BIOS-IN5410

Introduction to R programming

Learning goals

Introduce you to R and Rstudio

Basic R functionality

Find and install packages

Be able to read package manuals and find help

Read and write files

Plotting data

(Very rough) time plan

Friday Nov 19

09:15-10:00

- Introduction to R and RStudio
- Set up and get going
- Do Exercise 1

10:15 - 12:00

- Go through Exercise 1
- R packages and the Tidyverse
- Rectangular and tidy data
- Working with files
- Exercise 2
- Go through Exercise 2

12:45 - 14:00

- Manipulating data with dplyr
- Exercise 3

14:15 - 16:00

- Go through Exercise 3
- Basic plotting
- Exercise 4
- Go through exercise 4 together

Monday Nov 22

09:15 - 11:30

- Programming basics
  - For loops + Ex 5 (09:15 - 10:30)
  - Ex 5 + If statements + Ex 6 (10:45 - 11:30)
  - Go through exercise 6 (11:30 - 12:00)

12:45

- R scripts
  - Running R on the command line
  - Command line arguments
- Plotting with ggplot2

R resources

Introduction to Data Science - free online book (most of the material in this course is taken from here): https://rafalab.github.io/dsbook/

R for Data Science - free online book: https://r4ds.had.co.nz/

Software Carpentry - https://swcarpentry.github.io/r-novice-gapminder/

The R project

Environment for statistical computing and graphics

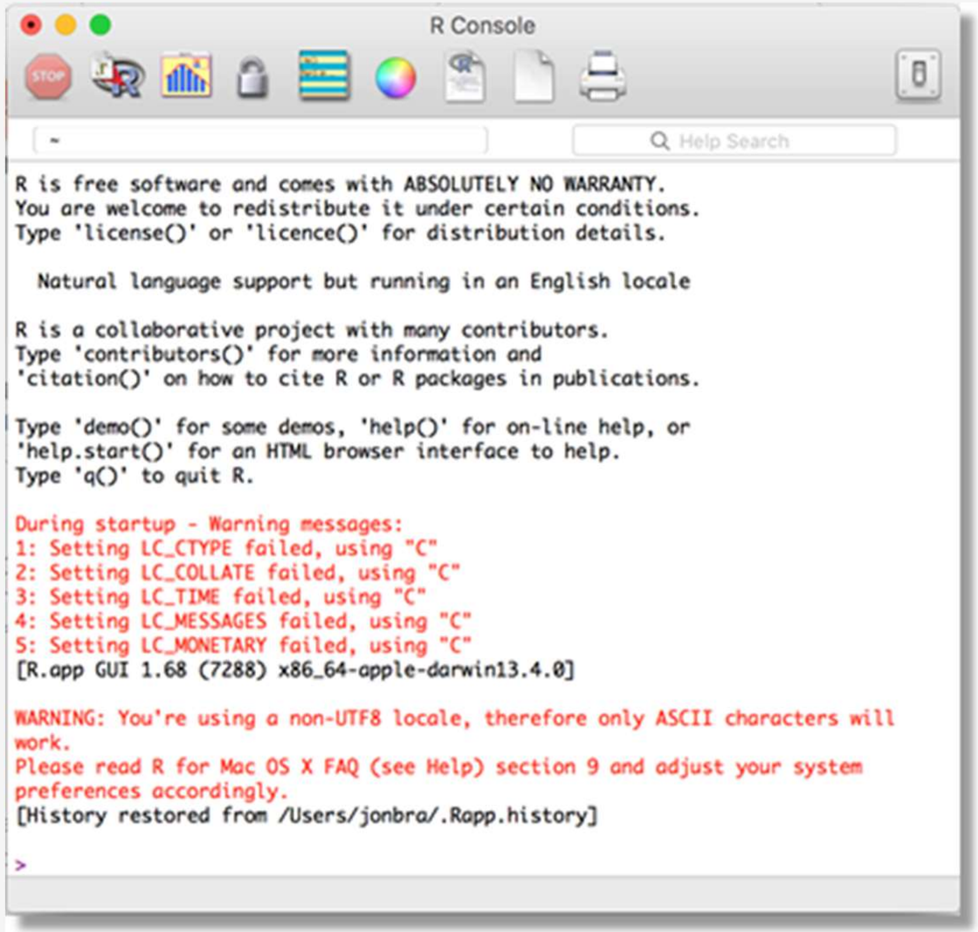
It’s free

Can be run on Windows, Mac, Unix…

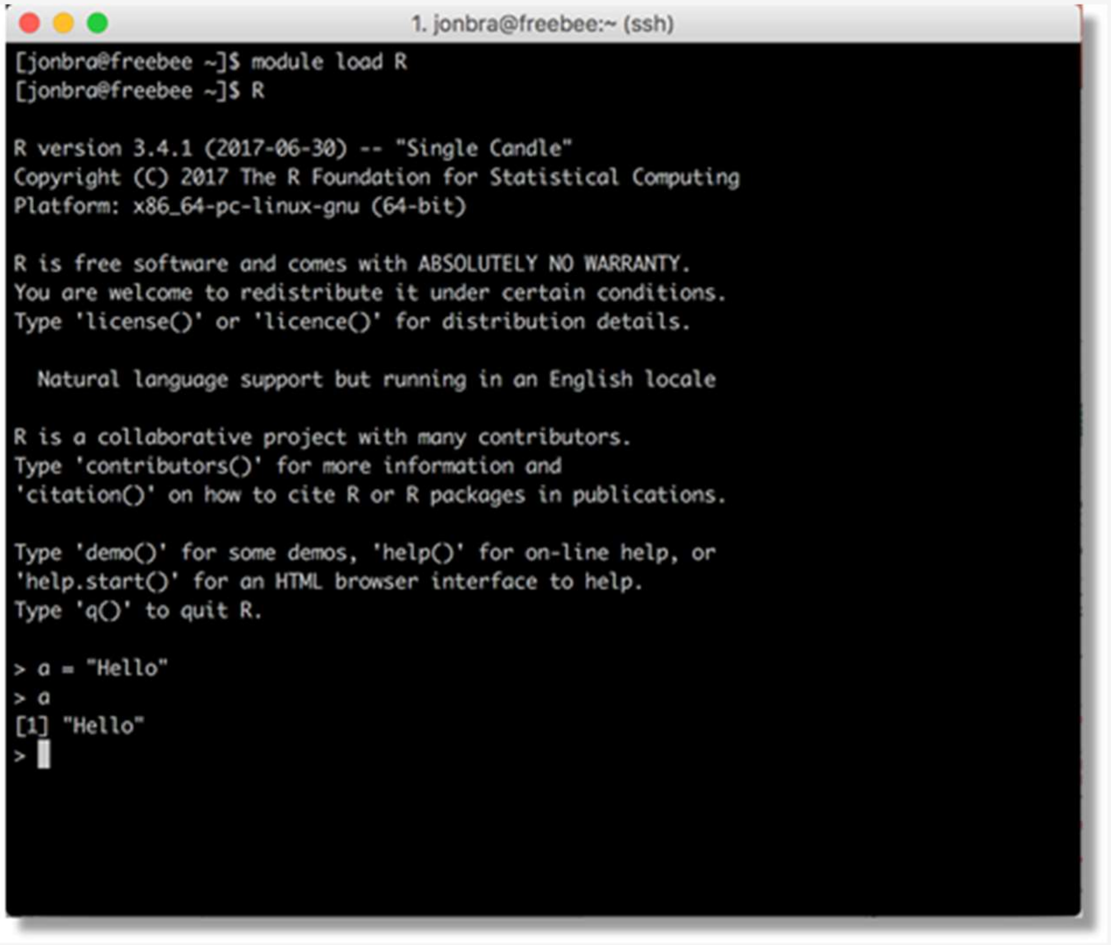
Extremely rich selection of packages

Very good for graphics and plotting

The R console

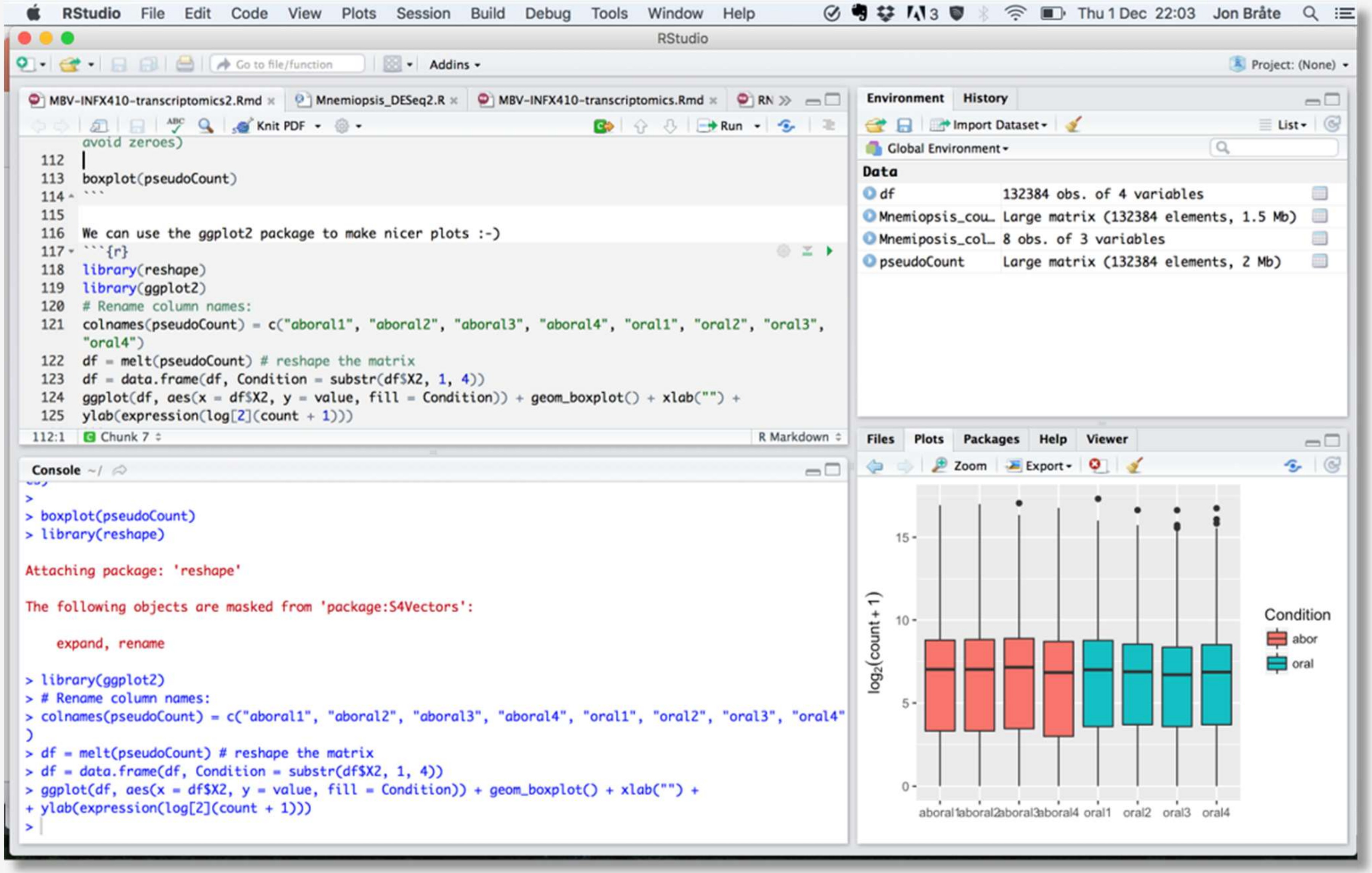


```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
[R.app GUI 1.68 (7288) x86_64-apple-darwin13.4.0]

WARNING: You're using a non-UTF8 locale, therefore only ASCII characters will
work.
Please read R for Mac OS X FAQ (see Help) section 9 and adjust your system
preferences accordingly.
[History restored from /Users/jonbra/.Rapp.history]

>
```



```
[jonbra@freebee ~]$ module load R
[jonbra@freebee ~]$ R

R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> a = "Hello"
> a
[1] "Hello"
>
```

RStudio - an R IDE



```
avoid zeroes)

boxplot(pseudoCount)
```

We can use the ggplot2 package to make nicer plots :-)
```{r}
library(reshape)
library(ggplot2)
# Rename column names:
colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3",
"oral4")
df = melt(pseudoCount) # reshape the matrix
df = data.frame(df, Condition = substr(df$X2, 1, 4))
ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
ylab(expression(log[2](count + 1)))
```

