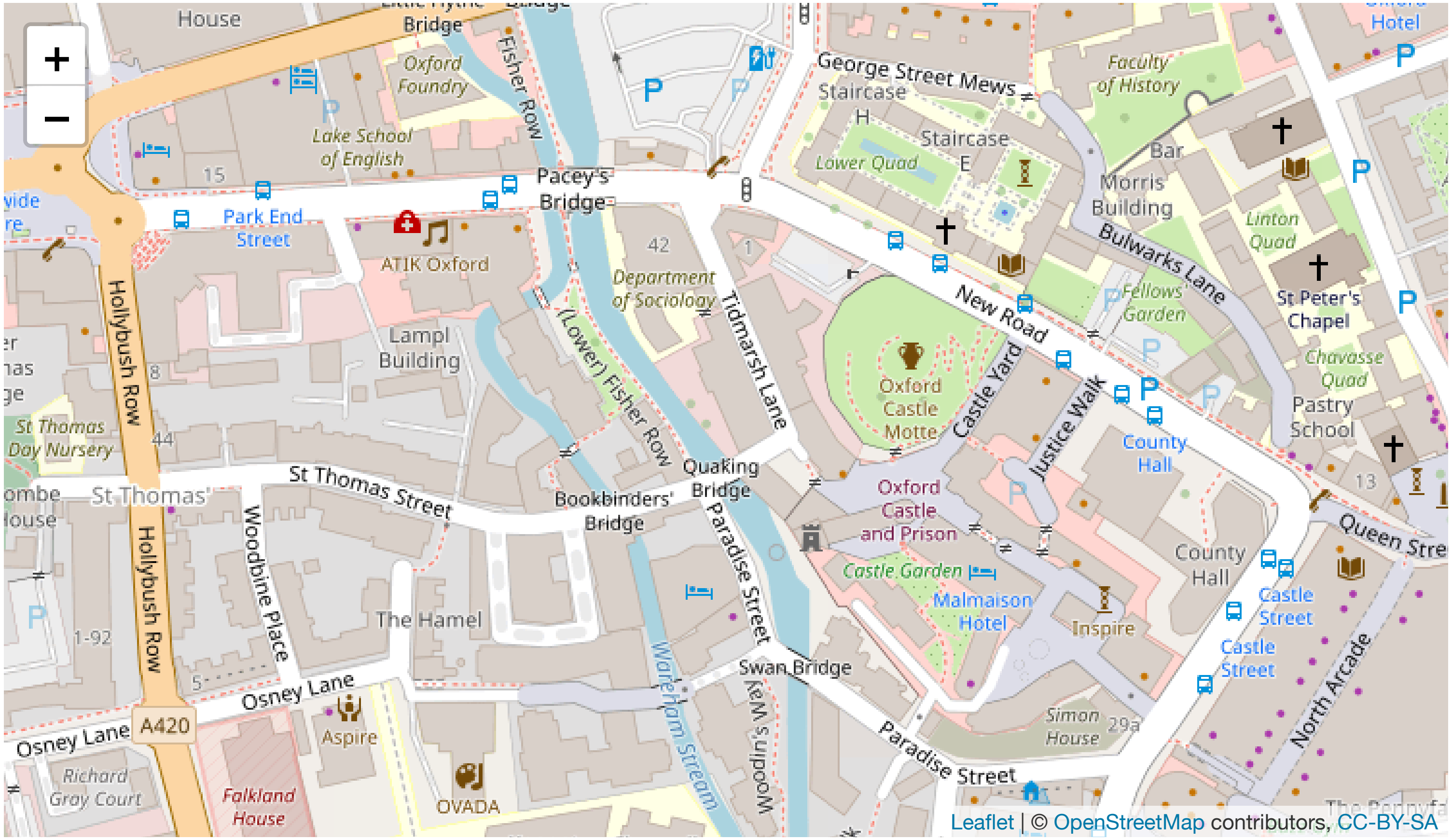
Introduction to R

Arun Frey

Department of Sociology, University of Oxford

07.11.2021

```r
library(leaflet)
leaflet() %>% addTiles() %>% setView(-1.264, 51.752, zoom = 17)
```

