works will be covered in other modules. Note that after Survived it's a tilda (~) not a minus (-)

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
rf_model <- randomForest(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked +
                           Title + FsizeD, data = train)
```

Let's quickly have a look at which variables the Random Forest has deemed important, and see if they align with our earlier assumptions:

```
varImpPlot(rf_model)
```

Accuracy tests to see how worse the model would perform without each variable, so a high decrease in accuracy would be expected for very predictive variables.Gini measures how pure the leaf nodes are. It tests to see the result if each variable is taken out: a high score means the variable is important.

So, let's use our test data – keep in mind that the random forest model hasn't seen this data before – so it's useful to gauge how well it performs.

```
survived <- test$Survived

#strip out the attributes that we don't want; this is strictly speaking
#needed, but it's useful to see how we could filter if we wanted to
wantedAttributes <- c('Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
                      'Fare', 'Embarked', 'Title', 'FsizeD')
test <- test[, names(test) %in% wantedAttributes]

str(test)
```

```
## 'data.frame':    223 obs. of  9 variables:
##  $ Pclass  : Factor w/ 3 levels "1","2","3": 3 3 3 3 3 3 2 3 3 3 ...
##  $ Sex     : Factor w/ 2 levels "female","male": 2 2 1 2 2 1 2 2 1 1 ...
##  $ Age     : num  70.5 2 4 39 2 17 35 22 18 14 ...
##  $ SibSp   : int  0 3 1 1 4 0 0 0 2 1 ...
##  $ Parch   : int  0 1 1 5 1 0 0 0 0 0 ...
##  $ Fare    : num  8.46 21.07 16.7 31.27 29.12 ...
##  $ Embarked: Factor w/ 3 levels "C","Q","S": 2 3 3 3 2 1 3 1 3 1 ...
##  $ Title   : Factor w/ 5 levels "Master","Miss",..: 3 1 2 3 1 4 3 3 2 2 ...
##  $ FsizeD  : Factor w/ 3 levels "large","singleton",..: 2 1 3 1 1 2 2 2 3 3 ...
```

```
forestPrediction <- predict(rf_model, test, type = "class")
forestMissclassificationRate <- mean(forestPrediction != survived)
print(paste('Accuracy randomForest: ',1-forestMissclassificationRate, ' %'))
```

```
## [1] "Accuracy randomForest:  0.847533632286996  %"
```

Just to close off, let's add just a little bit of context to this performance accuracy. We found out earlier that Sex is an important factor, so what if we always just *guessed* that if we have a female passenger they survived. How accurate would that be?

```
Gender <- rep(0,nrow(test))
Gender[test$Sex == 'female'] <- 1
GenderMissclassificationRate <- mean(Gender != survived)
print(paste('Accuracy all women survive: ',1-GenderMissclassificationRate, ' %'))
```

```
## [1] "Accuracy all women survive:  0.798206278026906  %"
```

Based on this result, how good does that 0.8475336% accuracy look now? Aside from whether you think this is/isn't a good result, it should be clear that reporting just an accuracy % can be misleading and underwhelming. Keep in mind when you do data mining and machine learning that you need to contextualise your % accuracy by considering how **hard** was is it to do well. We also didn't look at when we were wrong. To finish off, let's build a *co-incidence* matrix and explore when our random forest, and gender models were wrong:

```
table(survived, forestPrediction)
```

```
##          forestPrediction
## survived   0   1
##        0 130   9
##        1  25  59
```

```
table(survived, Gender)
```

```
##          Gender
## survived   0   1
##        0 123  16
##        1  29  55
```

To breifly explain, the diagonal (top left to bottom right) is important here:

vs.	Predicted: 0	Predicted: 1
Reality: 0	Correct	Incorrect
Reality: 1	Incorrext	Correct

So our random forest model is doing a little better than our simple hypothesis-based prediction model. However, what's important to take away, is that we can do pretty well just by visually exploring the dataset; provided (as was the case here) the dataset is not too complicated.