Data

df	132384 obs. of 4 variables
Mnemiopsis_cou…	Large matrix (132384 elements, 1.5 Mb)
Mnemiposis_col…	8 obs. of 3 variables
pseudoCount	Large matrix (132384 elements, 2 Mb)

```
>
> boxplot(pseudoCount)
> library(reshape)

Attaching package: 'reshape'

The following objects are masked from 'package:S4Vectors':

    expand, rename

> library(ggplot2)
> # Rename column names:
> colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3", "oral4"
)
> df = melt(pseudoCount) # reshape the matrix
> df = data.frame(df, Condition = substr(df$X2, 1, 4))
> ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
+ ylab(expression(log[2](count + 1)))
>
```

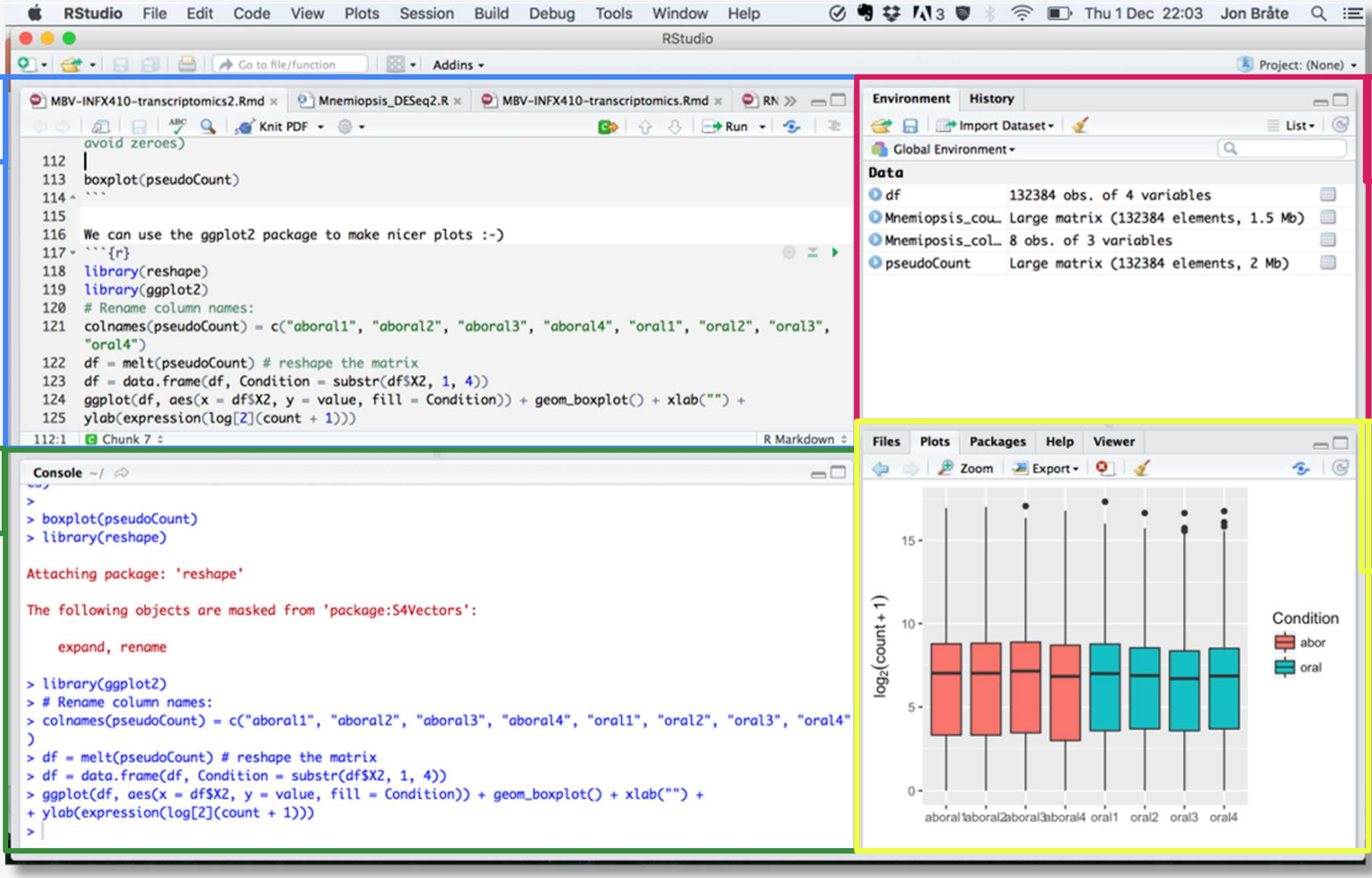
RStudio - an R IDE

Text editor

Console

Environment window

View plots, packages, files, help and more



```
avoid zeroes)

boxplot(pseudoCount)
```

We can use the ggplot2 package to make nicer plots :-)
```{r}
library(reshape)
library(ggplot2)
# Rename column names:
colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3",
"oral4")
df = melt(pseudoCount) # reshape the matrix
df = data.frame(df, Condition = substr(df$X2, 1, 4))
ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
ylab(expression(log[2](count + 1)))
```

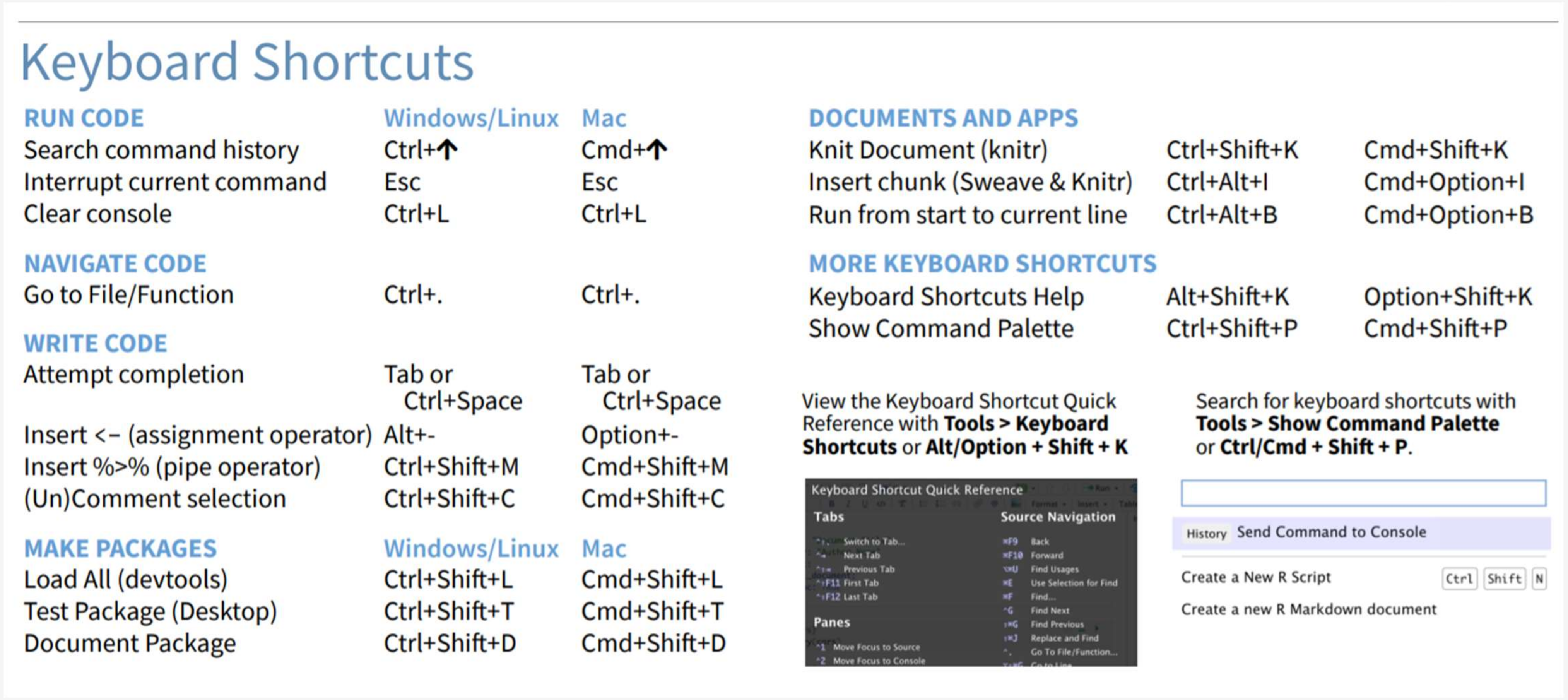
```
>
> boxplot(pseudoCount)
> library(reshape)

Attaching package: 'reshape'

The following objects are masked from 'package:S4Vectors':

    expand, rename

> library(ggplot2)
> # Rename column names:
> colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3", "oral4"
)
> df = melt(pseudoCount) # reshape the matrix
> df = data.frame(df, Condition = substr(df$X2, 1, 4))
> ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
+ ylab(expression(log[2](count + 1)))
>
```

Data

df	132384 obs. of 4 variables
Mnemiopsis_cou…	Large matrix (132384 elements, 1.5 Mb)
Mnemiposis_col…	8 obs. of 3 variables
pseudoCount	Large matrix (132384 elements, 2 Mb)

RStudio - cheat sheet

Check out the RStudio cheat sheet in the GitHub repo - especially the shortcuts.



Keyboard Shortcuts

RUN CODE	Windows/Linux	Mac
Search command history	Ctrl+↑	Cmd+↑
Interrupt current command	Esc	Esc
Clear console	Ctrl+L	Ctrl+L
NAVIGATE CODE		
Go to File/Function	Ctrl+.	Ctrl+.
WRITE CODE		
Attempt completion	Tab or Ctrl+Space	Tab or Ctrl+Space
Insert <– (assignment operator)	Alt+-	Option+-
Insert %>% (pipe operator)	Ctrl+Shift+M	Cmd+Shift+M
(Un)Comment selection	Ctrl+Shift+C	Cmd+Shift+C
MAKE PACKAGES	**Windows/Linux**	**Mac**
Load All (devtools)	Ctrl+Shift+L	Cmd+Shift+L
Test Package (Desktop)	Ctrl+Shift+T	Cmd+Shift+T
Document Package	Ctrl+Shift+D	Cmd+Shift+D

DOCUMENTS AND APPS		
Knit Document (knitr)	Ctrl+Shift+K	Cmd+Shift+K
Insert chunk (Sweave & Knitr)	Ctrl+Alt+I	Cmd+Option+I
Run from start to current line	Ctrl+Alt+B	Cmd+Option+B
MORE KEYBOARD SHORTCUTS		
Keyboard Shortcuts Help	Alt+Shift+K	Option+Shift+K
Show Command Palette	Ctrl+Shift+P	Cmd+Shift+P

View the Keyboard Shortcut Quick Reference with **Tools > Keyboard Shortcuts** or **Alt/Option + Shift + K**

Keyboard Shortcut Quick Reference

Tabs
- ^⇧, Switch to Tab...
- ^⇥ Next Tab
- ^⇧⇥ Previous Tab
- ^⇧F11 First Tab
- ^⇧F12 Last Tab

Panes
- ^1 Move Focus to Source
- ^2 Move Focus to Console

Source Navigation
- ⌘F9 Back
- ⌘F10 Forward
- ⌥⌘U Find Usages
- ⌘E Use Selection for Find
- ⌘F Find...
- ^G Find Next
- ⇧⌘G Find Previous
- ⇧⌘J Replace and Find
- ^. Go To File/Function...

Search for keyboard shortcuts with **Tools > Show Command Palette** or **Ctrl/Cmd + Shift + P**.