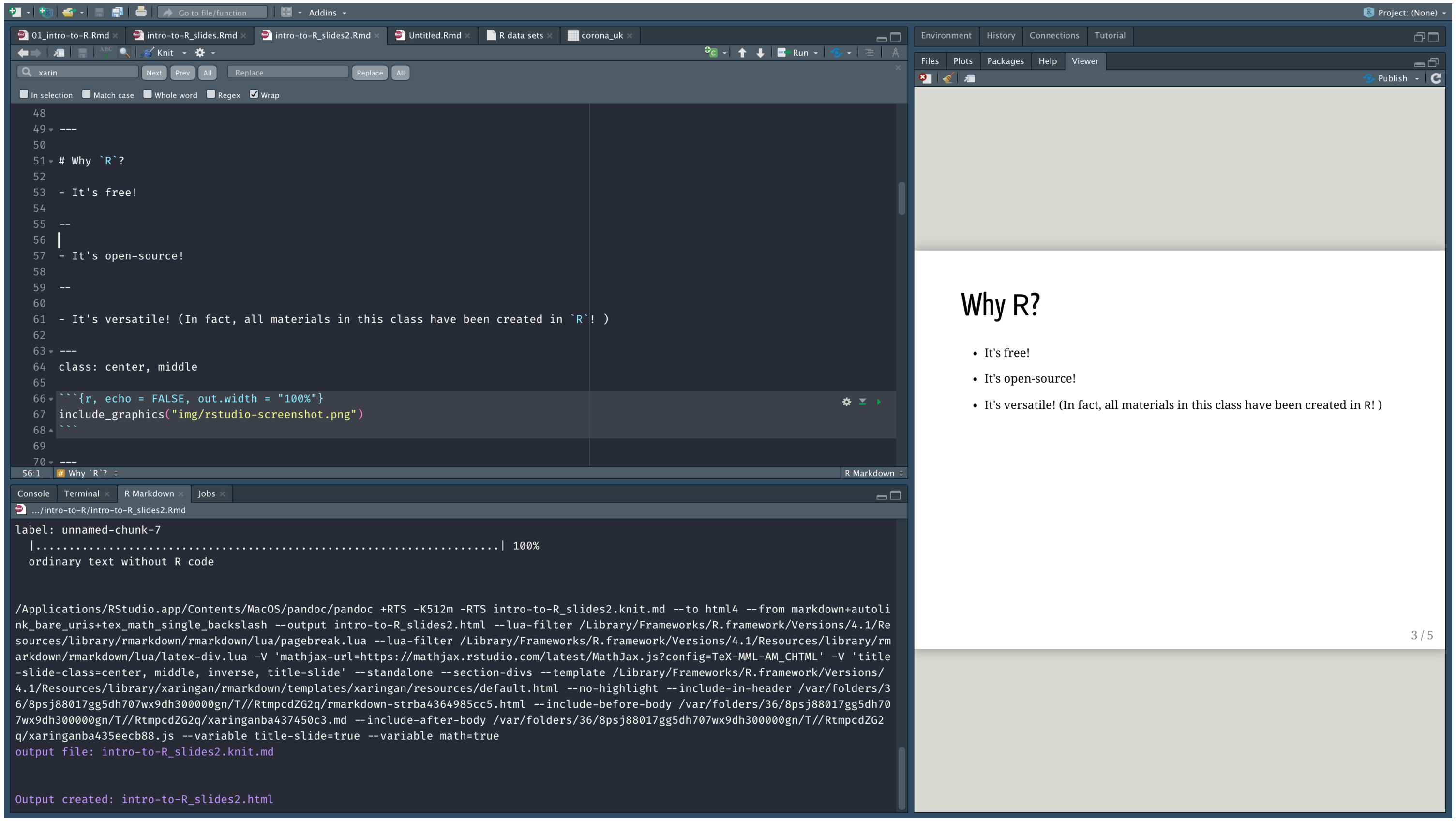


What is R?

- R is a free statistical programming software. You can download R here.
- In this course, we will be running R in the RStudio IDE.

Why R?

