Lecture 1: Introduction

Thomas Chadefaux

PO7001: Quantitative Methods I

Who?

Learning Outcomes

- Understand data concepts and basic descriptive quantitative analysis tools
- Work with real datasets to perform basic quantitative analyses
- Graph data effectively for presentation and analysis
- Recognize and understand the basics of the linear regression model
- Use the R statistical software package for analyzing and graphing data
- Understand sufficient theoretical and practical material to build on in a second, more advanced quantitative methods course

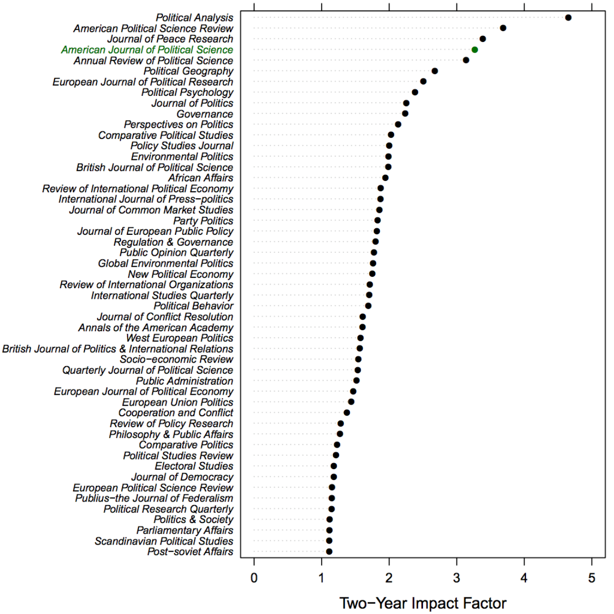
Why should I care?

Table 5b.

Distribution of Papers Accepted by Approach(%)

YEAR	FORMAL	QUANTITATIVE	FORMAL AND QUANTITATIVE	SMALL-N	INTERPRETIVE/ CONCEPTUAL	QUALITATIVE AND/OR EMPIRI- CAL	OTHER
	APPROACH						
2012–13	8.5	54	4	0	27.5	6	0
2011–12	12	48	14	2	19	5	0
2010–11	11	65	8	0	16	NA	0

Why should I care?



Software

- R. VERY powerful and free!
- Get it from http://www.r-project.org

Why R?

- Powerful
- Versatile
- Gives you a lot more freedom

Figure 1c. Analytics job trends for R and SPSS. Note that the legend labels at the top of the graph are truncated due to the very long size of the query.

Grading

- Problem Sets: 50%
  - 5 problem sets, each worth 10%
  - Submit as a single PDF on turnitin (see syllabus)
  - Scan if needed
- Exam: 50%. Specifics TBD depending on HSE guidelines

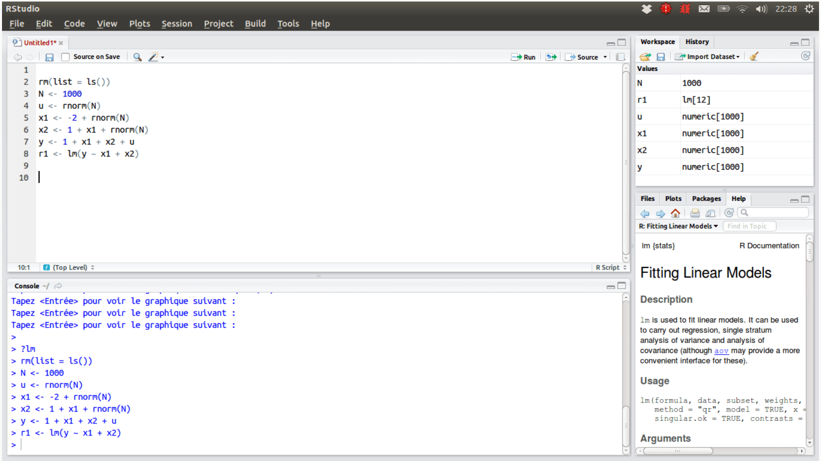
Texts for this course

- Verzani, John. 2005. Using R for introductory statistics. Boca Raton, FL: Chapman & Hall/CRC.
- Moore, David S., George P. McCabe and Bruce A. Craig. 2012. Introduction to the practice of statistics. 7th international edition ed. New York: W.H. Freeman.