History Send Command to Console

Create a New R Script Ctrl Shift N

Create a new R Markdown document

A (super) short introduction to R functionality

(you don’t need to remember all the details. Use the slides as a reference)

Variable assignment

We assign values to variables with the assignment operator "<-” (can also use “=”).
Just typing the variable by itself at the prompt will print out the value.

```
> x <- 1
> x
[1] 1
> x = 1
> x
[1] 1
> y <- 2
> x + y
[1] 3
```

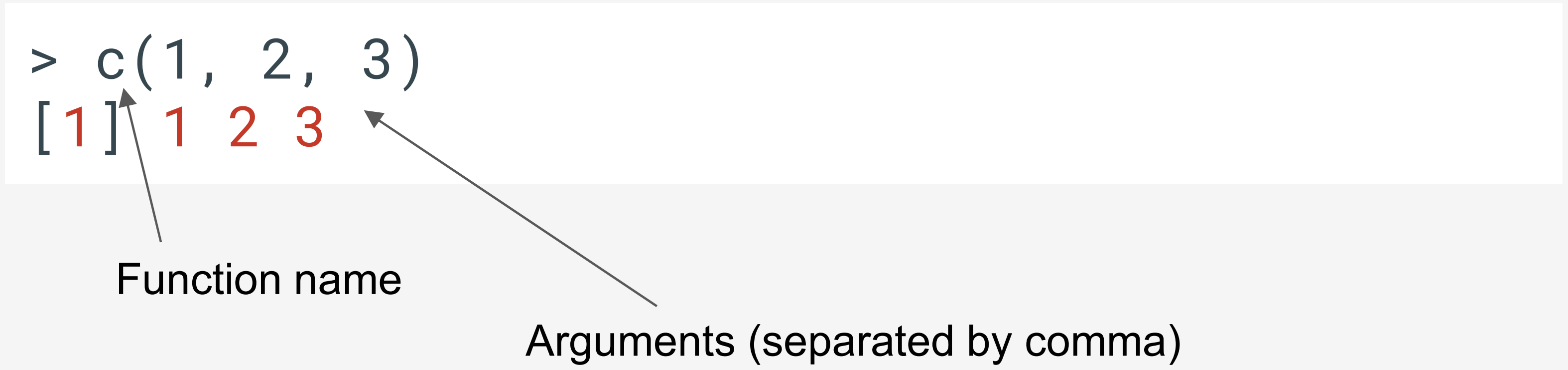
The prompt (like the $ in the Unix terminal)

R is very good for mathematics

```
> 1+1 # Simple arithmetic
[1] 2
> 2 + 3 * 4 # Operator precedence
[1] 14
> 3 ^ 2 # Exponentiation
[1] 9
> exp(1) # Basic mathematical functions are available
[1] 2.718282
> sqrt(10)
[1] 3.162278
> pi # The constant pi is predefined
[1] 3.141593
> 2*pi*6378 # Circumference of earth at equator (in km)
[1] 40074.16
```

Functions

R functions are invoked by its name, then followed by the parenthesis, and zero or more arguments. The following apply the function c() to combine three numeric values into a vector.



```
> c(1, 2, 3)
[1] 1 2 3
```

Comments

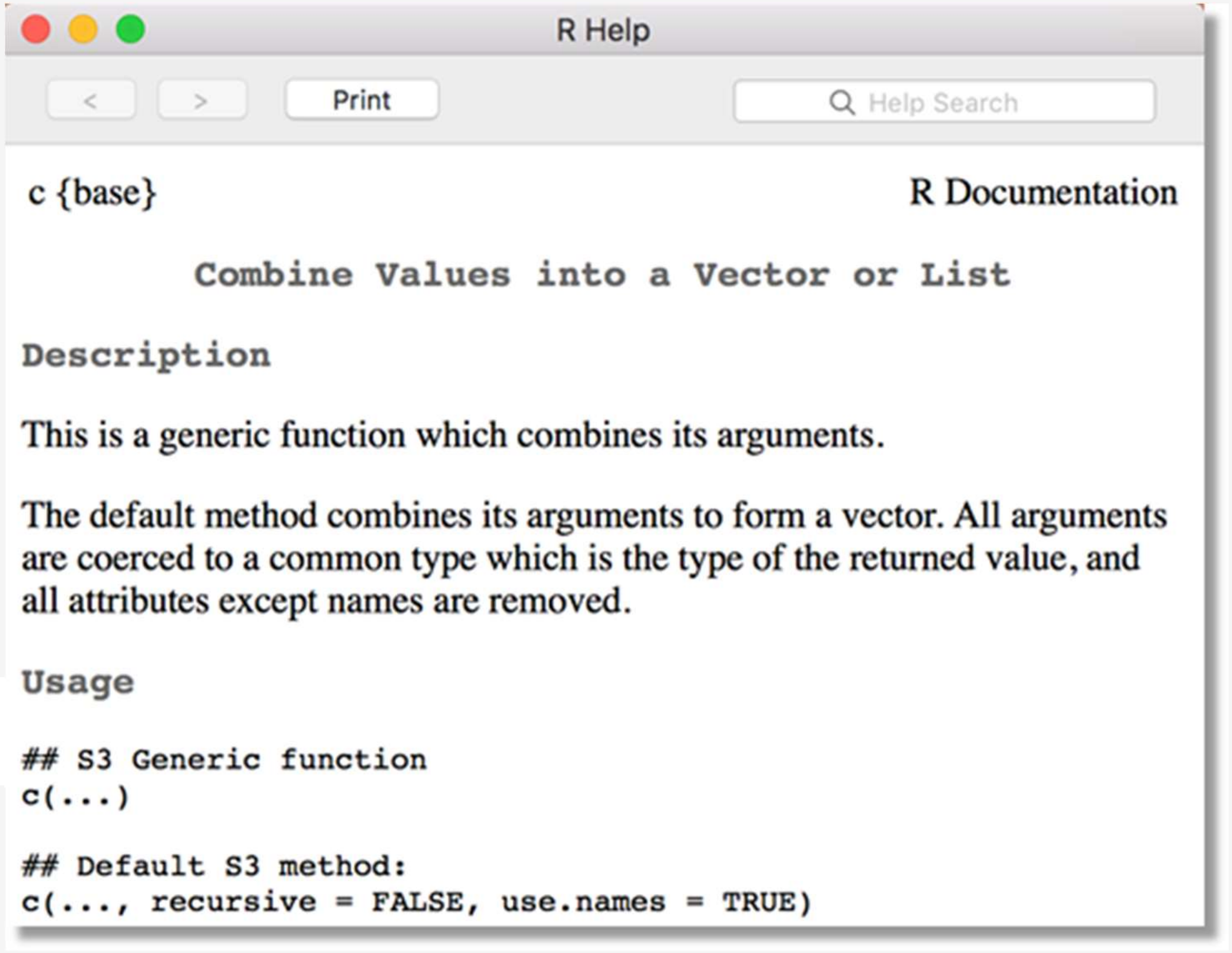
Just like in unix/bash, all text after the hash tag "#" within the same line is considered a comment.

```
> 1 + 1 # This is a comment
[1] 2
```

Getting help

R provides extensive documentation. For example, entering ?c or help(c) at the prompt gives documentation of the function c in R.

```
> help(c)
```



c {base} R Documentation

Combine Values into a Vector or List

Description

This is a generic function which combines its arguments.

The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value, and all attributes except names are removed.

Usage

```
## S3 Generic function
c(...)

## Default S3 method:
c(..., recursive = FALSE, use.names = TRUE)
```

Get started with R

Install R (r-project.org)

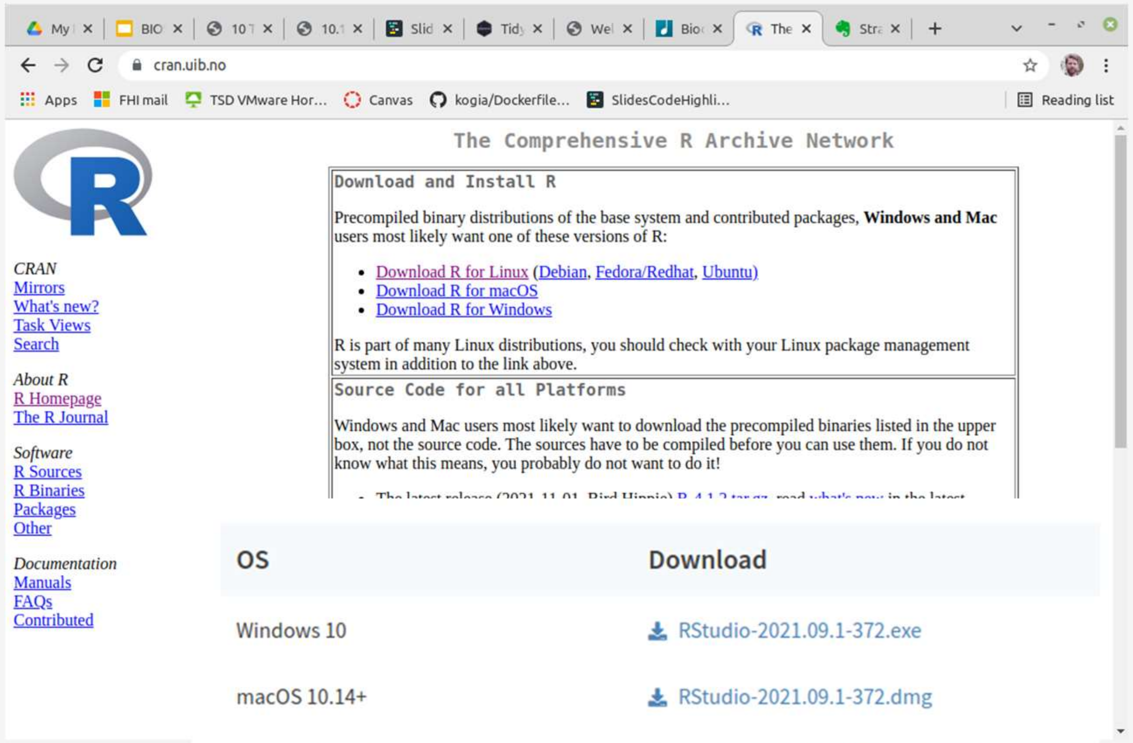
cran.uib.no

Choose the right OS

Install RStudio (rstudio.com)

Choose the right OS

https://www.rstudio.com/products/rstudio/download/#download



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

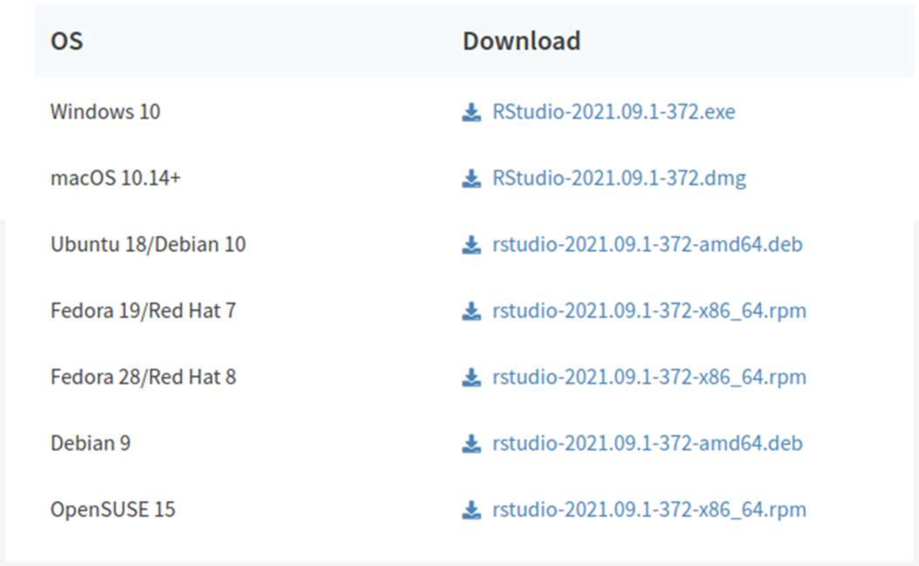
Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