- It's free!
- It's open-source!
- It's versatile! (In fact, all materials in this class have been created in `R`!)



~~~

Why `R`?

- It's free!

--

- It's open-source!

--

- It's versatile! (In fact, all materials in this class have been created in `R`!)

---
class: center, middle

```{r, echo = FALSE, out.width = "100%"}
include_graphics("img/rstudio-screenshot.png")
~~~

---
```

```
label: unnamed-chunk-7
  |.....................................................................| 100%
  ordinary text without R code

/Applications/RStudio.app/Contents/MacOS/pandoc/pandoc +RTS -K512m -RTS intro-to-R_slides2.knit.md --to html4 --from markdown+autoli
nk_bare_uris+tex_math_single_backslash --output intro-to-R_slides2.html --lua-filter /Library/Frameworks/R.framework/Versions/4.1/Re
sources/library/rmarkdown/rmarkdown/lua/pagebreak.lua --lua-filter /Library/Frameworks/R.framework/Versions/4.1/Resources/library/rm
arkdown/rmarkdown/lua/latex-div.lua -V 'mathjax-url=https://mathjax.rstudio.com/latest/MathJax.js?config=TeX-MML-AM_CHTML' -V 'title
-slide-class=center, middle, inverse, title-slide' --standalone --section-divs --template /Library/Frameworks/R.framework/Versions/
4.1/Resources/library/xaringan/rmarkdown/templates/xaringan/resources/default.html --no-highlight --include-in-header /var/folders/3
6/8psj88017gg5dh707wx9dh300000gn/T//RtmpcdZG2q/rmarkdown-strba4364985cc5.html --include-before-body /var/folders/36/8psj88017gg5dh70
7wx9dh300000gn/T//RtmpcdZG2q/xaringanba437450c3.md --include-after-body /var/folders/36/8psj88017gg5dh707wx9dh300000gn/T//RtmpcdZG2
q/xaringanba435eecb88.js --variable title-slide=true --variable math=true
output file: intro-to-R_slides2.knit.md

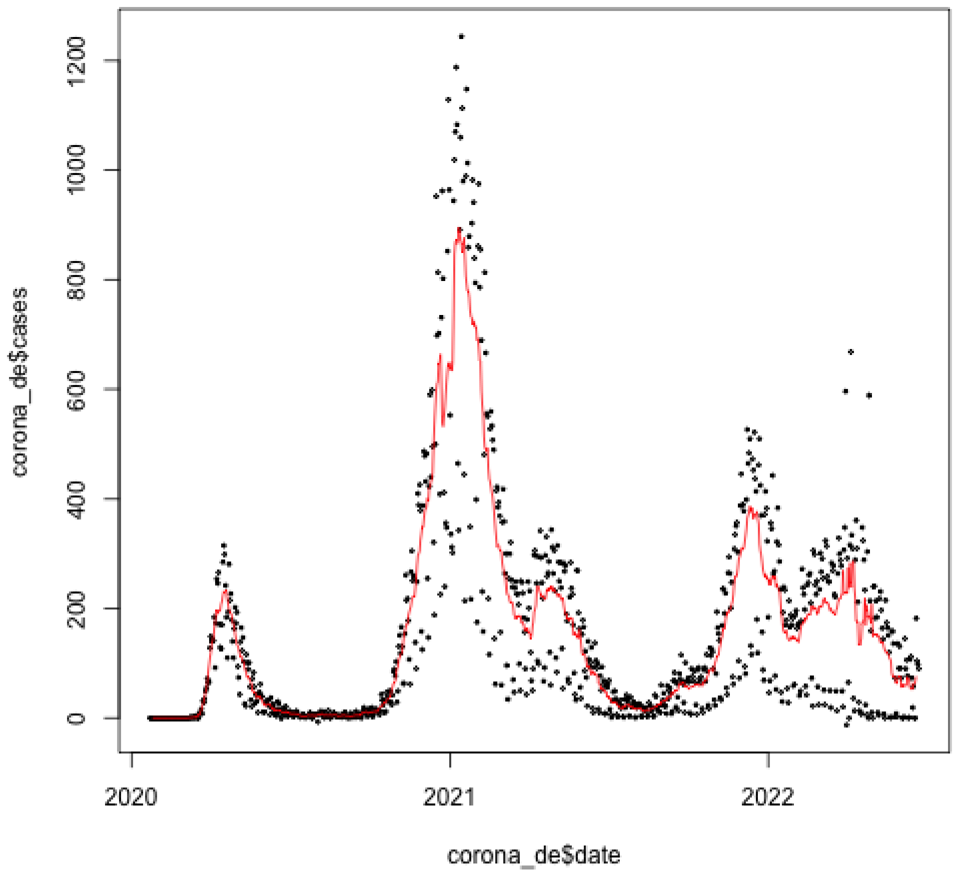
Output created: intro-to-R_slides2.html
```

Why R?

