Intro to R

## Downloading and installing R

You can obtain R from CRAN (the Comprehensive R Archive Network), at <http://cran.r-project.org/>. Under “Download R for Windows” click on “install R for the first time” and then on “Download R X.X.X for Windows (Currently R 4.1.2 available)”, which is a self-extracting installer.

After installing R, download and install R Studio Desktop at <https://www.rstudio.com/products/rstudio/download/#download>. R Studio is an integrated development environment which improves the user experience for R, allowing syntax-highlighting, tools for plotting, viewing history, and managing your workspace.

Apart from what you have downloaded there are thousands of add-on packages to R that add extra features and functionality, including related to epidemiology. You can download these manually, as needed.

## How R works

R is an object-oriented programming language, different from the record-oriented nature of SAS. While in SAS each procedure involves the sequential evaluation of every record in the data file before moving to the next step or procedure, each R command line manipulates one or more objects and values can be assigned to other objects or displayed.

Standard functions are built into R in a package named *base*. Additional functions can be installed in packages and are stored locally on the disk in a sub-directory of the folder where R is installed called “library”.

### R Studio

The default set-up for R studio includes a space for script files on the top left. Multiple script files can be open at a time and will run from the same working directory until changed. The bottom left is the console where you can see previously run commands and results. The top right shows the environment and any objects/data available. The bottom right has tabs to show the files in the working directory (“Files”), plots created in this session (“Plots”), installed and loaded packages (“Packages”), and options for help on functions (“Help”).

### The basics

R is case-sensitive, most functions are entirely lowercase while logical values ( *TRUE/T* and *FALSE/F*) are always uppercase.

Important to remember: Everything that **exists** in R is an “object”. Everything than **happens** in R is the result of a function.

Commands can be separated by a new line. Comments can be inserted anywhere, starting with a hash mark (#), everything from the hash mark to the end of the line is a comment.

Documentation on any function from *base* or a package that has been loaded can be obtained by typing “?” in the console (bottom left window) followed by the name of the function and pressing enter. The function documentation will appear in the “Help” window on the bottom right.

?mean()

Lines of code can be run from the script by pressing ctrl+enter when the cursor is on the relevant line. Or you can highlight several lines and press ctrl+enter (or the run button at the top right of the script window).

#### Variable assignment, arithmetic and logical operators

Values are assigned using either “=” or “<-” (keyboard shortcut Alt+-). Typing the variable after assignment will print the value.

x = 1  
y <- 2  
x

## [1] 1

y

## [1] 2

x and y now hold the values 1 and 2, respectively. They can be used in arithmetic or R functions.

Arithmetic operators:

|  |  |
| --- | --- |
| *Operator* | *Description* |
| + | addition |
| - | subtraction |
| \* | multiplication |
| / | division |
| ^ or \*\* | exponentiation |

Logical operators:

|  |  |
| --- | --- |
| *Operator* | *Description* |
| > | greater than |
| >= | greater than or equal to |
| == | exactly equal to |
| != | not equal to |
| & | and |
|  |  |
| 0<=x<=1 | (x>=0) & (x<=1) |
| is.na(x) | test for missing values in x |
|  |  |

x + y # x plus y

## [1] 3

z <- y - x # assign to z the value of y minus x  
z # show the value of z

## [1] 1

is.na(z) # is the value of z missing?

## [1] FALSE

z < 3 # is the value of z less than 3?

## [1] TRUE

z == 3 # is the value of z equal to 3?

## [1] FALSE

z != 3 # is the value of z not equal to 3?

## [1] TRUE

Multiple values can be assigned to an object with *‘c()’* or **concatenate**.

a <- c(3,5,7) # assign 3, 5, and 7 to a  
a # show the value of a

## [1] 3 5 7

a + 1 # add one to a (repeats for each value of a), this value is shown but not saved (no assignment operator)

## [1] 4 6 8

max(a) # what is the maximum

## [1] 7

a <- 7 # over write the value of a with 7  
a # show the value of a

## [1] 7

rm(a) # remove the value of a from the environment (can no longer be used)

#### Data classes and structures

Several data classes can be used in R including numeric, ordered factors, character strings, and logical values. Structure of data can be identified with the function *‘str()’*. Character strings should always be enclosed in quotation marks and logical values should always be in uppercase.

str(x) # structure of x

## num 1

str("hello") # structure of "hello"

## chr "hello"

b <- list(1, TRUE, "hello") # lists can hold multiple structures at the same time  
str(b) # structure of b

## List of 3  
## $ : num 1  
## $ : logi TRUE  
## $ : chr "hello"

Data can be stored in vectors, lists, data frames, and more.