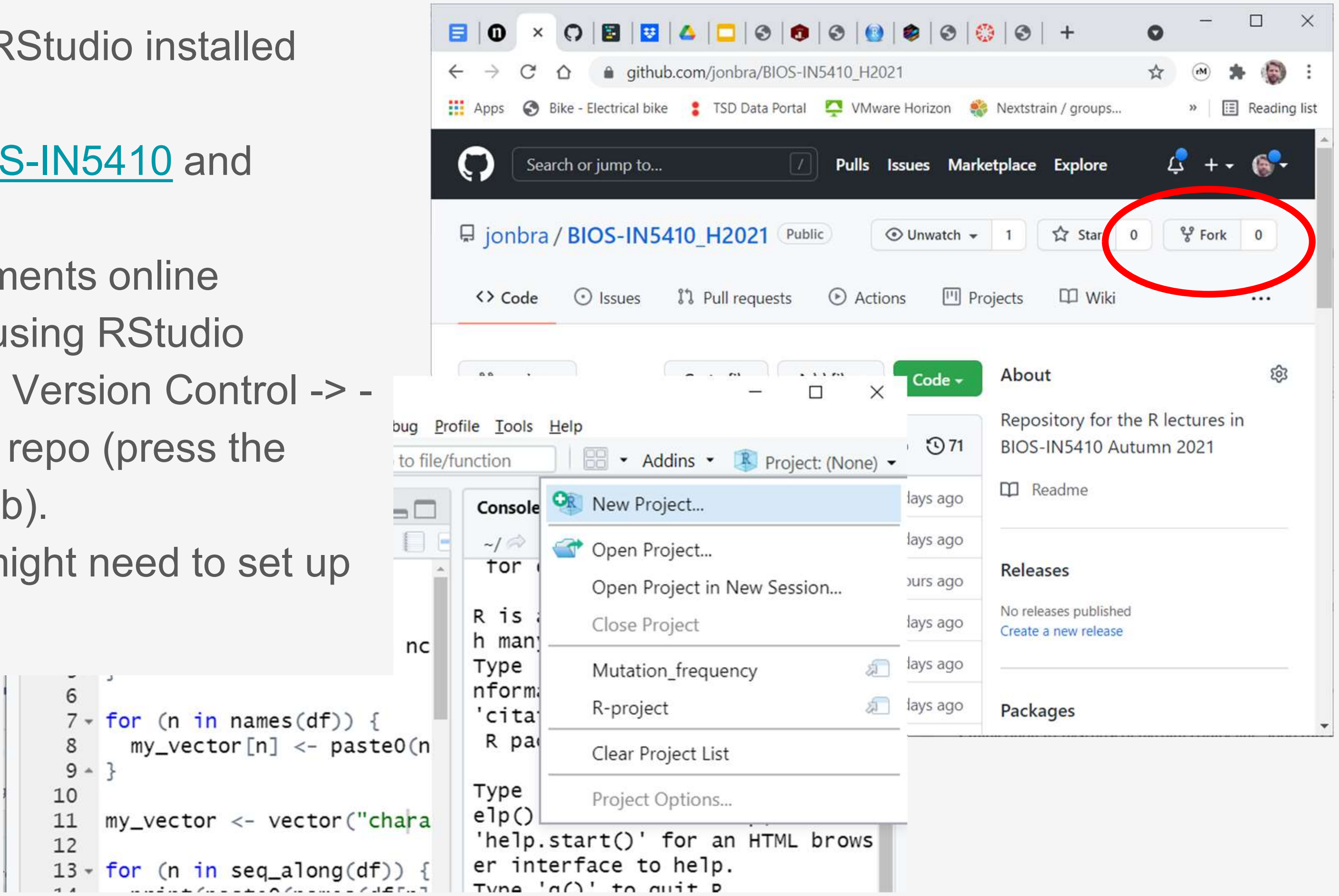
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!



OS	Download
Windows 10	RStudio-2021.09.1-372.exe
macOS 10.14+	RStudio-2021.09.1-372.dmg
Ubuntu 18/Debian 10	rstudio-2021.09.1-372-amd64.deb
Fedora 19/Red Hat 7	rstudio-2021.09.1-372-x86_64.rpm
Fedora 28/Red Hat 8	rstudio-2021.09.1-372-x86_64.rpm
Debian 9	rstudio-2021.09.1-372-amd64.deb
OpenSUSE 15	rstudio-2021.09.1-372-x86_64.rpm

Time to try R for yourself

- First, make sure you have R and RStudio installed and working
- Then go to github.com/jonbra/BIOS-IN5410 and either:
    - Just read the different documents online
    - Or, fork and clone the repo using RStudio (Project -> New Project… -> Version Control -> -> Git -> Paste the link to the repo (press the green Code button on GitHub).
    - NB! To clone the repo you might need to set up ssh keys – can be tricky!



```
for (n in names(df)) {
  my_vector[n] <- paste0(n
}

my_vector <- vector("chara

for (n in seq_along(df)) {
```

```
'help.start()' for an HTML brows
er interface to help.
```

Time to try R for yourself

- Make sure R and RStudio is installed and working.
- Test writing commands, both in the editor and the console.
- Try to assign some variables, change them, etc.
- Do Exercise 1 in your repo (we will always go through the exercises together).
- And just play around in R and RStudio (e.g. check out the cheat sheet).
- *And help each other! I haven’t given you all the details you need so you need to check the help menus and search the web.*

First break

R-packages

In addition to “base R”, there are thousands of so-called “packages” that gives additional functionality to R.

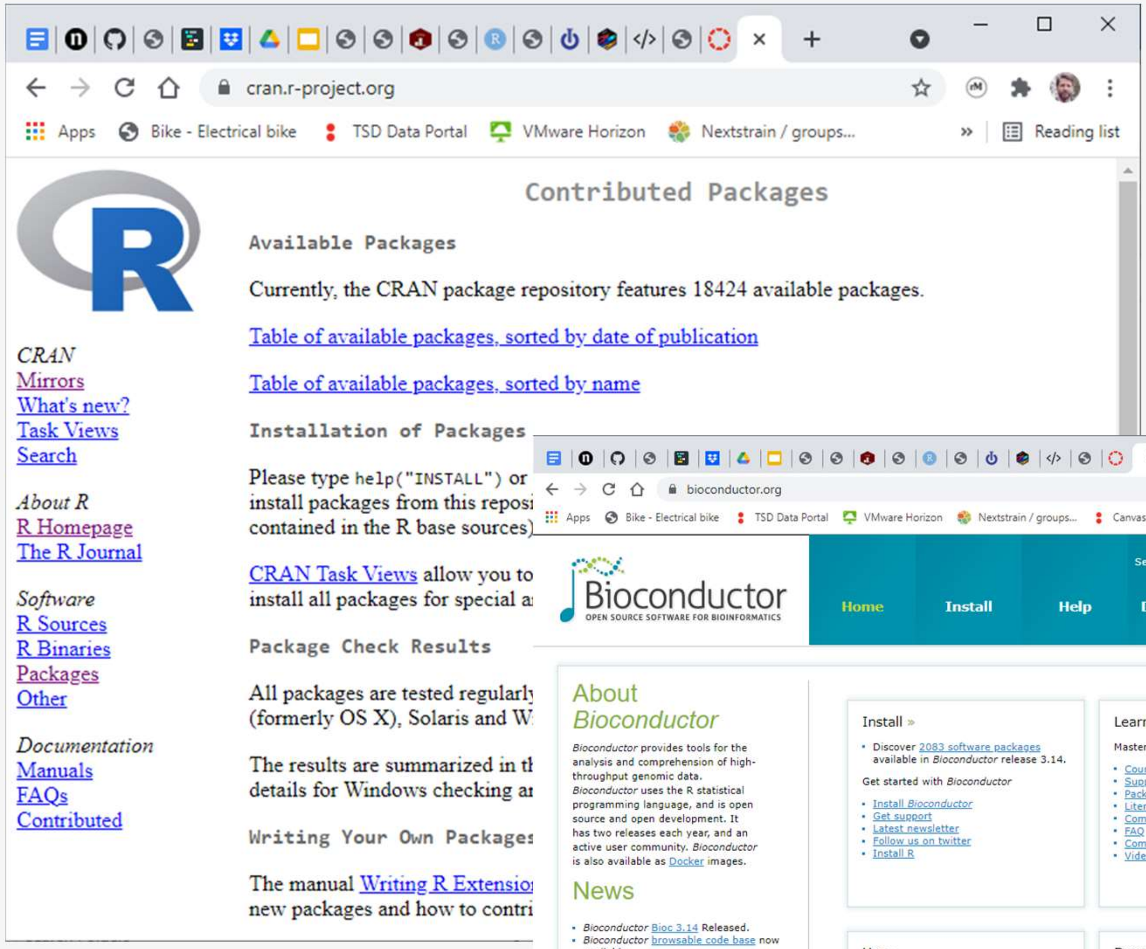
CRAN and Bioconductor are the main repositories for packages.

Packages needs to be installed, e.g. by typing

```
install.packages("package")
```

And activated before use by typing

```
library("package")
```



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Contributed Packages

Available Packages

Currently, the CRAN package repository features 18424 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type `help("INSTALL")` or
install packages from this reposi
contained in the R base sources)

CRAN Task Views allow you to
install all packages for special a

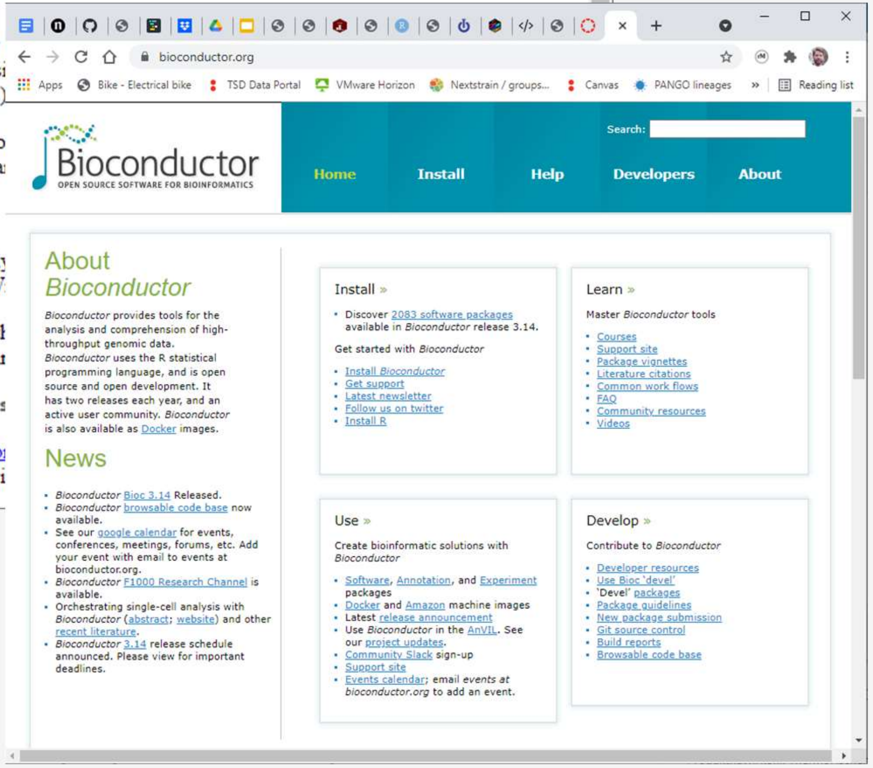
Package Check Results

All packages are tested regularly
(formerly OS X), Solaris and W

The results are summarized in th
details for Windows checking ar

Writing Your Own Packages

The manual Writing R Extensio
new packages and how to contri



About *Bioconductor*

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. *Bioconductor* uses the R statistical programming language, and is open source and open development. It has two releases each year, and an active user community. *Bioconductor* is also available as Docker images.

News

- *Bioconductor* Bioc 3.14 Released.
- *Bioconductor* browsable code base now available.
- See our google calendar for events, conferences, meetings, forums, etc. Add your event with email to events at bioconductor.org.
- *Bioconductor* F1000 Research Channel is available.
- Orchestrating single-cell analysis with *Bioconductor* (abstract; website) and other recent literature.
- *Bioconductor* 3.14 release schedule announced. Please view for important deadlines.

Install »

- Discover 2083 software packages available in *Bioconductor* release 3.14.

Get started with *Bioconductor*

- Install *Bioconductor*
- Get support
- Latest newsletter
- Follow us on twitter
- Install R

Learn »

Master *Bioconductor* tools

- Courses
- Support site
- Package vignettes
- Literature citations
- Common work flows
- FAQ
- Community resources
- Videos

Use »

Create bioinformatic solutions with *Bioconductor*

- Software, Annotation, and Experiment packages
- Docker and Amazon machine images
- Latest release announcement
- Use *Bioconductor* in the AnVIL. See our project updates.
- Community Slack sign-up
- Support site
- Events calendar; email *events at bioconductor.org* to add an event.

