Into to R Lab 2: Interacting with data frames

Simon Caton

Basics of data types

In this previous lab, we introduced at different types of *objects*:

- Vectors
- Matricies
- Lists
- Factors
- Data Frames, and
- Functions

and different kinds of data:

- Numeric data: numerical representations of attributes
- Categorical data: as the name implies, data grouped into some sort of category or multiple categories

R can switch or cast data between it's different data types:

- numeric
- logical – boolean, i.e. TRUE or FALSE
- character – strings
- factor – categoricals
- matrix
- array
- data.frame

A special attribute known as the class of the object is used to allow for an object-oriented style of programming in R, and determines how R handles an object. The simplest example here is printing the object to the command line; a data.frame will be handled differently to a numeric. Similar to OO paradigms, R supports getters and setters of attributes of an object. E.g.

```
x <- c(1:3, 7, 8:10)
attributes(x)
```

```
## NULL
```

to get all attributes. To determine the class of an object, we use the *class* function:

```
class(x)
```

```
## [1] "numeric"
```

So x is an integer object (well a vector of integers) with the values 1, 2, 3, 7, 8, 9, 10. So in this case, it has no attributes.

Example:

```
z <- 0:9
class(z)
```

```
## [1] "integer"
```

```
digits <- as.character(z)
class(digits)
```

```
## [1] "character"
```

```
d <- as.integer(digits)
class(d)
```

```
## [1] "integer"
```

Here we cast *z* (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to characters instantiating *digits* as a new vector of characters, then cast *digits* back to a new vector *d* of integers. R has many of these "*as.*" functions for casting between the its various data types:

- as.integer
- as.factor
- as.numeric
- as.character
- as.matrix
- as.array
- as.data.frame
- …

The data.frame object

It's also useful to think about objects as storing multiple instances of one or more data types. Let's say we had a class of 3 students: Amy, Bill, and Carl, we can store them in a vector of strings:

```
name <- c("Amy", "Bill", "Carl")
```

Let's say that each student undertakes 2 modules (DAD and BDA) and we want to represent their scores:

```
DAD <- c(80, 65, 50)
BDA <- c(70, 50, 80)
```

Now let's capture some demographic information about our students:

```
gender <- as.factor(c("F", "M", "M"))
nationality <- as.factor(c("IRL", "UK", "IRL"))
age <- c(20, 21, 22)
```

Now we have 6 vectors each with three components (one for each student). It would be useful to combine all these observations into a single object, this is where the idea of a data.frame comes in.

```
student.df <- data.frame(name, age, gender, nationality, DAD, BDA)
```

If we quickly revisit the *attributes* of the data frame that we have just built, we can see in this instance that we get a slightly more interesting result

```
attributes(student.df)
```

```
## $names
## [1] "name"        "age"         "gender"      "nationality" "DAD"
## [6] "BDA"
##
## $row.names
## [1] 1 2 3
##
## $class
## [1] "data.frame"
```

Task Run the functions: print, str, and summary on the student.df data.frame

---

We can also use all of the indexing mechanisms we used before with square brackets, e.g:

```
student.df['gender']
```

```
##   gender
## 1      F
## 2      M
## 3      M
```

will access the gender column.

but we can also address the data.frame more directly:

```
student.df$gender
```

```
## [1] F M M
## Levels: F M
```

or if we want to be really precise:

```
student.df$gender[2]
```

we can also execute functions on a data.frame

```
mean(student.df$BDA)
```

```
## [1] 66.66667
```

but only if they make sense! For example, *mean(student.df$gender)* will throw an error.

We can also introduce (or engineer) new features into our data. For example, an average grade:

```
student.df$average <- (student.df$BDA + student.df$DAD)/2
student.df
```

```
##   name age gender nationality DAD BDA average
## 1  Amy  20      F         IRL  80  70    75.0
## 2 Bill  21      M          UK  65  50    57.5
## 3 Carl  22      M         IRL  50  80    65.0
```

Notice here the power of R. We have computed the average for every invidividual without looping through the data.frame. In this example, that was was possible, however sometimes we will need to loop through the data.frame.