Vectors are the simplest data structure, consisting of an ordered collection of numbers. A number itself could be considered a vector with the length of one.

Vectors can be subset using square brackets ([]) after the name of the vector.

myvector <- c(2,4,8,3,9,4) # assign values to "myvector"  
myvector # show the values of myvector

## [1] 2 4 8 3 9 4

myvector[2] # show the second value of myvector

## [1] 4

myvector[1:3] # show the first three values of myvector

## [1] 2 4 8

myvector[c(2,4)] # show the second and fourth values

## [1] 4 3

myvector<=4 # which values of myvector are <= 4?

## [1] TRUE TRUE FALSE TRUE FALSE TRUE

myvector[myvector<=4] # show the values of myvector that are <= 4

## [1] 2 4 3 4

Data frames are the most common way to store data in R. A data frame is made up of a list of equal-length vectors where each element of the list is a column and the length of each element of the list is the number of rows. Data frames can store different classes in each column (numeric, character, etc). Let’s import data before manipulating data frames.

#### Importing and exploring data

Data can be imported in a variety of formats including csv, xlsx, or from SPSS, SAS, or STATA. Data can also be pulled directly into R from xmart/GHO. Importing files other than csv may require installing/loading a specific package.

##### Installing packages

Packages can be installed from R Studio by clicking on the “Packages” window in the bottom right corner, pressing the “Install” button, and typing the package name(s), and clicking “Install”. After packages are installed, they must be loaded in order to use. Packages are loaded in the console or code script with the function *‘library(packagename)’*. Packages can also be installed with code (but the lines are commented out in this script).

# install.packages("readxl") # install package "readxl" to be able to read Excel files into R  
library(readxl) # load package readxl

##### Importing Excel data

When importing data into R, it’s important to consider where the working directory for R is located. If your data is saved in the working directory, you only need to specify the file name. If the file is saved elsewhere, the entire path must be used. You can learn the current working director with *‘getwd()’* and you can change it with *‘setwd(“yourpathhere”)’*. The file being read in should be assigned (with the assignment operator “<-”, keyboard shortcut Alt+-) to something. Below I assign the policy survey file: **“PolicySurvey\_All.labels.xlsx”** to the object **“policySurvey”**. Be sure to replace my path with yours!

policySurvey <- read\_excel("C:/Users/kpeve/OneDrive - London School of Hygiene and Tropical Medicine/My OneDrive Documents/WHO/Policy-survey/data/raw/PolicySurvey\_All.labels.xlsx")

If you click the top right window “Environment” tab, you should see (in addition to other objects we have created today), **“policySurvey”**, 155 observations of 1016 variables. If you click it, it will open a window where you can scroll around to look at the raw data. Let’s have a look at the data. The function **‘head()’** shows the first six of whatever you put in the brackets. If it’s a vector, the first six items. If it’s a data frame, the first six rows, if it’s a column of a data frame, the first six values. Columns of a data frame are specified after the name of the data frame with a “$”.

head(policySurvey) # show the first six rows of policySurvey (it has too many columns to show all of them, but you'll see the first six rows of the first few columns and then a list of other columns). Note it also shows the data class of each column. The first several columns are "<chr>" for character strings.