Develop »

Contribute to *Bioconductor*

- Developer resources
- Use Bioc 'devel'
- 'Devel' packages
- Package guidelines
- New package submission
- Git source control
- Build reports
- Browsable code base

Tidyverse

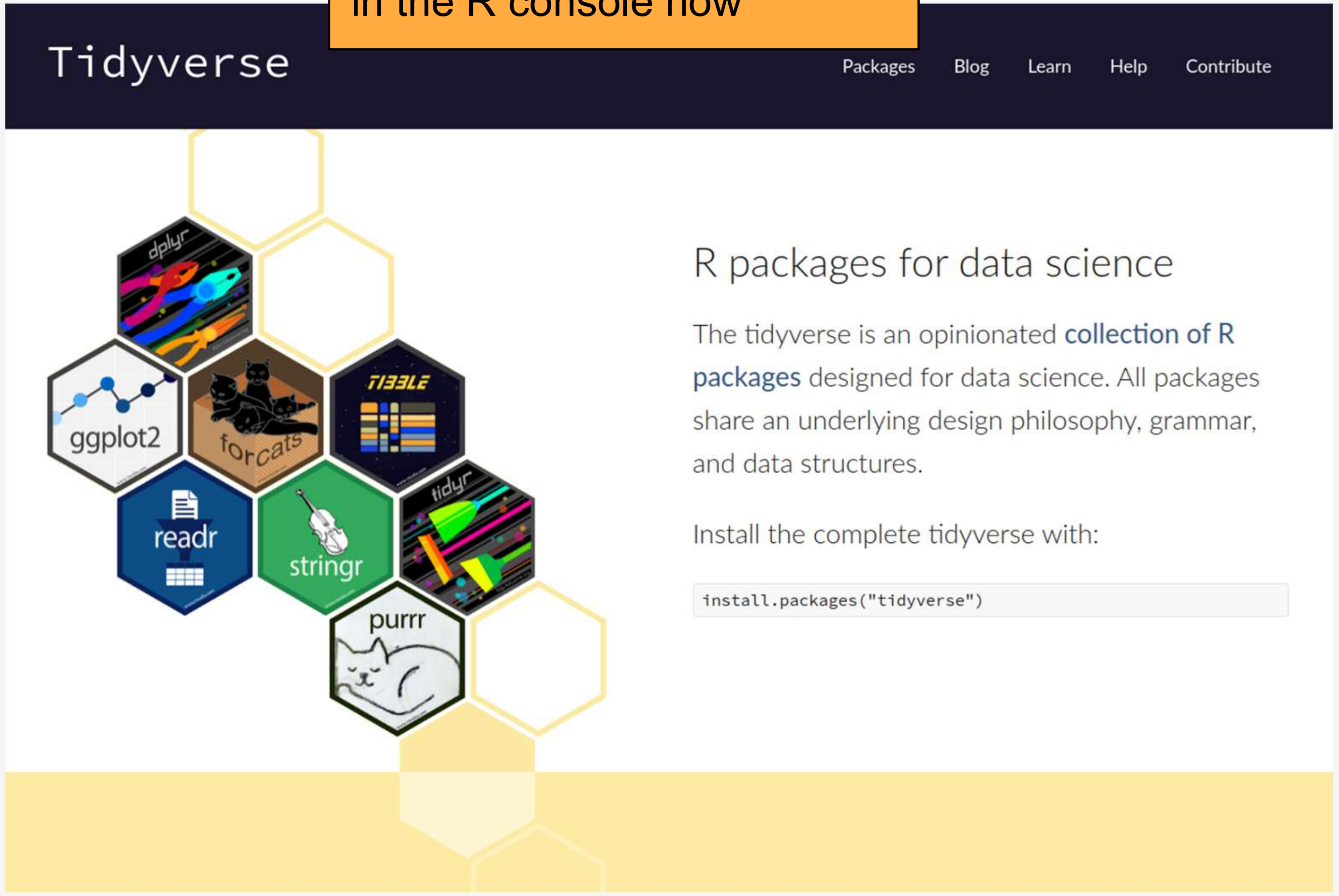
*"A system of packages for **data manipulation**, **exploration** and **visualization** that share a common design philosophy."*

Centered around "Rectangular data structures" (e.g. data frames, matrices..)

tidyverse.org

```
install.packages("tidyverse")
```

Everyone should try to run install.packages("tidyverse") in the R console now



R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

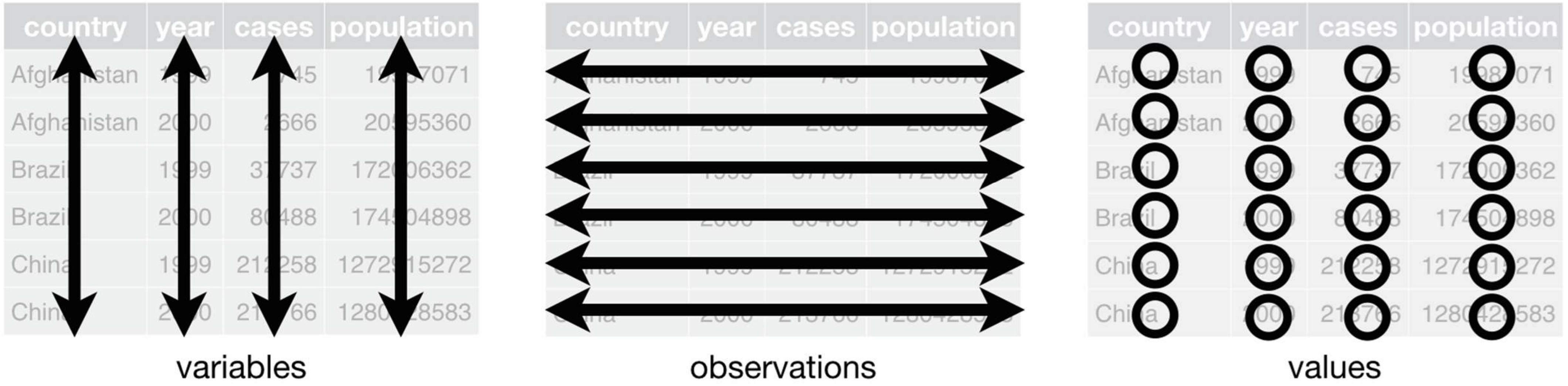
Free online book for learning R and the tidyverse: https://r4ds.had.co.nz/

The rectangular data type

A lot of the work you will do in R is centered around “rectangular data”, or data frames. Data frames are like tables with each row is a record and the columns are the different variables.

```
       state abb region population total
1    Alabama  AL  South    4779736   135
2     Alaska  AK   West     710231    19
3    Arizona  AZ   West    6392017   232
4   Arkansas  AR  South    2915918    93
5 California  CA   West   37253956  1257
6   Colorado  CO   West    5029196    65
```

Tidy data



R for Data Science, Hadley Wickham

Tidy data

We say that a data table is in ***tidy format*** if each row represents one observation and columns represent the different variables available for each of these observations.

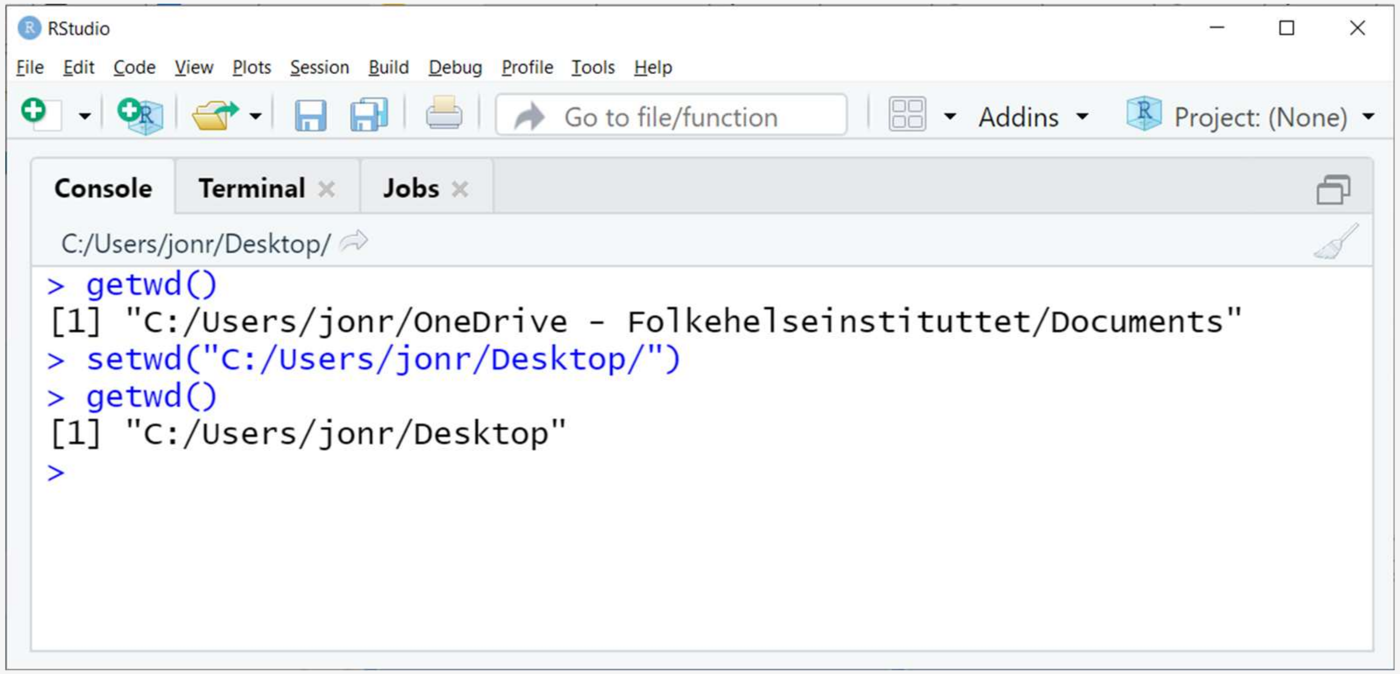
```
      country year fertility
1     Germany 1960      2.41
2 South Korea 1960      6.16
3     Germany 1961      2.44
4 South Korea 1961      5.99
5     Germany 1962      2.47
6 South Korea 1962      5.79
```

```
      country 1960 1961 1962
1     Germany 2.41 2.44 2.47
2 South Korea 6.16 5.99 5.79
```

https://rafalab.github.io/dsbook/tidyverse.html

Working directory

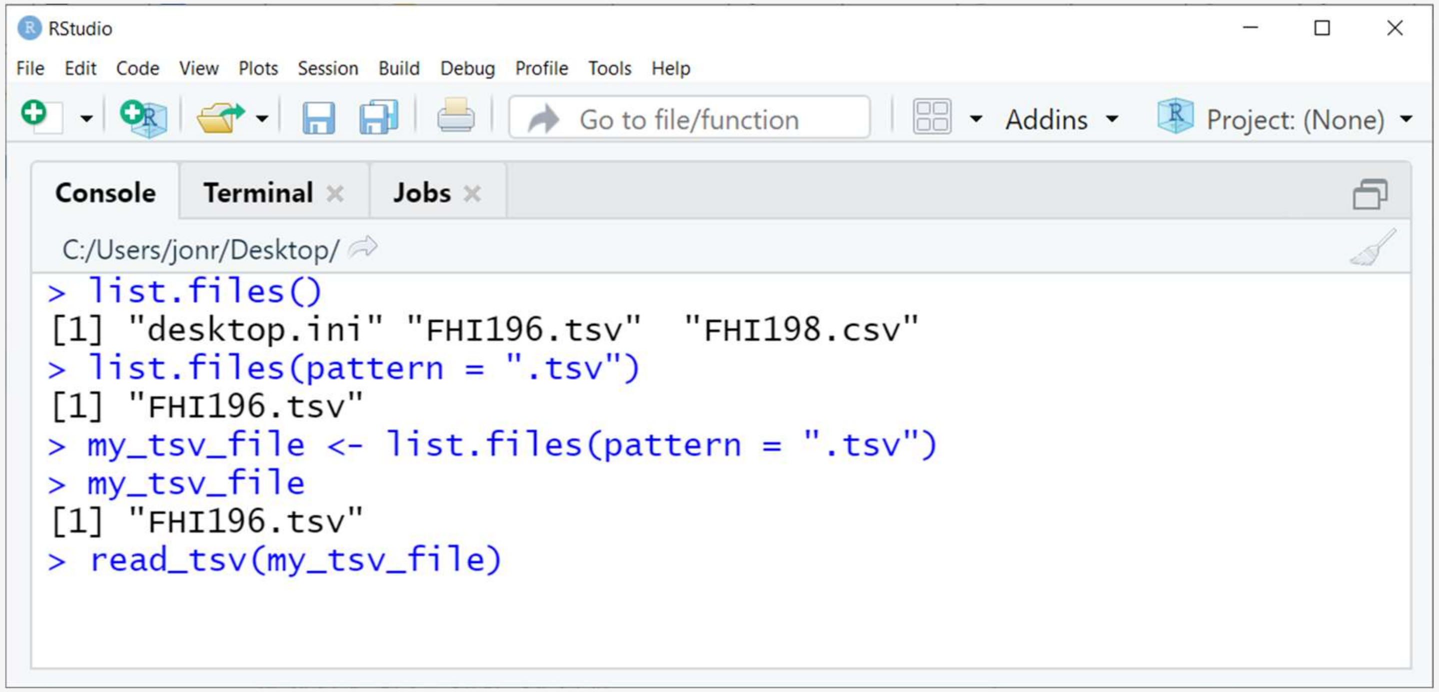
The *getwd()* function let’s you see where on your file system R is currently working. Change the working directory with *setwd()*.



```
> getwd()
[1] "C:/Users/jonr/OneDrive - Folkehelseinstituttet/Documents"
> setwd("C:/Users/jonr/Desktop/")
> getwd()
[1] "C:/Users/jonr/Desktop"
>
```

