ETC3250: Lab 1 extra

Business Analytics 25/26 July 2016

The aim of this first lab session is to make sure everyone is familiar with using R through the RStudio interface.

1. R is a language for data analysis, which is why it is the top software for data science today. However at its basic level, R can be used as a (scientific) calculator. Click on the console tab and figure out what each of the following is doing.

   ```
   (100+2)/3
   5*10^2
   1/0
   0/0
   (0i-9)^(1/2)
   sqrt(2 * max(-10, 0.2, 4.5))
   x <- sqrt(2 * max(-10, 0.2, 4.5)) + 100
   x
   log(100)
   log(100, base=10)
   ```

   Typing the name of any object will cause it be printed to the console.

2. Check that these are equivalent

   ```
   y = 4
   y <- 4
   4 -> y
   ```

3. R is great for matrix calculations:

   ```
   X <- matrix(c(3,4,5,2), nrow=2, ncol=2)
   t(X)
   Xinv <- solve(X)
   X %*% Xinv

   A <- matrix(rnorm(200), nrow=5, ncol=40)
   B <- A %*% t(A)
   dim(B)
   diag(B)
   eigen(B)
   svd(A)$d^2
   eigen(B)$values
   ```

   Why are the last two lines the same? [Don't worry if you don't know!]

4. Find the help page for the `mean` command, either from the help menu, or by typing one of these:

   ```
   help(mean)
   ?mean
   ```

5. Find the example in the help page for the `mean` command.

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Most help pages have examples at the bottom.

What does the last line do?

6. Visualizing your data is one of the essential elements of data analysis. We are going to primarily use the `ggplot2` package for making data plots. The reason is that it provides elegant graphics in a concise conceptual framework. We will learn more about this later in the semester, but let's get started using the quick plot function `qplot`:

```
library(ggplot2)
df <- data.frame(x=x, y=c(rep("yes", 7), rep("no", 5)))
df
qplot(x, data=df)
qplot(x, data=df, binwidth=5)
qplot(y, x, data=df,  geom="boxplot", xlab="")
qplot(factor(0), x, data=df,  geom="boxplot", xlab="")
```

Different R functions can require different data input types. Many of the original functions operate on matrices, but more recently written functions require data frames as input. The package `ggplot2` likes to have data frames.

7. Rather than type all your commands in the console panel, it is convenient to open an R script file and type them there. Then highlight and click "Run". That way you have a record of what you have done, and can go back and make changes as required.

   Set up a new R script file called "Lab 1".

   You can add comments to your R file by beginning the line with `#`.

8. The function `rnorm` generates random numbers from a standard normal distribution. Produce a histogram of 200 random numbers from N(0,1)

```
# 200 random numbers from N(0,1)
qplot(rnorm(200))
qplot(factor(0), rnorm(200), geom="boxplot", xlab="")
```

   Modify these commands so that the boxplot uses the same numbers as the histogram. (Hint: save the output of `rnorm` for re-use.) Notice in these commands that the `qplot` will also take the output of `rnorm` directly, which is a numeric vector.

9. Let's look at some data sets from the ISLR package. First we need to make sure it is installed.

```
library(ISLR)
```

   If that returns an error, then click on the "Packages" tab, click "Install" and proceed to install the ISLR package. It will download from the "CRAN" repository.

   Alternatively, you can use `install.packages("ISLR")` at the command line.

   A package only needs to be installed once, but you have to load it via the `library` command in each session.

   Once the package is installed, try again with

```
library(ISLR)
```

10. Then look at the `OJ` data set:

```
help(OJ)
head(OJ)
View(OJ)
summary(OJ)
OJ[,"PriceCH"]
```

Can you figure out what the square brackets mean in the output from the last command?

11. Now lets make some plots of the data

```
saleprice <- data.frame(SalePrice=c(OJ$SalePriceMM,OJ$SalePriceCH),
                type=c(rep("MM",nrow(OJ)),rep("CH",nrow(OJ))))
qplot(type, SalePrice, data=saleprice, geom="boxplot")
qplot(Purchase, PriceDiff, data=OJ, geom="boxplot")
qplot(PriceCH, SalePriceCH, data=OJ)
qplot(PriceCH, SalePriceCH, data=OJ, position="jitter") +
  geom_abline(intercept=0,slope=1)

qplot(WeekofPurchase, SalePriceCH, data=OJ, position="jitter")
```

Make sure you understand what is being plotted in each case, and what the graphs are telling you about the data.

12. Tabulating variables:

```
table(OJ$StoreID)
table(OJ$Purchase, OJ$SpecialCH)
```

What do these tables tell you?

13. The `summary` command can be applied to almost anything to get a summary of the object. Try it on some other data sets in the `ISLR` package. Note that the `summary` is just what R thinks should be the summary, and it may not always be the best summary. If someone asks you to "summarise" this data set, you may need to think about what is important and use different functions that are appropriate for the situation.