## # A tibble: 6 x 1,016  
## iso3 whoname who\_region wb\_income uniqueid startlanguage BI\_0 BI\_0\_other  
## <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>   
## 1 ALB Albania Europe Upper midd~ E010 en Europ~ .   
## 2 ISR Israel Europe High income E011 en Europ~ .   
## 3 SWE Sweden Europe High income E013 en Europ~ .   
## 4 PRT Portugal Europe High income E014 en Europ~ .   
## 5 HRV Croatia Europe High income E015 en Europ~ .   
## 6 EST Estonia Europe High income E016 en Europ~ .   
## # ... with 1,008 more variables: BI\_1 <chr>, BI\_1\_other <chr>, BI\_2 <chr>,  
## # BI\_3 <chr>, BI\_4 <chr>, BI\_5 <chr>, BI\_6 <chr>, BI\_6\_other <chr>,  
## # BI\_7 <chr>, BI\_8 <chr>, BI\_submit <chr>, BI\_submit\_date <chr>, CC\_1 <chr>,  
## # CC\_2 <chr>, CC\_3 <chr>, CC\_4 <chr>, CC\_5\_MOH <chr>, CC\_5\_GOV <chr>,  
## # CC\_5\_GOV\_comment <chr>, CC\_5\_WHO <chr>, CC\_5\_UNICEF <chr>,  
## # CC\_5\_UNFPA <chr>, CC\_5\_OTHER <chr>, CC\_5\_OTHER\_comment <chr>, CC\_6 <chr>,  
## # CC\_7 <chr>, CC\_8 <chr>, CC\_9\_MOH <chr>, CC\_9\_GOV <chr>,  
## # CC\_9\_GOV\_comment <chr>, CC\_9\_WHO <chr>, CC\_9\_UNICEF <chr>,  
## # CC\_9\_UNFPA <chr>, CC\_9\_OTHER <chr>, CC\_9\_OTHER\_comment <chr>, CC\_10 <chr>,  
## # CC\_11\_a\_A <chr>, CC\_11\_a\_B <dbl>, CC\_11\_a\_C <dbl>, CC\_11\_a\_D <chr>,  
## # CC\_11\_b\_A <chr>, CC\_11\_b\_B <dbl>, CC\_11\_b\_C <dbl>, CC\_11\_b\_D <chr>,  
## # CC\_11\_c\_A <chr>, CC\_11\_c\_B <dbl>, CC\_11\_c\_C <dbl>, CC\_11\_c\_D <chr>,  
## # CC\_11\_d\_A <chr>, CC\_11\_d\_B <dbl>, CC\_11\_d\_C <dbl>, CC\_11\_d\_D <chr>,  
## # CC\_11\_e\_A <chr>, CC\_11\_e\_B <dbl>, CC\_11\_e\_C <dbl>, CC\_11\_e\_D <chr>,  
## # CC\_11\_f\_A <chr>, CC\_11\_f\_B <dbl>, CC\_11\_f\_C <dbl>, CC\_11\_f\_D <chr>,  
## # CC\_11\_g\_A <chr>, CC\_11\_g\_B <dbl>, CC\_11\_g\_C <dbl>, CC\_11\_g\_D <chr>,  
## # CC\_11\_h\_A <chr>, CC\_11\_h\_B <dbl>, CC\_11\_h\_C <dbl>, CC\_11\_h\_D <chr>,  
## # CC\_11\_i\_A <chr>, CC\_11\_i\_B <dbl>, CC\_11\_i\_C <dbl>, CC\_11\_i\_D <chr>,  
## # CC\_11\_j\_A <chr>, CC\_11\_j\_B <dbl>, CC\_11\_j\_C <dbl>, CC\_11\_j\_D <chr>,  
## # CC\_11\_k\_A <chr>, CC\_11\_k\_B <dbl>, CC\_11\_k\_C <dbl>, CC\_11\_k\_D <chr>,  
## # CC\_12 <chr>, CC\_13 <chr>, CC\_14\_a <chr>, CC\_14\_b <chr>, CC\_14\_c <chr>,  
## # CC\_14\_d <chr>, CC\_14\_e <chr>, CC\_14\_f <chr>, CC\_14\_g <chr>, CC\_14\_h <chr>,  
## # CC\_14\_i <chr>, CC\_14\_j <chr>, CC\_14\_k <chr>, CC\_14\_k\_other <chr>,  
## # CC\_15 <chr>, CC\_16 <chr>, CC\_17 <chr>, CC\_18 <chr>, CC\_19 <chr>,  
## # CC\_20\_a <chr>, ...