File system - access files

lists.files() and *list.dirs()* will show the files and the directories in the working directory. Use the *pattern* argument to filter what kind of files or directories to be listed.



```
> list.files()
[1] "desktop.ini" "FHI196.tsv"  "FHI198.csv"
> list.files(pattern = ".tsv")
[1] "FHI196.tsv"
> my_tsv_file <- list.files(pattern = ".tsv")
> my_tsv_file
[1] "FHI196.tsv"
> read_tsv(my_tsv_file)
```

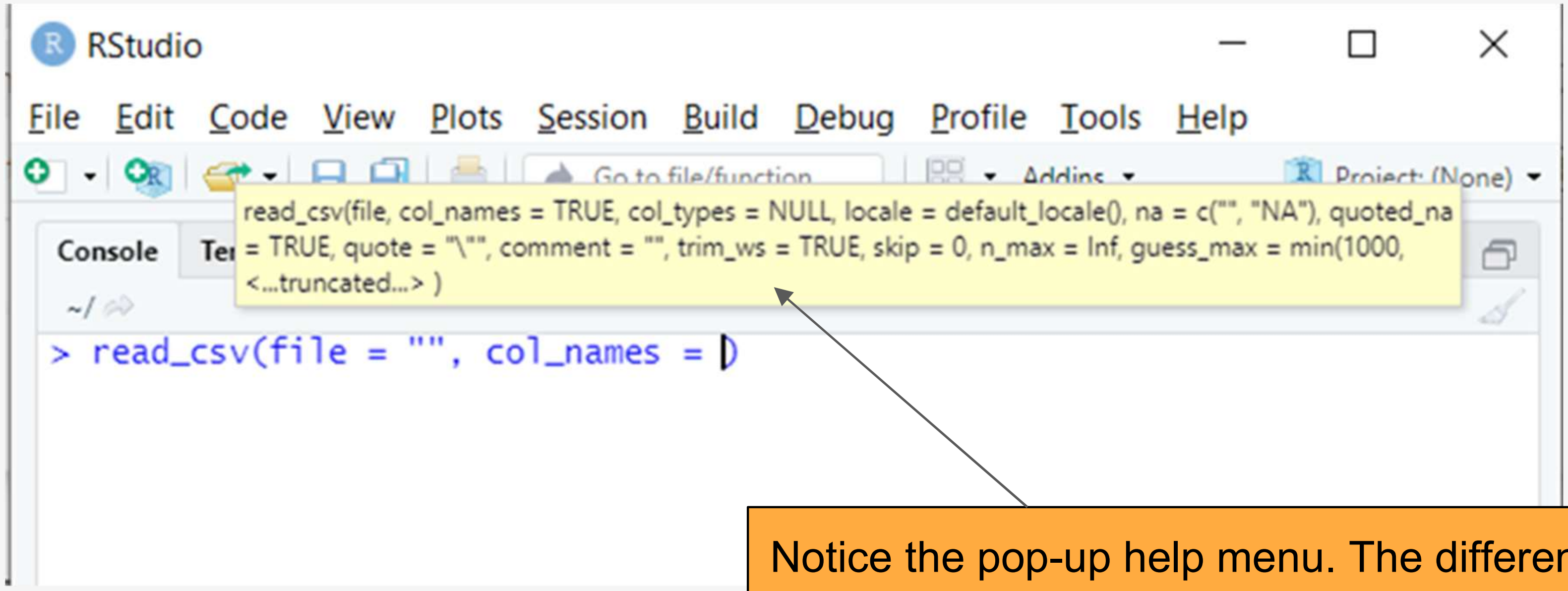
Getting data into R - the readr package

There are many ways of getting data from files into R. The readr package offers several functions for reading different data types.

```
read_csv(): comma separated (CSV) files
read_tsv(): tab separated files
read_delim(): general delimited files
read_fwf(): fixed width files
read_table(): tabular files where columns are
separated by white-space.
read_log(): web log files
```

Getting data into R - the readr package

The functions have different arguments that can be used to further specify the structure of the file to be read. E.g. does the file have a header line? What type of symbol separates the columns? Are there any lines that should be skipped? Etc.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console
~/

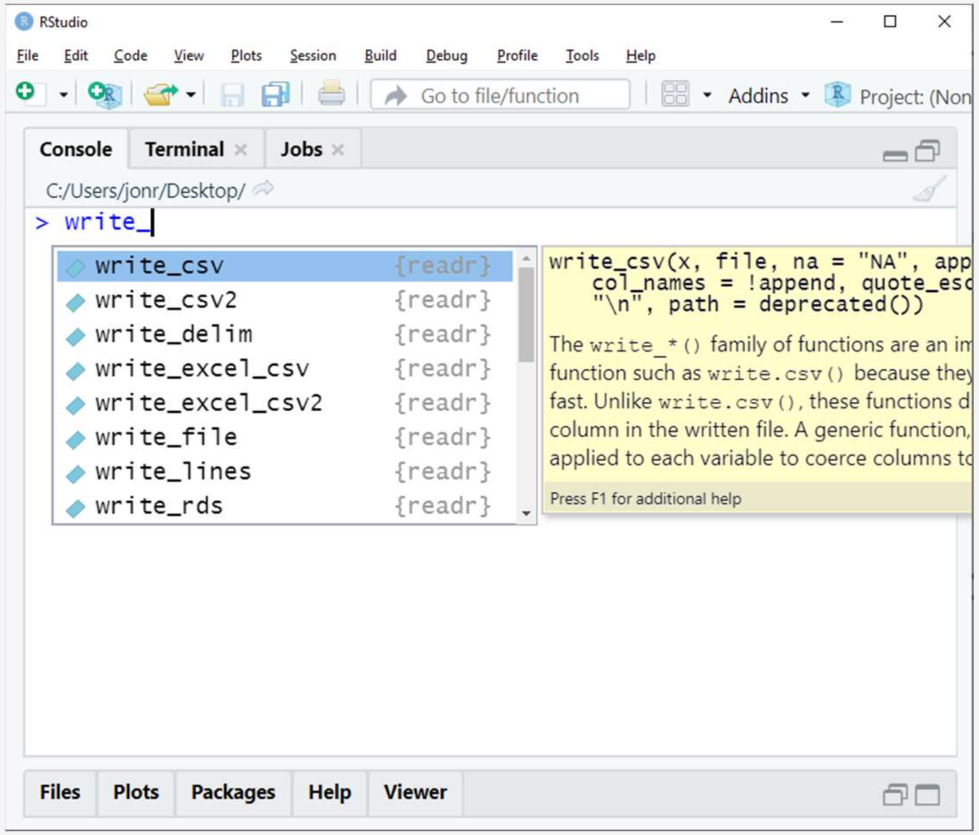
```
read_csv(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na
= TRUE, quote = "\"", comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
<...truncated...> )
```

```
> read_csv(file = "", col_names = )
```

Notice the pop-up help menu. The different arguments are shown, with default values.

Getting data out of R

The readr package also comes with complementary write functions that can write files in different formats.



```
> write_
write_csv          {readr}
write_csv2         {readr}
write_delim        {readr}
write_excel_csv    {readr}
write_excel_csv2   {readr}
write_file         {readr}
write_lines        {readr}
write_rds          {readr}
```

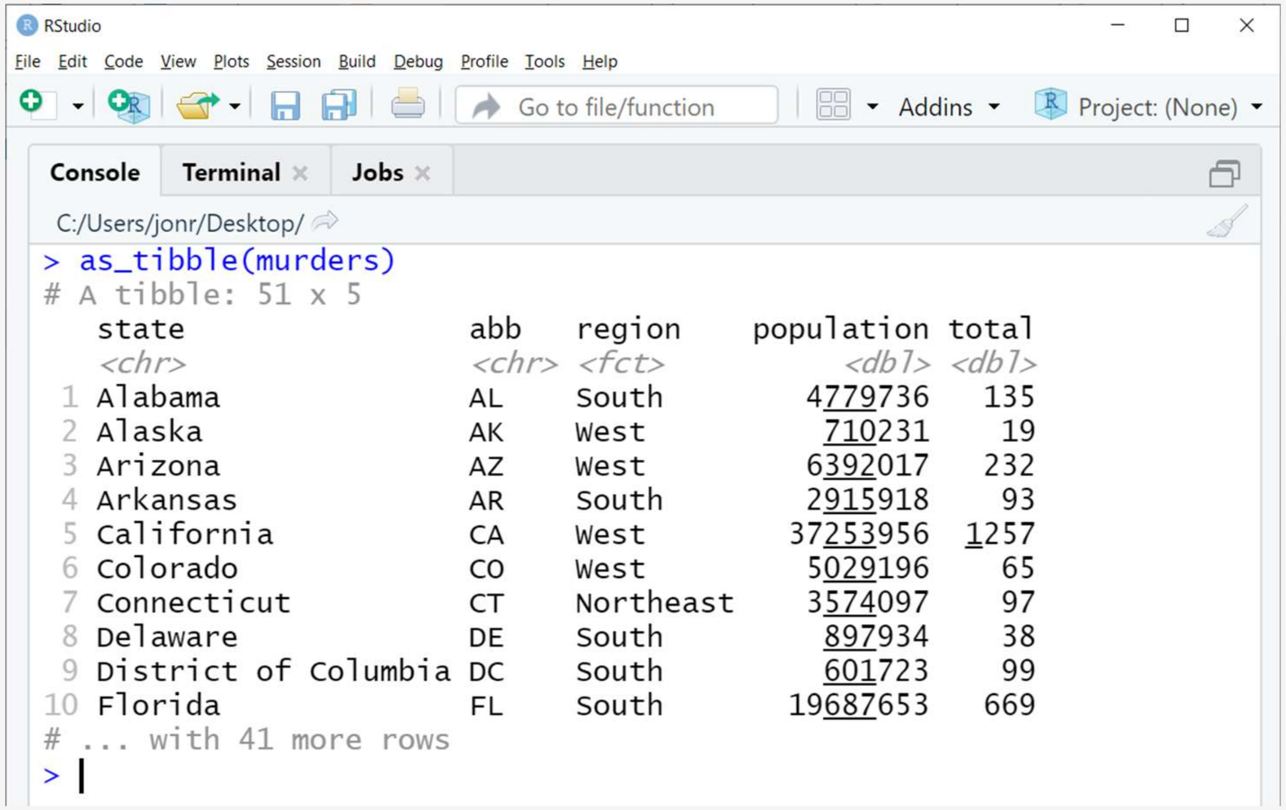
```
write_csv(x, file, na = "NA", app
    col_names = !append, quote_esc
    "\n", path = deprecated())
```

The write_*() family of functions are an im
function such as write.csv() because they
fast. Unlike write.csv(), these functions d
column in the written file. A generic function,
applied to each variable to coerce columns to