14. Now we will read some data from a file. Download the PISA data set from `dicook.github.io/BusinessAnalyticsCourse/` and save it locally. You can read it into R using

```
pisa <- read.csv("PISA-oz.csv", stringsAsFactors=FALSE)
```

You will need to set your working directory to wherever you saved the file, or save the data into your current working directory, else you will produce an error. Either use `setwd` or the menu "Session" / "Choose Working Directory". To find out where R is currently working use `getwd()`.

Or you can also directly read the data from the web using the web address of the data as the filename. Or alternatively, you can get R to open a file browser and navigate yourself to the location of the data file:

```
pisa <- read.csv(file.choose(), stringsAsFactors=FALSE)
```

What does the `stringsAsFactors` do?

The PISA data set contains results from Australia for the "Programme for International Student Assessment" in 2012. The survey tests the skills and knowledge of 15-year-old students. Data were collected on approximately 14,500 Australian students between late July and early September 2012.

15. Check the data:

```
dim(pisa)
colnames(pisa)
View(pisa)
str(pisa)
head(pisa)
```

16. Which columns of pisa are numeric?

    Which columns are character?

17. How many different schools were sampled (according to the variable `SCHOOLID`)?

    There are several ways of answering this question. First use the `table` command. Then try using a combination of `length` and `unique`.

18. Look at the distribution of birth months amongst the Australian students:

```
qplot(factor(ST03Q01, labels=month.abb), data=pisa, xlab="Month")
```

    Can you explain the variation? Why are February and May the smallest?

19. Perhaps we should adjust for month length:

```
monthdays <- c(31,28,31,30,31,30,31,31,30,31,30,31)
monthtot <- table(pisa$ST03Q01)
qplot(factor(1:12, ordered=TRUE, labels=month.abb),
      c(monthtot)/monthdays, xlab="Month", ylab="Birthdays per day",
      geom="bar", stat="identity")
```

20. Check if the differences are statistically significant after adjusting for month length:

```
chisq.test(monthtot, p=monthdays/365)
```

    Why is it so?

21. What if we split by sex and turn into percentages:

```
male <- table(subset(pisa, ST04Q01=="Male")$ST03Q01)/monthdays
male <- male / sum(male) * 100
female <- table(subset(pisa, ST04Q01!="Male")$ST03Q01)/monthdays
female <- female / sum(female) * 100
sextot <- data.frame(birthdays=c(male,female),
            sex=c(rep("Male",12),rep("Female",12)),
            month=factor(rep(month.abb,2),levels=month.abb,ordered=TRUE))
qplot(month, birthdays, data=sextot, geom="bar", fill=sex,
      stat="identity", position=position_dodge(), ylab="Birthdays per day") +
  geom_hline(yintercept=100/12, slope=0, lty=2)
```

The largest deviation from what you would expect is for males born in May. Why?

22. It is easy to create your own data:

```
mynumbers <- 5:12
```

Names can be almost anything, except for special characters `^, !, $, @, +, -, /, *`. It is good practice to name your objects with some meaning for what they contain, be reasonably short (less typing). They should not be the same as common R functions; for example, don't use `data` because it is also used to load stored data from packages, or `c` because this is an R function that allows you to collect a bunch of objects together. You won't get errors by using these names but you may get confused when you come back and look at your code later.

23. Objects can be of different types. The object 'mynumbers' is a vector of numbers. Numbers can be various types also: integer or double.

```
typeof(mynumbers)
is.numeric(mynumbers)
is.vector(mynumbers)
length(mynumbers)
```

24. Other common types of objects for data analysis are characters, logicals, factors, dates. Factors store categorical data. Dates have a special format that enables it to be treated similarly to how we use dates in real life.

```
mytext <- c("hello", "class")
length(mytext)
mylogic <- c(TRUE, FALSE, TRUE, TRUE)
gender <- factor(c("male", "female", "female", "female", "male"))
levels(gender)
summary(gender)
```

25. One of the powerful aspects of R is to build on the reproducibility. If you are going to do the same analysis over and over again, compile these operations into a function that you can then apply to different data sets.

```
y <- c(1,2,3,4,5,6)

average = function(x)
{
  return(sum(x)/length(x))
}

ybar <- average(y)
```

Try your function on other data.

26. Now write a function to compute the mode of some data:

```
mymode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
mymode(c(rep("A", 5), rep("B", 3)))
```

27. Modify your function to have an optional label:

```
mymode <- function(x, labelit="m") {
  ux <- unique(x)
  ux <- ux[which.max(tabulate(match(x, ux)))]
  names(ux) <- labelit
  ux
}
```

- `x` is an input to the function that is essential to provide each time.
- `label` is an optional addition that takes default value `"m"`, but the user could change it to something else, simply by changing it in the function on the fly.

```
mymode(x=c(rep("A", 5), rep("B", 3)))
mymode(x=c(rep("A", 5), rep("B", 3)), labelit="mode")
```

28. Write an R function `stats` to compute the mean, min, max, and the deciles, from a vector of data. It should begin like this:

```
stats <- function(y)
{
  meany <- mean(y)
```

You will need to search the R help facilities to find functions to compute each of the statistics. The function should return the statistics as a single vector in numerical order with appropriate names for the elements. Your function should be robust to missing values (i.e., the statistics should be computed on the non-missing values).