- It's free!
- It's open-source!
- It's versatile! (In fact, all materials in this class have been created in R!)

3 / 5
```

## This course

- This course is designed to introduce you to the basics of `R` programming.
- By the end, you will know how to:
    1. Generate and transform numeric, logical, and character vectors
    2. Deal with missing values
    3. Load and inspect data
    4. Generate descriptives

Go to Github to download this course: https://github.com/ArunFrey/intro-to-R

```r
library(coronavirus)

corona_de <- coronavirus[coronavirus$country=="Germany" &
                           coronavirus$type=="death", ]

corona_de$death_7 <-  zoo::rollmean(corona_de$cases, k = 7, fill = NA

plot(x = corona_de$date,
     y = corona_de$cases,
     cex = 0.3)

lines(x = corona_de$date,
     y = corona_de$death_7,
     type = "l",
     cex = 1.5,
     col = "red")
```


corona_de$cases
0
200
400
600
800
1000
1200
2020
2021
2022
corona_de$date

If you want to dive deeper

This course only scratches the surface of what you can do in `R`.

For more elaborate introductions and more advanced guides, see the following (free!) books:

- YaRrr! The Pirate's Guide to R by _Nathaniel D. Phillips
    - This book is a great introduction to base R, and lots of material I draw from in this presentation is based on the book!