R Crash Course

What you need

- R: download at www.r-project.org, install
- RStudio: go to www.rstudio.com and install



R Crash Course: Numbers

Add numbers

```r
1 + 2
## [1] 3
```

Assign numbers to a variable

```r
x <- 1 + 2
```

Note that x was not printed. To see the contents of x, just type:

```r
x
## [1] 3
```

How would you store the result of $3x$ in a variable called *myprecious*?

```r
myprecious <- x*3
```

Vectors

What if we want to create a larger collection of numbers? Use the c() function:

```r
z <- c(1, 2, 3, 5, 6)
```

Or, for a sequence of integers, use ':'

```r
z <- 1:6
```

Now let's add the number 7 to that vector, without retyping the whole sequence

```r
z <- c(z, 7)
```

Notice that I have now overwritten z:

```r
z
## [1] 1 2 3 4 5 6 7
```

Exercise

Combine two sequences into a variable called ‘mycat’: the integers from 1 to 15, and the number 100

operations

You can conduct all kinds of operations on z:

```
z*2
## [1]  2  4  6  8 10 12 14
(z+100)/6
## [1] 16.83333 17.00000 17.16667 17.33333 17.50000 17.6666
```

Take the square root of (z +1), and then log it:

```
z2 <- sqrt(z + 1)
z3 <- log(z2)
z3
## [1] 0.3465736 0.5493061 0.6931472 0.8047190 0.8958797 0
```

Or you can do it in one go:

```
z3 <- log(sqrt(c(1,2,3,4,5,6,7)*2))
z3
```

Recycling

What happens if you add c(1,2,3,4) to c(1,2)?

```
c(1,2,3,4) + c(1,2)
## [1] 2 4 4 6
```

The shorter vector gets 'recycled'. However, the code below yields a warning:

```
c(1,2,3,4) + c(1,2,3)
## Warning in c(1, 2, 3, 4) + c(1, 2, 3): longer object le
## shorter object length
## [1] 2 4 6 5
```

Creating random numbers

There are many ways to do this and we'll cover this later, but for now we'll draw a number from a uniform distribution (i.e., each number is as likely to be picked as any other) between 0 and 100:

```
runif(n = 1,    #we want one number
      min = 0,  # between 0
      max = 100) # and 100
## [1] 41.5198
```

and an integer between 0 and 100:

```
sample(x = 0:100,# draw a number from this list
       size = 1) # we only want one number
## [1] 8
```

Exercise:

- Draw 10 numbers from a uniform distribution, and save them in a variable called ‘mrn’
- Sample one number from mrn

Creating data

The workhorse of data analysis in R is the data frame. To create a data frame, for example:

```
localdat <- data.frame(ID = c(1,2),
                       gender = c('male', 'female'),
                       income = c(50000, 60000))
```

You can put complex statements inside that data.frame:

```
localdat <- data.frame(age = round(runif(4,0,100)),
                       income=round(runif(4, 0, 100000)),
                       gender=c('male', 'female'))
```

Importing and exporting data

Instead of creating them, you typically import data frames from your local file system or from the web. head() lets you take a look at the first few rows.

```
localdat <- read.csv('mydata.csv')
head(localdat, 4)
##   ID age income gender
## 1  1   4  49672   male
## 2  2  56  20300 female
## 3  3  56  73388   male
## 4  4  99  33374 female
```

Export it to your local file system:

```
write.csv(localdat, file = 'newfile.csv')
```

Extracting data

Localdat is a data frame. Loosely, a table with data. To access its information, we need to ask R for a row and column number, in that order. For example, to ask for row 1 and column 2, we would write:

```
localdat[1, 2]
## [1] 4
```

Or perhaps we want to see all columns associated with row 1, in which case we leave the column indicator empty, and similarly it we want all rows associated with a column:

```
localdat[1, ]
##   ID age income gender
## 1  1   4  49672   male
localdat[, 2]
## [1]  4 56 56 99 22 22 29 14 91
```

Extracting data (Cont'd)