To have a slightly more involved task, let's extend our data.frame a little. First, we need to clean up our data.frame, as we have an error in it. Let's look at the structure of the data.frame:

```
str(student.df)
```

```
## 'data.frame':    3 obs. of  7 variables:
##  $ name       : Factor w/ 3 levels "Amy","Bill","Carl": 1 2 3
##  $ age        : num  20 21 22
##  $ gender     : Factor w/ 2 levels "F","M": 1 2 2
##  $ nationality: Factor w/ 2 levels "IRL","UK": 1 2 1
##  $ DAD        : num  80 65 50
##  $ BDA        : num  70 50 80
##  $ average    : num  75 57.5 65
```

R has tried to "help" us and encoded name as a factor, we don't want this as it will prevent us from adding new names to our data.frame

---

Task Complete the following to cast student.df$name to a character vector

```
student.df$name <- as.?????(student.df$name)
```

Now we can add more rows:

```
student.df <- rbind(student.df, c("Dennis", 23, "M", "UK", 55, 70))
```

Note that if you add a nationality other than UK or IRL, or if you didn't cast your factor appropriately you will get an error something along the lines of **"invalid factor level, NA generated"**. Hence, we need to relax the encoding of nationality (currently a factor) as well.

NOTE You may argue at this point (and you would be right!) that we shouldn't have made nationality a factor if we knew that we would be adding to it. Thinking back to KDD and CRISP-DM this is exactly why it is so important to explore and understand the data, as well as the (business) requirements of the analysis/analyses we will conduct. Similiarly, it's also why we need to plan and sketch out what analysis/analyses we will conduct in order to ensure we prepare our data appropriately for our analysis/analyses.

Tasks

1. Cast nationality to a character vector
2. Add another 5-10 students, not all with the same nationality or gender!
3. Rebuild the nationality factor (cast the nationality vector back to a factor)

Now, let's explore the power of factors a little, and try to understand why categorical data can be useful. Let's be policitically incorrect, and explore the average score for each nationality. First let's see what nationalities we have (you should have more if you did task 2 above properly):

```
levels(student.df$nationality)
```

```
## [1] "IRL" "UK"
```

So, to work out which country according to our non-exhaustive and probably not representative sample of students appears to be better at statistics:

```
averages <- tapply(student.df$BDA, student.df$nationality, mean)
averages
```

```
## IRL  UK
##  75  50
```

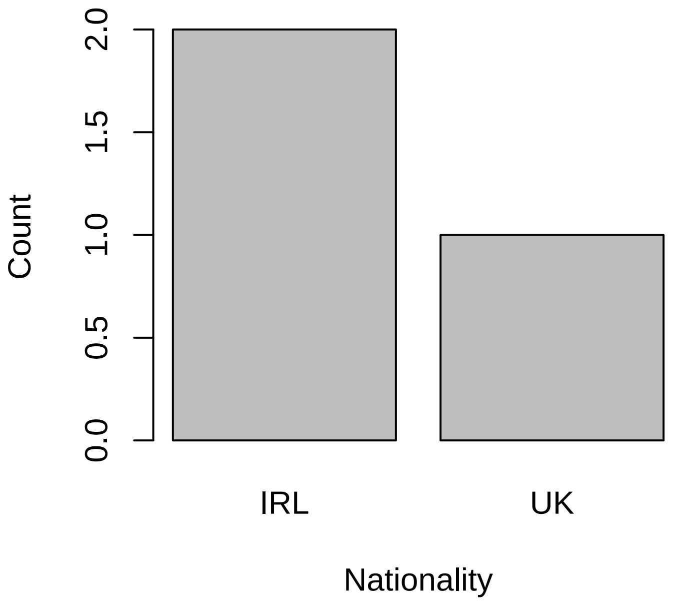
However, an average may be misleading here. According to the table above, we could interpret Irish students to be better at statistics. Perhaps we should inspect our data a little better to understand the validity of this observation.

```
table(student.df$nationality)
```

```
##
## IRL  UK
##   2   1
```

Or graphically:

```r
barplot(table(student.df$nationality), xlab = "Nationality", ylab="Count")
```



I have here only 3 observations, so we need to be careful about how we interpret results like these in the presence of a small number of samples.