- R for Data Science by Gareth Golemund and Hadley Wickham
    - This book is fantastic if you want to learn about the `tidyverse` and tidy R, which is a different way of writting R code than base R.
- Advanced R by Hadley Wickham
    - Ths book is good for those who want to get a really in-depth understanding of how `R` operates.

If you want to dive deeper

The `R` community is very welcoming and inclusive. If you are feeling stuck, chances are someone has had the same issue before.

Here are some helpful resources and great groups to join:

- http://www.google.com
- http://www.r-bloggers.com
- http://www.stackoverflow.com
- https://rladies.org
- https://www.rstudio.com/resources/cheatsheets/ (list of cheatsheets)

Installing R and RStudio

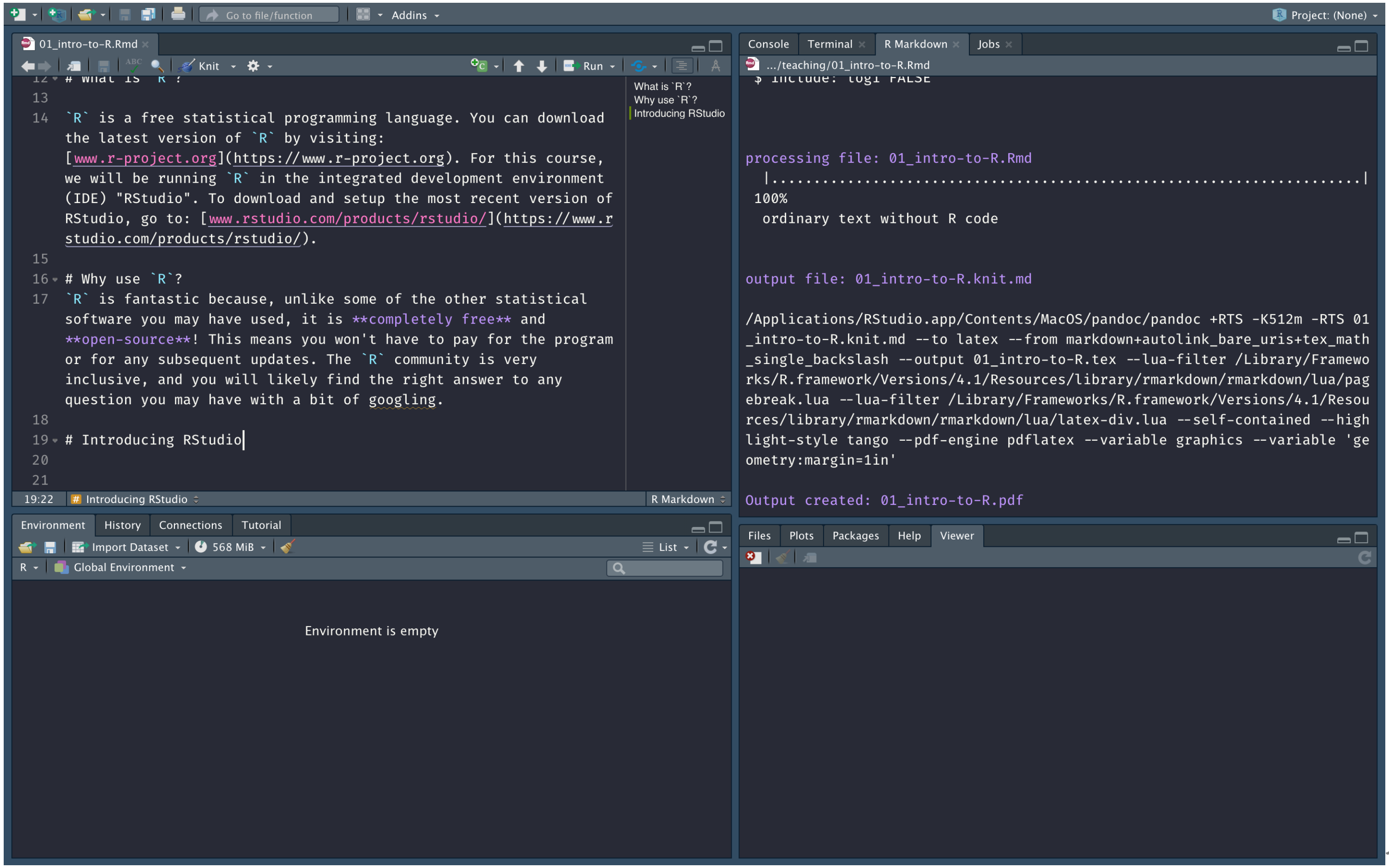
- Download and install R from here https://cloud.r-project.org
- For Macs, you may have to download different versions:
    - Intel chip (R-4.1.1.pkg)
    - Apple Silicon M1 chip (R-4.1.1-arm64.pkg)
- Download RStudio here: https://www.rstudio.com/products/rstudio/download/

In this course we will be using R exclusively through RStudio.

Download and install R and RStudio

Using RStudio

When you open RStudio it should look something like this:



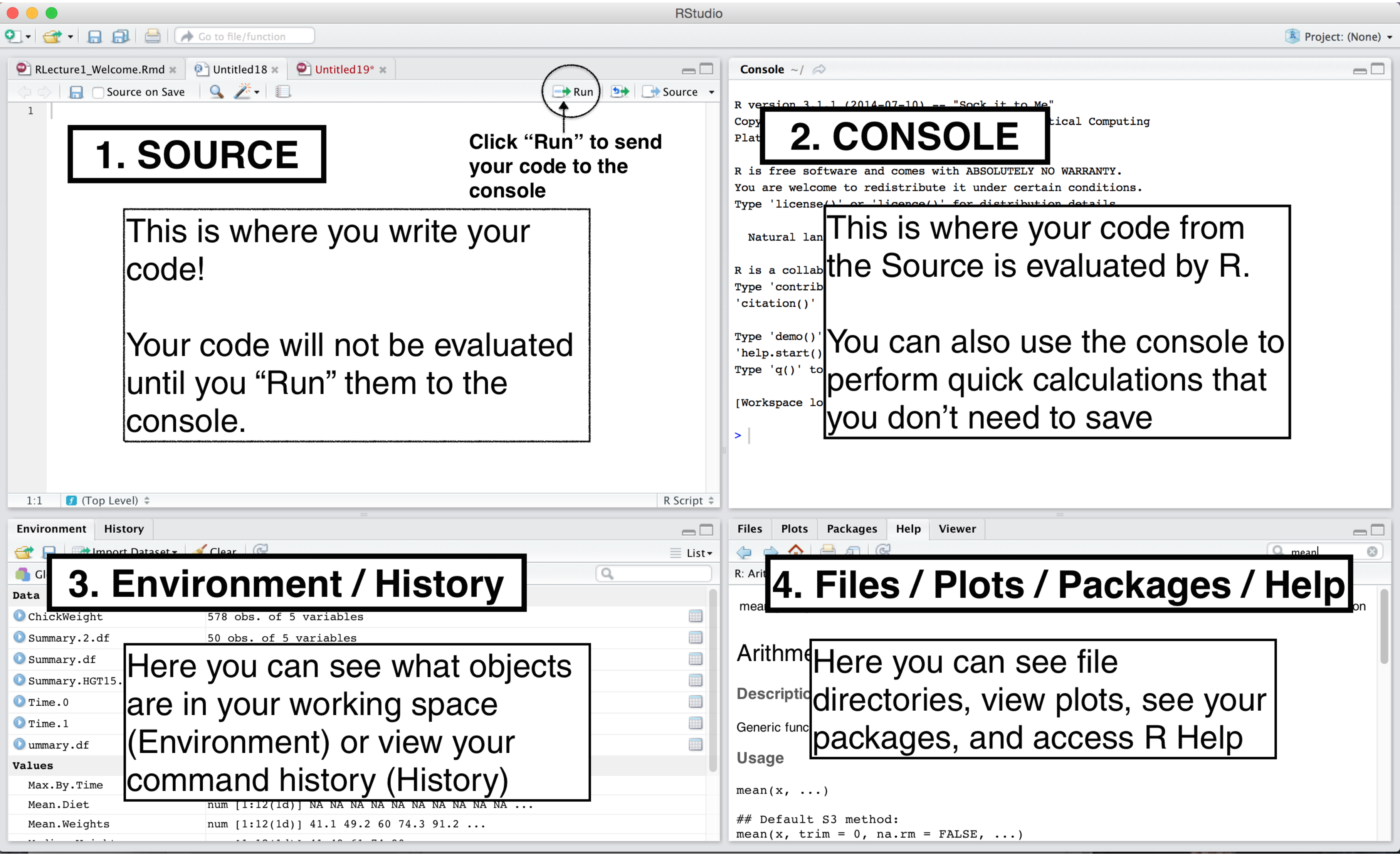
```
# What is `R`?