We can also ask for a specific variable by name in three ways, though the first one is the most common

```
localdat$age
## [1]  4 56 56 99 22 22 29 14 91
localdat[, 'age']
## [1]  4 56 56 99 22 22 29 14 91
with(localdat, age)
## [1]  4 56 56 99 22 22 29 14 91
```

Extracting data (Cont'd)

We might have more specific requests. E.g., we want to see all males younger than 50 with income of less than 20000. In this case there is only one, with ID 7:

```
localdat[localdat$age < 50
         & localdat$gender =='male'
         & localdat$income < 20000, ]
##   ID age income gender
## 7  7  29  13124   male
```

Exercise

Find the ID of all females in localdat?

Summarizing data

What is the mean income?

```r
mean(localdat$income)
## [1] 45678.89
```

What is the maximum age?

```r
max(localdat$age)
## [1] 99
```

Also useful:

```r
summary(localdat$income)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4893   20300   49672   45679   73388   84749
```

Exercise

What is the average income of women under the age of 50?

Exercise

What is the average income of women under the age of 50?

```r
mean(localdat$income[localdat$gender == 'female' &
                        localdat$age < 50])
## [1] 65805
```

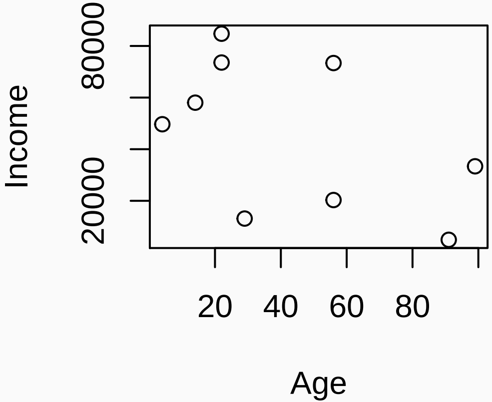
OR

```r
with(localdat, mean(income[gender == 'female' &
                              age < 50]))
## [1] 65805
```

XY plots

To plot x, y:

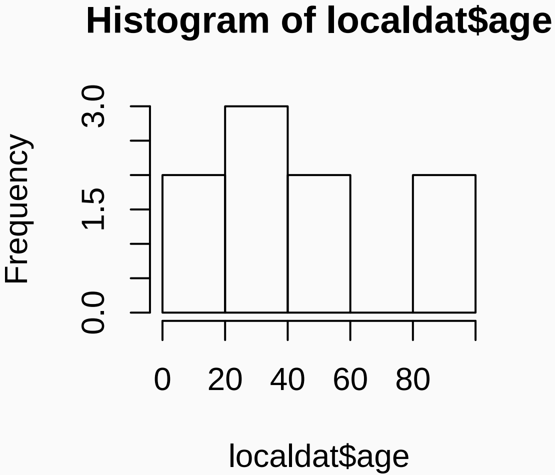
```r
x <- localdat$age
y <- localdat$income
plot(x, y, xlab='Age', ylab='Income')
```



Histogram

To plot x, y:

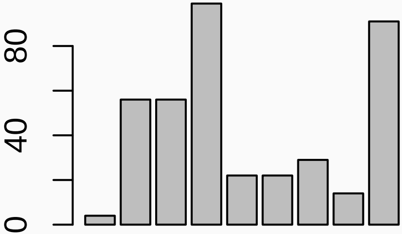
```r
hist(localdat$age)
```



Barplot

To plot x, y:

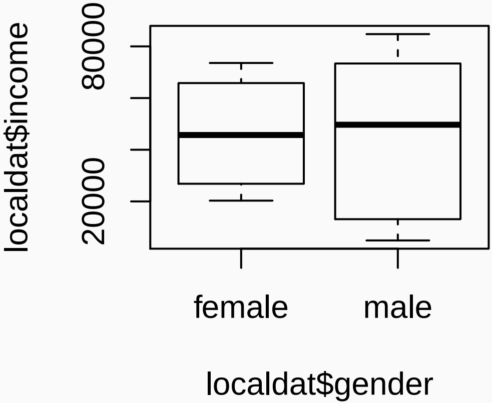
```
barplot(localdat$age)
```



Boxplot

Plot income by gender:

```
plot(localdat$income ~ localdat$gender)
```



if statements

Often you will want to check if something is equal, greater, smaller than something else. "if" will tell you if a certain statement is true or not.

```
if(1==2) {print('We need to rethink math')}
if(1==1) {print('Math is ok')}
## [1] "Math is ok"
```