Press F1 for additional help

Tibbles

A tibble is a special kind of data frame. Tibbles are the preferred format in the tidyverse and most tidyverse operations result in a tibble. Tibbles also display better when printed in R.



```
> as_tibble(murders)
# A tibble: 51 x 5
   state                abb   region    population total
   <chr>                <chr> <fct>          <dbl> <dbl>
 1 Alabama              AL    South        4779736   135
 2 Alaska               AK    West          710231    19
 3 Arizona              AZ    West         6392017   232
 4 Arkansas             AR    South        2915918    93
 5 California           CA    West        37253956  1257
 6 Colorado             CO    West         5029196    65
 7 Connecticut          CT    Northeast    3574097    97
 8 Delaware             DE    South         897934    38
 9 District of Columbia DC    South         601723    99
10 Florida              FL    South       19687653   669
# ... with 41 more rows
> 
```

Do Exercise 2

Manipulating rectangular data with the dplyr package

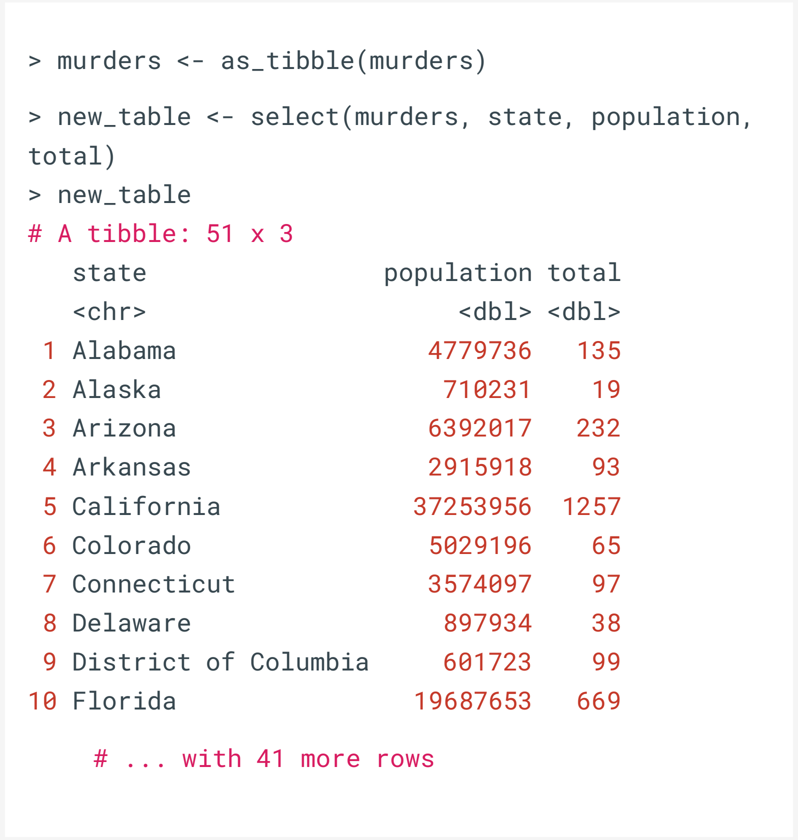
The dplyr package

The **dplyr** package of the tidyverse has functions for doing some of the most common operations when working with data frames. For example:

```
mutate() # adds new variables by manipulating existing variables
select() # picks variables based on their names.
filter() # picks cases based on their values.
summarise() # reduces multiple values down to a single summary.
arrange() # changes the ordering of the rows.
group_by() # perform operations “by group”
```

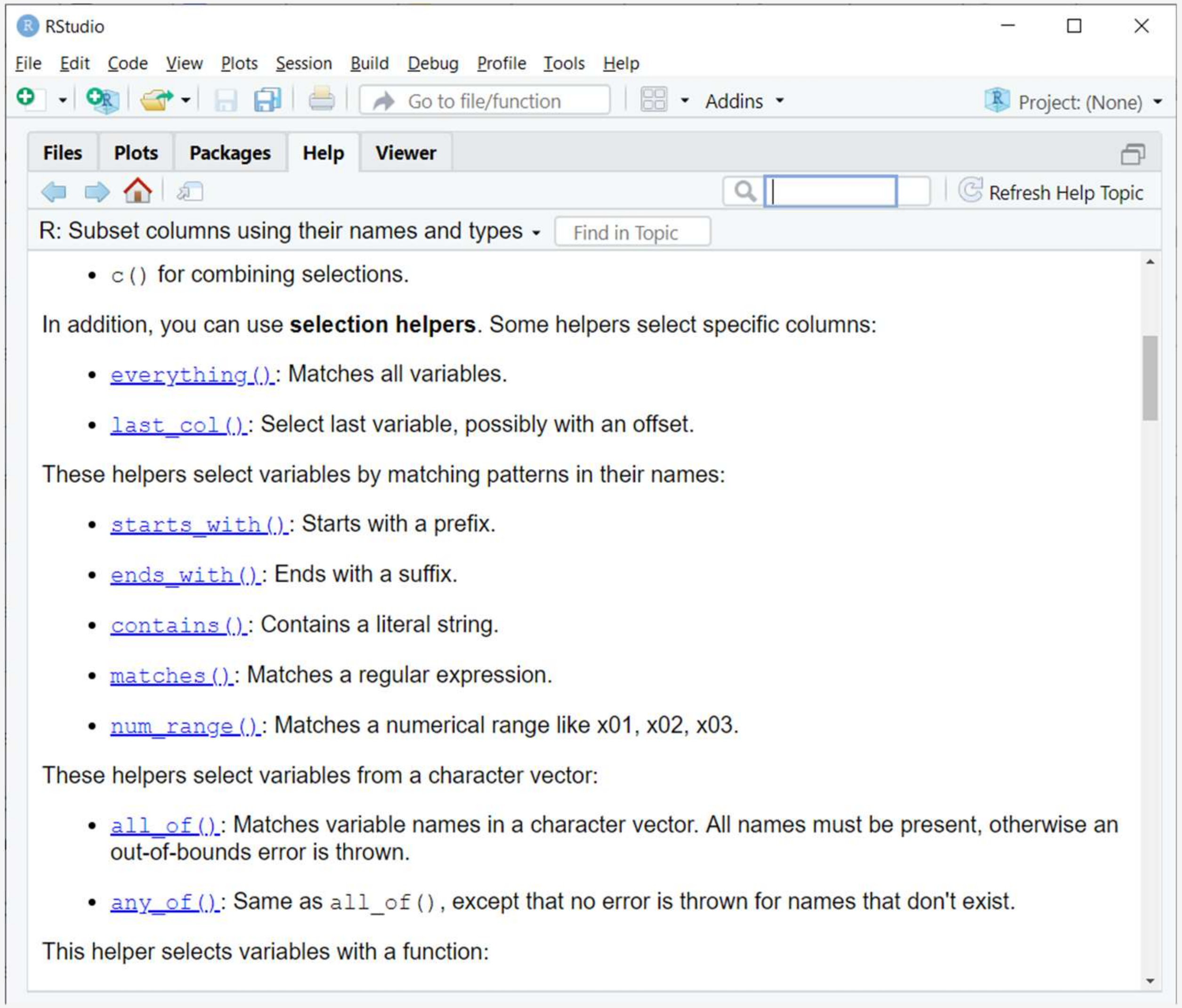
Selecting columns with **select()**

select() allows you to select different columns based on a wide range of different criteria. Check the cheat sheet or the help pages for all the options.



```
> murders <- as_tibble(murders)

> new_table <- select(murders, state, population,
total)
> new_table
# A tibble: 51 x 3
   state                population total
   <chr>                     <dbl> <dbl>
 1 Alabama                 4779736   135
 2 Alaska                   710231    19
 3 Arizona                 6392017   232
 4 Arkansas                2915918    93
 5 California             37253956  1257
 6 Colorado                5029196    65
 7 Connecticut             3574097    97
 8 Delaware                 897934    38
 9 District of Columbia     601723    99
10 Florida                19687653   669

    # ... with 41 more rows
```

Selecting columns with **select()**


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Files Plots Packages Help Viewer
Refresh Help Topic
R: Subset columns using their names and types
Find in Topic

- `c()` for combining selections.

In addition, you can use **selection helpers**. Some helpers select specific columns:

- `everything()`: Matches all variables.
- `last_col()`: Select last variable, possibly with an offset.

These helpers select variables by matching patterns in their names:

- `starts_with()`: Starts with a prefix.
- `ends_with()`: Ends with a suffix.
- `contains()`: Contains a literal string.
- `matches()`: Matches a regular expression.
- `num_range()`: Matches a numerical range like x01, x02, x03.

These helpers select variables from a character vector:

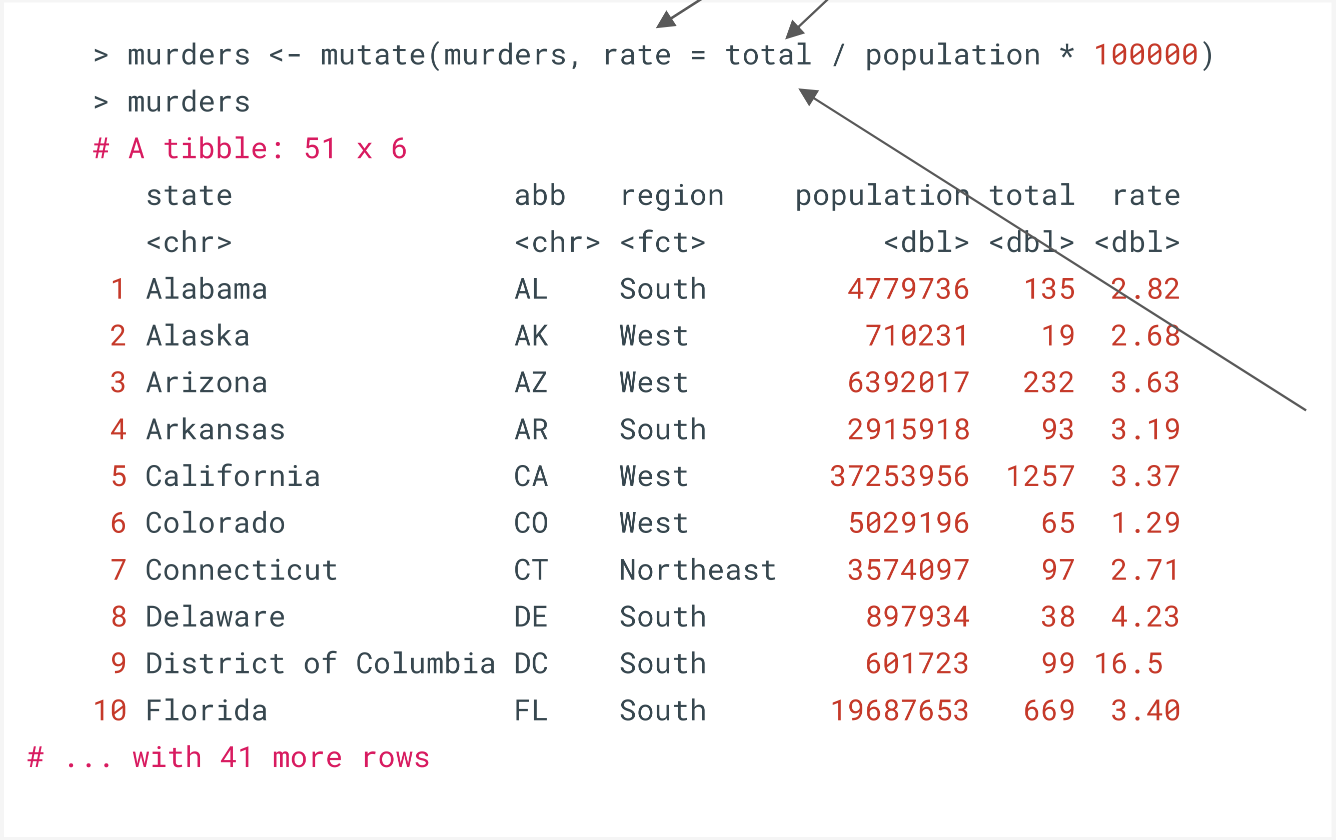
- `all_of()`: Matches variable names in a character vector. All names must be present, otherwise an out-of-bounds error is thrown.
- `any_of()`: Same as `all_of()`, except that no error is thrown for names that don't exist.