`R` is a free statistical programming language. You can download the latest version of `R` by visiting: [www.r-project.org](https://www.r-project.org). For this course, we will be running `R` in the integrated development environment (IDE) "RStudio". To download and setup the most recent version of RStudio, go to: [www.rstudio.com/products/rstudio/](https://www.rstudio.com/products/rstudio/).

# Why use `R`?
`R` is fantastic because, unlike some of the other statistical software you may have used, it is **completely free** and **open-source**! This means you won't have to pay for the program or for any subsequent updates. The `R` community is very inclusive, and you will likely find the right answer to any question you may have with a bit of googling.

# Introducing RStudio
```

```
$ include: logi FALSE



processing file: 01_intro-to-R.Rmd
  |.....................................................................|
 100%
  ordinary text without R code


output file: 01_intro-to-R.knit.md

/Applications/RStudio.app/Contents/MacOS/pandoc/pandoc +RTS -K512m -RTS 01
_intro-to-R.knit.md --to latex --from markdown+autolink_bare_uris+tex_math
_single_backslash --output 01_intro-to-R.tex --lua-filter /Library/Framewo
rks/R.framework/Versions/4.1/Resources/library/rmarkdown/rmarkdown/lua/pag
ebreak.lua --lua-filter /Library/Frameworks/R.framework/Versions/4.1/Resou
rces/library/rmarkdown/rmarkdown/lua/latex-div.lua --self-contained --high
light-style tango --pdf-engine pdflatex --variable graphics --variable 'ge
ometry:margin=1in'