Tasks

1. compute the min, max, and standard deviation for BDA, and DAD
2. build a data.frame of the results from 1. one row for BDA and one for DAD with min, max, mean and sd as the columns

---

Built in R datasets

R comes with many datasets built into it, most of these are used in teaching environments and are reasonably clean, and easy to use. The *mtcars* dataset is one such example. Let's have a look at it and explore this data set a little.

```r
data(mtcars) #loads the built-in dataset
str(mtcars) # shows us the structure of the dataset
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```r
summary(mtcars) # computes some basic descriptive information of the dataset
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

So the encoding of this data may be sub-optimal. Run

```r
?mtcars
```

to access the meta data for this dataset.

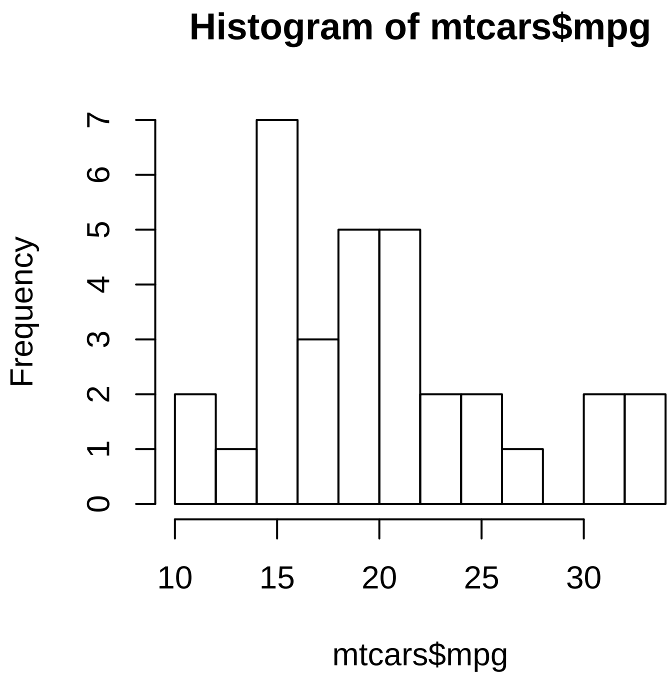
Let's fix a few things:

```r
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$am <- factor(mtcars$am, labels=c("Automatic", "Manual"), levels=c(0,1))
```

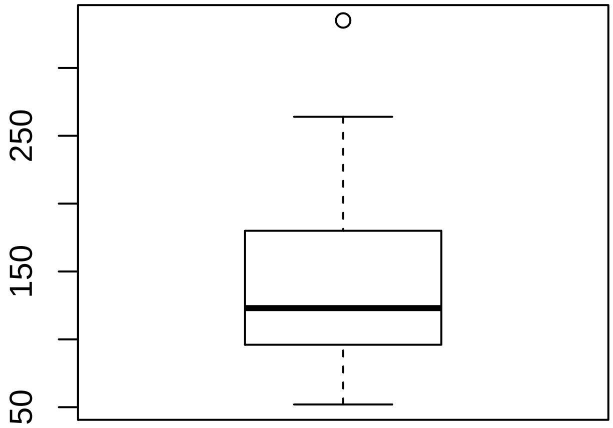
Task Rerun str and summary to see the changes we made

Now, some basic data exploration tasks:

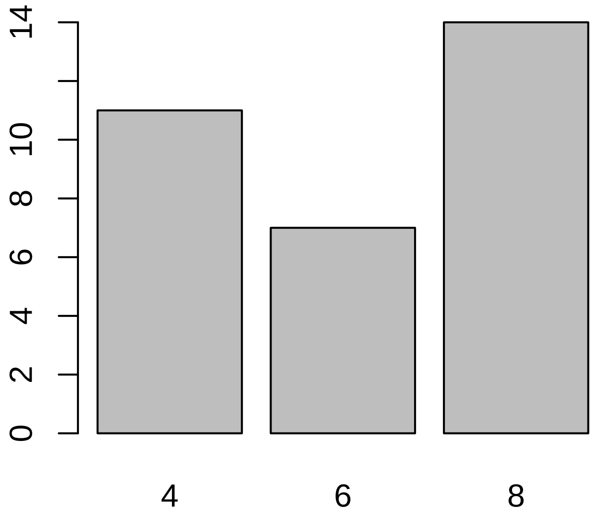
```r
hist(mtcars$mpg, breaks = 10) #breaks controls how granular the histogram will be
```