head(policySurvey$iso3) # show the first six values of the iso3 column

## [1] "ALB" "ISR" "SWE" "PRT" "HRV" "EST"

str(policySurvey$MN\_42) # show the data class of the MN\_42 variable

## chr [1:155] "Yes" "Yes" "NA" "No" "Yes" "Yes" "Yes" "Yes" "Yes" "No" "Yes" ...

##### Summarising data

Let’s get further into the data:

table(policySurvey$who\_region) # see the frequencies for who\_region

##   
## Africa Americas Eastern Mediterranean   
## 42 30 16   
## Europe South-East Asia Western Pacific   
## 39 11 17

table(policySurvey$AD\_26\_e\_1) # table of AD\_26\_e\_1 (age minors can receive mental health services without parent/legal consent)

##   
## 12 13 14 15 16 18 100   
## 2 2 4 2 8 24 1

mean(policySurvey$AD\_26\_e\_1, na.rm=T) # mean of AD\_26\_e\_1, note because there are missing values, you must specify "na.rm=T" to remove (rm) NA values from the equation

## [1] 18.51163

summary(policySurvey$AD\_26\_e\_1) # summary shows the min, max, 1st/3rd quartiles, mean, median, and number of missing values (na.rm=T isn't needed)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 12.00 16.00 18.00 18.51 18.00 100.00 112

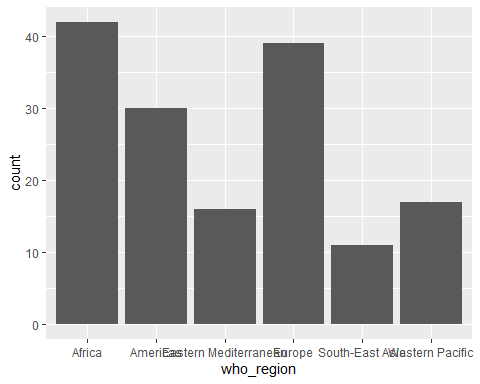
min(policySurvey$AD\_26\_e\_1, na.rm = T) # show minimum value (note na.rm=T is needed), max() shows the maximum value

## [1] 12

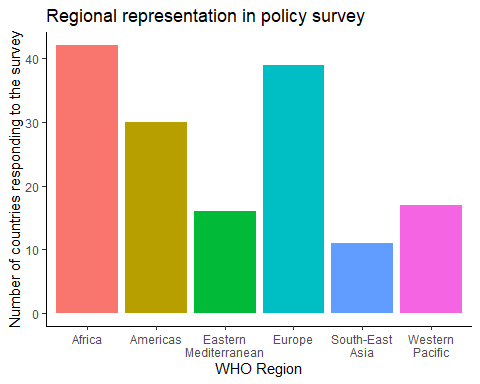
##### Visualising data

The most common plotting package is called ggplot2. We need to install and load it.

# install.packages("ggplot2", "stringr")  
library(ggplot2)  
library(stringr)  
  
# a very simple plot  
  
ggplot(data=policySurvey, aes(x=who\_region))+ # aes is for aesthetics, bar plots only need an x or a y but not both, most other types of plots require x and y to be specified   
 geom\_bar() # define the type of plot



# let's customise it  
  
ggplot(data=policySurvey, aes(x=str\_wrap(who\_region, 10), fill=who\_region))+ # wrap the region labels after 10 characters to fit better   
 geom\_bar() +  
 labs(x="WHO Region", y="Number of countries responding to the survey", title="Regional representation in policy survey")+  
 theme\_classic() + # get rid of the grey tile background  
 guides(fill = F) # don't show the legend for colour



##### Xmart data

Data can be pulled from xmart directly into R. We will need to install/load a few packages. The code below takes data from the MNCAH xmart, specifically the MCA\_FACT\_DATA table. It filters for just the rows including indicators for AC\_A4V (ANC4) and TECI\_ORSZINC. It also selects just the columns specified (excluding other system columns we don’t need). If you get an HTTP error 500, this means the server is busy, try again in a few min. I get this error frequently but can often run the line again successfully immediately or 5-10 minutes later.