We can make this cleaner using "else":

```
if(1==2){
     print('We need to rethink math')
     } else
          print('Math is ok')
## [1] "Math is ok"
```

Loops

Often you will want to repeat a certain operation multiple times. For example, you may want to print all the integers from 1 to 8.

```
for(i in 1:8){
     print(i)
}
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
```

Exercise

- Draw 10 numbers from a uniform distribution;
- Calculate and print their mean using the functions mean() and print();
- Repeat this operation 100 times.

Solution

Exercise: draw 10 numbers from a uniform distribution; calculate and print their mean using the functions mean() and print(); repeat this operation 12 times.

```
for(i in 1:12){
     mrv <- runif(10)
     mean.mrv <- mean(mrv)
     print(mean.mrv)
}
## [1] 0.4778828
## [1] 0.3624058
## [1] 0.4041901
## [1] 0.5876693
## [1] 0.5343676
## [1] 0.595939
## [1] 0.4004675
```

Exercise

Programme the following situation: there is an urn with 100 balls, 30 of which are black, 70 are red. Draw a ball from this urn, print its color. Repeat this last "draw and print" 12 times PS: the function rep() might be useful. For example, rep('a', 10) will print a 10 times

```r
rep('a', 10)
##  [1] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
```

Solution

```r
blackballs <- rep('black', 30) # create blackballs
redballs <- rep('red', 70) # create redballs
urn <- c(blackballs, redballs) # create the urn

for(i in 1:12){
     mydraw <- sample(urn, 1)
     print(mydraw)
}
## [1] "black"
## [1] "black"
## [1] "red"
## [1] "red"
## [1] "red"
## [1] "red"
## [1] "red"
```

Extension

Reuse the previous function, but this time save your results (black, red, etc.) in a variable called mydraws. How many red/black balls did you get? (the function length(x) calculates the length of vector x)

Solution

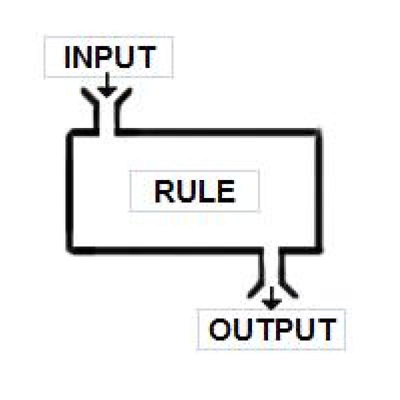
```r
blackballs <- rep('black', 30) # create blackballs
redballs <- rep('red', 70) # create redballs
urn <- c(blackballs, redballs) # create the urn

mydraws <- NULL
for(i in 1:12){
     mydraw <- sample(urn, 1)
     print(mydraw)
     mydraws <- c(mydraws, mydraw)
}
## [1] "black"
## [1] "red"
## [1] "black"
## [1] "black"
## [1] "red"
```

Creating your own Functions

Often you will want to reuse the same routine. It is then useful to create your own function.

Remember from math camp what a function is? A function is basically a machine. It's the same in programming



Creating your own Functions

Create a function that: 1. generate a sequence of n random numbers drawn from a normal distribution; 2. calculate its mean; 3. find its maximum

```
myfunction <- function(){
     x <- rnorm(100)
     print(mean(x))
     print(max(x))
     }
myfunction()
## [1] 0.03691016
## [1] 2.563162
```

(Note that you need 'print', because R does not return the results of calculations done within a function or loop)

Using arguments in your functions

Often you will want to pass an argument to your function. For example, instead of generating a random sequence of numbers, you want to ask your function to calculate the mean and max of a given sequence of numbers

```
myfunction <- function(x){
     print(mean(x))
     print(max(x))
     }
myfunction(x = 1:10)
## [1] 5.5
## [1] 10
```

Exercise: write a function that will return the sum of any two numbers

Getting help

Type ?function. For example:

```
?c
?mean
```