```r
boxplot(mtcars$hp)
```



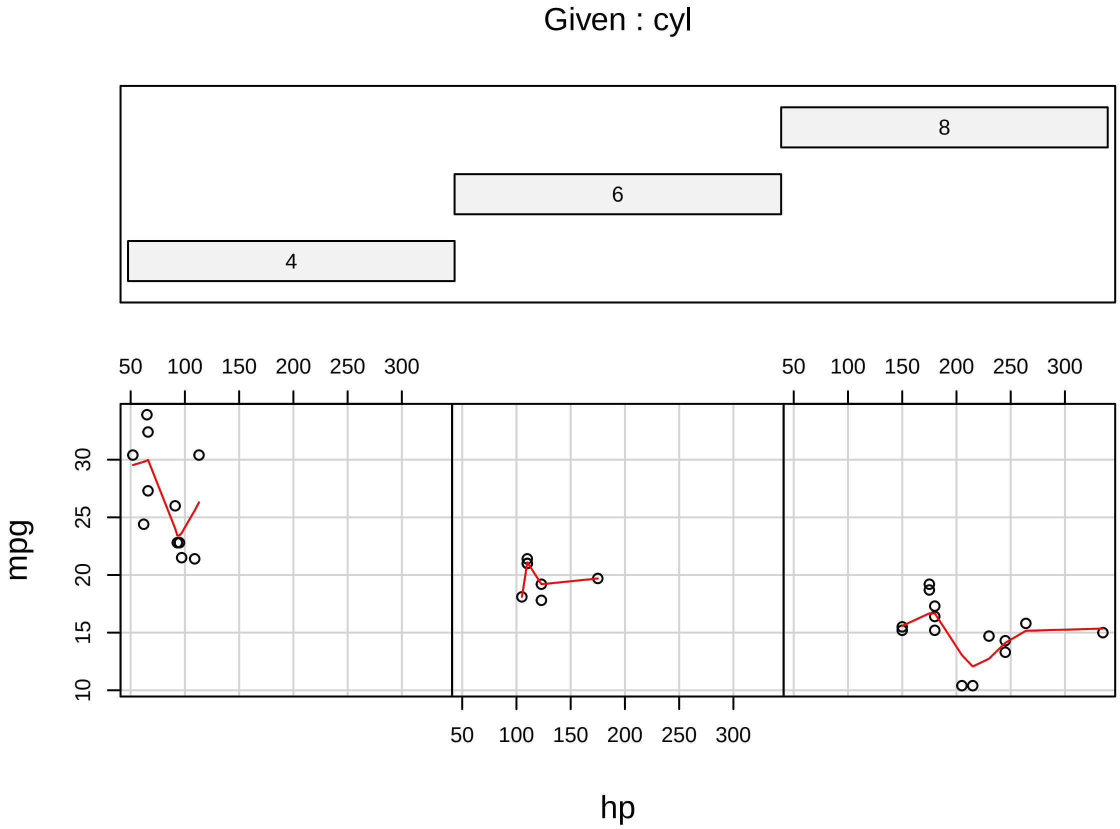
```r
barplot(table(mtcars$cyl))
```



See https://en.wikipedia.org/wiki/Box_plot for an overview of the box plot.

These methods of looking at data are fine, but don't tell us too much. Things start to get interesting when we view relationships across all these columns at once.

```
coplot(mpg ~ hp | cyl, data = mtcars, panel = panel.smooth, rows = 1)
```



Exercises

We'll come back to graphically representing data next time. For now though here some exercises for the mtcars dataset. You may need to refer to Lab 1 for some of these questions.

1. Are there more automatic or manual cars?
2. Which car is heaviest, and which is fastest?
3. Do automatic or manual cars have on average a better mpg?
4. How many cars have above average hp?
5. Of the cars that have above average hp, how many have 6 cylinders?
6. Of the cars that have above average hp, and 6 cylinders, how many are automatic?

Questions covering material we haven't covered yet (in case you finish early)

7. Make a boxplot of mpg split by no. of cylinders
8. In the above boxplot are there outliers?
9. Take a random 50% sample of the dataset, and rerun questions above to see what changes
10. Split the dataset in 2, one half containing only automatics, and one half containing only manual transmissions
11. For each half, plot mpg against hp using the *plot* function
12. Add an "abline" that "fits" a linear model between the two variables in the plots you just drew

Solutions will be available next week. However, I expect you to have completed the first 6, and attempted at least 1 of questions 7-12.