# install.packages("jsonlite", "httr")  
library(jsonlite)

## Warning: package 'jsonlite' was built under R version 4.0.5

library(httr)

## Warning: package 'httr' was built under R version 4.0.5

xmart <- fromJSON("https://frontdoor-l4uikgap6gz3m.azurefd.net/MNCAH/MCA\_FACT\_DATA?$filter=INDICATOR\_FK%20in%20(%22AC\_A4V%22,%22TECI\_ORSZINC%22)&$select=INDICATOR\_FK,YEAR\_FK,COUNTRY\_FK,COUNTRY\_FK\_\_CODE,ValueNumeric,ValueLow,ValueHigh,ValueStdDev,ValueStdErr,Comments,AGEGROUP\_FK,Answer\_FK,CauseGHE\_FK,CauseGHEChild\_FK,CauseENVCAUSE\_FK,DatasourceLong,DatasourceShort,ImpSan\_FK,ImpWater\_FK,LivingArrangements\_FK,MEASDEF\_FK,MOTHEREDUCATION\_FK,ProficiencyLevel\_FK,ResideLTA\_FK,RESIDENCEAREA\_FK,SABDefinition\_FK,SEVERITY\_FK,SEVERITYHEARING\_FK,SEVERITYVISION\_FK,SEX\_FK,WEALTHQUINTILE\_FK")  
  
ANC4\_ORSZ <- as.data.frame(xmart$value)  
  
# what are the column names of the data frame  
  
names(ANC4\_ORSZ)

## [1] "INDICATOR\_FK" "YEAR\_FK" "COUNTRY\_FK"   
## [4] "COUNTRY\_FK\_\_CODE" "ValueNumeric" "ValueLow"   
## [7] "ValueHigh" "ValueStdDev" "ValueStdErr"   
## [10] "Comments" "AGEGROUP\_FK" "Answer\_FK"   
## [13] "CauseGHE\_FK" "CauseGHEChild\_FK" "CauseENVCAUSE\_FK"   
## [16] "DatasourceLong" "DatasourceShort" "ImpSan\_FK"   
## [19] "ImpWater\_FK" "LivingArrangements\_FK" "MEASDEF\_FK"   
## [22] "MOTHEREDUCATION\_FK" "ProficiencyLevel\_FK" "ResideLTA\_FK"   
## [25] "RESIDENCEAREA\_FK" "SABDefinition\_FK" "SEVERITY\_FK"   
## [28] "SEVERITYHEARING\_FK" "SEVERITYVISION\_FK" "SEX\_FK"   
## [31] "WEALTHQUINTILE\_FK"

##### Data manipulation

Dplyr is part of a common data manipulation package called tidyverse.

# install.packages("tidyverse")  
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v tibble 3.1.0 v purrr 0.3.4  
## v tidyr 1.1.3 v dplyr 1.0.5  
## v readr 1.4.0 v forcats 0.5.0

## Warning: package 'tibble' was built under R version 4.0.5

## Warning: package 'tidyr' was built under R version 4.0.5

## Warning: package 'dplyr' was built under R version 4.0.5

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x purrr::flatten() masks jsonlite::flatten()  
## x dplyr::lag() masks stats::lag()

# Let's filter for just ANC4 rows  
# the pipe operator ( %>% ) passes data from one function to the next to continue manipulating the same object. The keyboard shortcut is ctrl+shift+M  
  
ANC4 <- ANC4\_ORSZ %>% filter(INDICATOR\_FK=="AC\_A4V")  
  
# Notice in the environment tab, you now have ANC$\_ORSZ (2433 observations) and a second object ANC4 which is 839 observations, let's see what years have data for ANC4  
  
table(ANC4$YEAR\_FK)

##   
## 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005   
## 4 6 14 11 6 14 17 16 16 17 24 18 16 24 22 24   
## 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020   
## 23 35 25 34 48 49 55 50 57 42 45 42 43 33 9

#let's take just data from 2010 or later  
  
ANC4 <- ANC4 %>% filter(YEAR\_FK>=2010) # this replaces the previous ANC4 data frame with a data frame for ANC4 only including rows from 2010 or later  
table(ANC4$YEAR\_FK)

##   
## 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020   
## 48 49 55 50 57 42 45 42 43 33 9

# how many countries have data  
  
length(unique(ANC4$COUNTRY\_FK\_\_CODE))

## [1] 148