Output created: 01_intro-to-R.pdf
```

Using RStudio

RStudio is an integrated development environment specifically developed for `R`, that lets you write code, run scripts, and view the results all in one.

- **Source**: This is the code editor, where you write and save your code.
- **Console**: This is where the output of your code will be printed.
- **Environment/History**: This is where any objects, such as vectors, matrices, or dataframes, will be stored.
- **Viewer**: This viewer previews any plots you create. You can also check your folder files and call for help here.



1. SOURCE

Click "Run" to send your code to the console

This is where you write your code!

Your code will not be evaluated until you "Run" them to the console.

2. CONSOLE

```
R version 3.1.1 (2014-07-10) -- "Sock it to Me"
Copy                                         tical Computing
Plat

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 lan

R is a collab
Type 'contrib
'citation()'

Type 'demo()'
'help.start()
Type 'q()' to

[Workspace lo

>
```

This is where your code from the Source is evaluated by R.

You can also use the console to perform quick calculations that you don't need to save

3. Environment / History

```
Data
ChickWeight        578 obs. of 5 variables
Summary.2.df       50 obs. of 5 variables
Summary.df
Summary.HGT15.
Time.0
Time.1
ummary.df
Values
Max.By.Time
Mean.Diet          num [1:12(1d)] NA NA NA NA NA NA NA NA NA NA ...
Mean.Weights       num [1:12(1d)] 41.1 49.2 60 74.3 91.2 ...
```

Here you can see what objects are in your working space (Environment) or view your command history (History)

4. Files / Plots / Packages / Help

```
R: Arit
mea                                                    on

Arithme
Descriptio
Generic func
Usage

mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Here you can see file directories, view plots, see your packages, and access R Help

Tips for using RStudio

Always write code into the source code file, except for small checks and tests.

To execute the line of source code where the cursor currently resides you can press the **Ctrl + Enter/Cmd + Enter key**, rather than manually pressing the Run toolbar button.

Annotate your code using #

```
# Introduction to R
# 07.09.2021

# Use head(x, n = 2) to see the first two rows of a dataframe
head(mtcars, n = 2)
##               mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

Send code from the source to the console

Using packages

R comes with a list of built in functions, but often you will want to use other functions written not by the original creators of `R`, but by other people.

If you want to use functions written by other people, you have to install it as a `package`.

To do this, we have to first install the package once, and then load it whenever we would like to use it.

```
# install the package (you only need to do this once)
install.packages('praise')
```

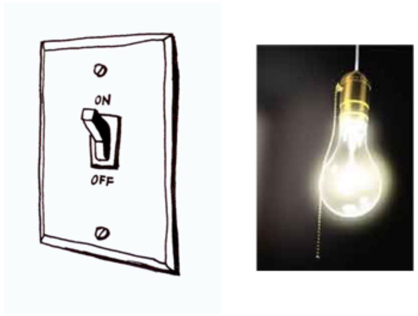
```
# load the package to use its functions
library(praise)
praise()
## [1] "You are primo!"
```

Installing a package

```
install.packages('my.package')
```

Loading a package

```
library('mypackage')
```



Credit: YaRrr! The Pirate's Guide to R

Using packages

Sometimes you only need to use a very specific function of a package one time, and loading the entire package may seem unnecessary.

You can use `package::function` to call the function you are after. This tells `R` to only load the package for this specific chunk of code.

```
cowsay::say("Welcome to the course!", by = "cow")
## 
##  ----- 
## Welcome to the course! 
##  ------ 
##     \   ^__^ 
##      \  (oo)\ ________ 
##         (__)\         )\ /\ 
##              ||------w|
##              ||      ||
```

Feeling stuck?

You can use `?` whenever you want to read the documentation of a particular command.

```
# how should I specify the mean and standard deviation of a normal d
?rnorm
# how does a histogram work
?hist
# how does the mean() function work
?mean
```

Introducing the Basics

What we'll cover today

Introducing the Basics

1. Object types
2. Vectors
3. Missingness
4. Vector functions
5. Dataframes
6. Loading data

Objects and functions

Almost everything in `R` is either an object or a function.

- **Object**: number, vector, dataset, summary statistic, regression model, etc.
- **Function**: takes objects as arguments, does something, and returns an object.