This helper selects variables with a function:

Adding columns with **mutate()**

total and *population* are columns in the data. *rate* is created by mutate()

mutate() allows to add a column by doing operations on other columns in the data frame.



```
> murders <- mutate(murders, rate = total / population * 100000)
> murders
# A tibble: 51 x 6
   state                abb   region    population total  rate
   <chr>                <chr> <fct>          <dbl> <dbl> <dbl>
 1 Alabama              AL    South        4779736   135  2.82
 2 Alaska               AK    West          710231    19  2.68
 3 Arizona              AZ    West         6392017   232  3.63
 4 Arkansas             AR    South        2915918    93  3.19
 5 California           CA    West        37253956  1257  3.37
 6 Colorado             CO    West         5029196    65  1.29
 7 Connecticut          CT    Northeast    3574097    97  2.71
 8 Delaware             DE    South         897934    38  4.23
 9 District of Columbia DC    South         601723    99 16.5
10 Florida              FL    South       19687653   669  3.40
# ... with 41 more rows
```

Subsetting rows with **filter()**

filter() allows to select rows based on various criteria. E.g. select states with murder rate below or equal to 0.7.

```
> filter(murders, rate <= 0.7)
# A tibble: 5 x 6
  state         abb   region        population total  rate
  <chr>         <chr> <fct>              <dbl> <dbl> <dbl>
1 Hawaii        HI    West             1360301     7 0.515
2 Iowa          IA    North Central    3046355    21 0.689
3 New Hampshire NH    Northeast        1316470     5 0.380
4 North Dakota  ND    North Central     672591     4 0.595

5 Vermont       VT    Northeast         625741     2 0.320
```

The “pipe”

NB! The “pipe” is not part of base R, but needs to be activated by loading a package (e.g. library(tidyverse)).

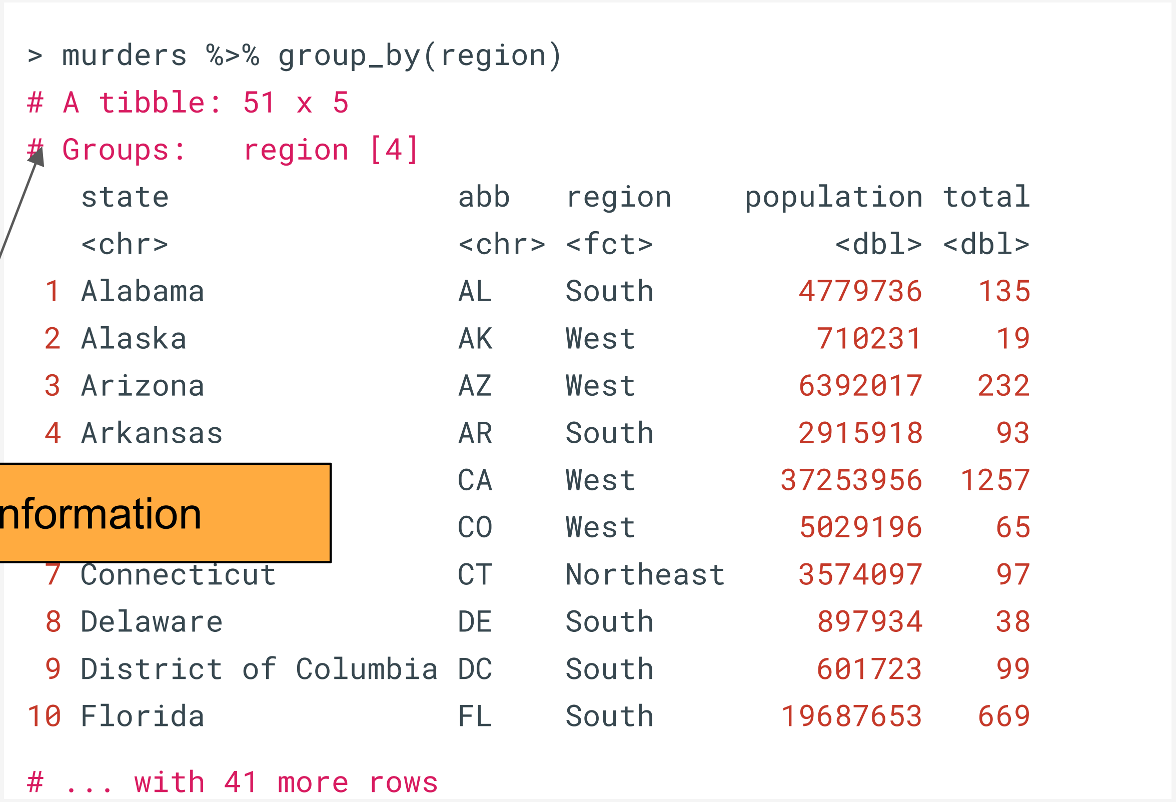
Just like “|” in unix/bash, the %>% (NB: look for the RStudio shortcut) symbol allows you to chain operations together. The pipe is particularly useful when using “tidyverse-style” functions (you will learn about that soon).

```
> murders %>% mutate(rate = total / population * 100000) %>%
filter(rate <= 0.7)
# A tibble: 5 x 6
  state          abb   region         population total  rate
  <chr>          <chr> <fct>               <dbl> <dbl> <dbl>
1 Hawaii         HI    West              1360301     7 0.515
2 Iowa           IA    North Central     3046355    21 0.689
3 New Hampshire  NH    Northeast         1316470     5 0.380
                a ND    North Central      672591     4 0.595
                 VT    Northeast          625741     2 0.320
```

Notice how the data object is no longer the first argument in the mutate() and filter() functions.

group_by()

group_by() allows you to split the data into groups and perform operations on each group.



```
> murders %>% group_by(region)
# A tibble: 51 x 5
# Groups:   region [4]
   state                abb   region    population total
   <chr>                <chr> <fct>          <dbl> <dbl>
 1 Alabama              AL    South        4779736   135
 2 Alaska               AK    West          710231    19
 3 Arizona              AZ    West         6392017   232
 4 Arkansas             AR    South        2915918    93
                        CA    West        37253956  1257
                        CO    West         5029196    65
 7 Connecticut          CT    Northeast    3574097    97
 8 Delaware             DE    South         897934    38
 9 District of Columbia DC    South         601723    99
10 Florida              FL    South       19687653   669

# ... with 41 more rows
```

Notice the new Groups information

group_by(), then summarize

The function summarize() works particularly well on grouped data frames. Summarize can be used to quickly generate descriptive statistics.

```
> murders %>% group_by(region) %>%
summarize(count = n())
# A tibble: 4 x 2
  region        count
* <fct>         <int>
1 Northeast         9
2 South            17
3 North Central    12
4 West             13
```

group_by(), then summarize

The function summarize() works particularly well on grouped data frames.
Summarize can be used to quickly generate descriptive statistics.

```
> murders %>% mutate(rate = total / population * 100000)
%>% group_by(region) %>% summarize(median_rate =
median(rate)) %>% filter(median_rate < 2.0)
# A tibble: 3 x 2
  region        median_rate
  <fct>               <dbl>
1 Northeast            1.80
2 North Central        1.97
3 West                 1.29
```

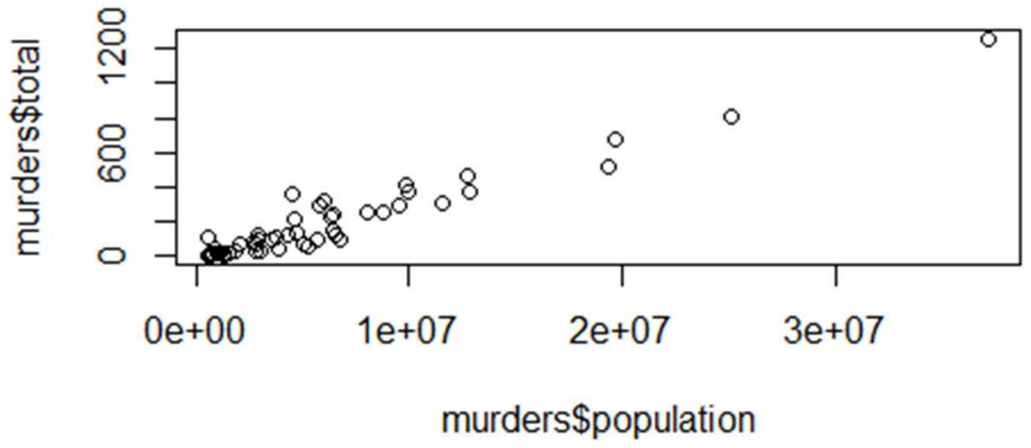
Do Exercise 3

Basic plotting

Basic plotting in R - scatterplot

R has several functions for making plots to quickly visualize your data. The **plot()** function can plot two variables against each other. plot() takes two arguments, x = and y = .

```
> plot(murders$population, murders$total)
```

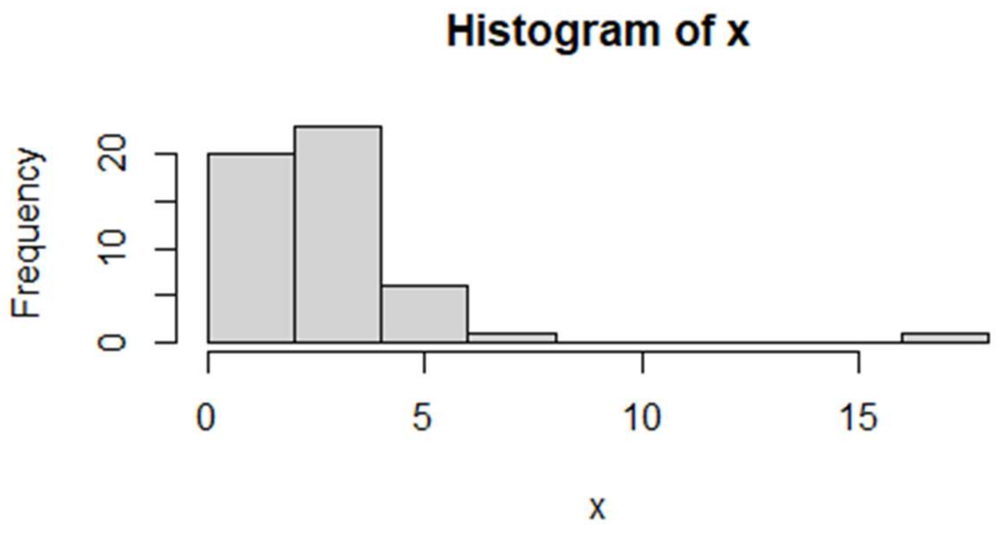


The "$" is called "the accessor" and is the way to access the different variables (columns) in data frames in base-R language.

Basic plotting in R - histogram

The **hist()** function is a quick method to get a summary of your data.

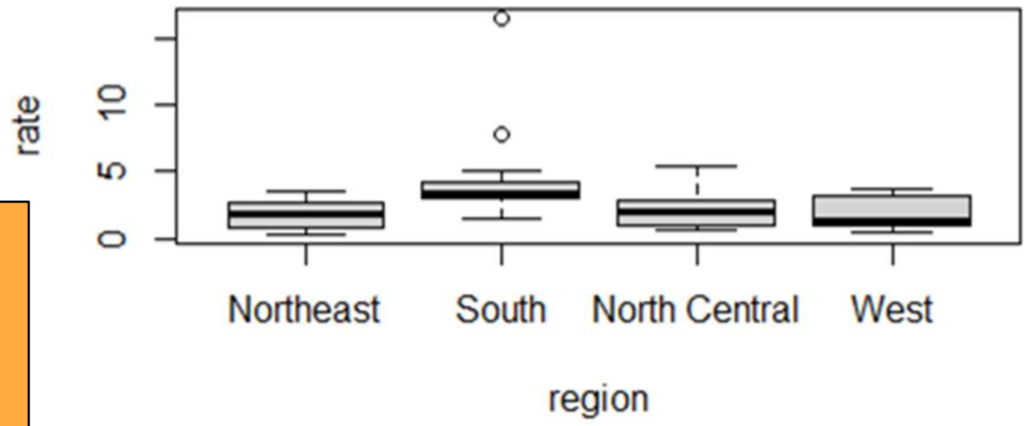
```
> x <- murders$total / murders$population*100000
> x <- hist(x)
```



Basic plotting in R - boxplot

The **boxplot()** function is great for quickly comparing groups of data.

```
murders <- mutate(murders, rate = total / population * 100000)
boxplot(rate~region, data = murders)
```



The “~” symbol (tilde) is used here to design a “formula”. The formula tells R to calculate statistics of the “rate” (e.g. median) across the different “regions”.

Do Exercise 4