```r
# Create a vector object called height
height <- c(189, 178, 166, 178, 190)

# apply the mean() function to the object height
mean(height)
## [1] 180.2
```

→ The function `mean()` takes the object `height`, calculates the average, and returns a single number.

Objects and functions

When you use `R`, you will mostly:

1. Define objects
2. Apply functions to those objects
3. Repeat!

R as a calculator

```
3+5
## [1] 8

10/2
## [1] 5

sqrt(4)
## [1] 2
```

```

"Hello world!"
## [1] "Hello world!"

"1" + "3"
## Error in "1" + "3": non-numeric argument to binary operator
```

The <- operator

You can assign values to variables using the `<-` operator. You can then use the variable in subsequent operations.

```
x <- 9 + 11
x
## [1] 20
y <- x / 2
y
## [1] 10
```

```
greetings <- "Hello world!"
greetings
## [1] "Hello world!"
```

Quiz

Just by looking at the code, what do each of the following lines return?

```
12 - 2
#A:

x <- 12 - 2

y <- x * 2
y

y/2
y

z <- "1 + 2"
z

z + 3
```

Vectors

We can create longer vectors by using `c()` (read: concatenate).

Numeric vectors

```
w <- 2
y <- c(1, 2, 3)
z <- c(4, 5, 6)
z
## [1] 4 5 6
```

Character vectors

```
welcome <- c("Welcome", "to", "this", "course!")
welcome
## [1] "Welcome" "to"      "this"    "course!"
```

Vectors

For longer vectors, writing out each element can be tedious. In addition to `c()`, there are other options.

Function	**Example**	**Result**
`c(a, b, ...)`	`c(1, 5, 9)`	1, 5, 9
`a:b`	`1:5`	1, 2, 3, 4, 5
`seq(from, to, by, length.out)`	`seq(from = 0, to = 6, by = 2)`	0, 2, 4, 6
`rep(x, times, each, length.out)`	`rep(c(7, 8), times = 2, each = 2)`	7, 7, 8, 8, 7, 7, 8, 8

Logical vectors

While numeric vectors can include any number and character values any character string, logical vectors can only take the values of either `TRUE` or `FALSE`.

Logical vectors are therefore often used to distinguish between two groups, or select a certain subset of variables.

In the example below, we create a logical vector that distinguishes between ages below and above the age of 18.

```r
age <- c(14, 19, 23, 13, 16, 19, 18)

is_18 <- age >= 18
is_18
## [1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
```

The vector `is_18` is `TRUE` when age is 18 or higher, and `FALSE` otherwise.

Logical operators

In the previous example we use >= to distinguish between ages below and above Some logical operators include:

Operator	Description
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	exactly equal to
`!=`	not equal to
`!x`	Not x
`x \& y`	x AND y
`isTRUE(x)`	test if X is TRUE

Missing values

When we deal with data in the real world, there is often lots of missingness.

Missing values are denoted with `NA`.

`NA`s behave differently to other values.

```
num_vec <- c(5, NA, 15, 20, 25, NA)
num_vec / 5
## [1]  1 NA  3  4  5 NA

NA + 3
## [1] NA

c("hello", "my", "name", "is", NA)
## [1] "hello" "my"    "name"  "is"    NA
```

Complete Exercise 1 on the worksheet

Vector functions

Length: Checks the length of a vector

```
x <- 2
y <- c(0.5, 45, 7, 45, 0.5)
z1 <- c("1 2 3 4 5 ", "6 7 8")
length(x)
## [1] 1
length(y)
## [1] 5
length(z)
## [1] 3
```

Sorting/unique: sorts or displays the unique values of a vector

```
sort(y)
## [1]  0.5  0.5  7.0 45.0 45.0

unique(y)
## [1]  0.5 45.0  7.0
```

Numeric vector functions

Function	Example	Result
`sum(x)`, `product(x)`	`sum(1:10)`	55
`min(x)`, `max(x)`	`min(1:10)`	1
`mean(x)`, `median(x)`	`mean(1:10)`	5.5
`sd(x)`, `var(x)`, `range(x)`	`sd(1:10)`	3.0276504
`summary(x)`	`summary(1:10)`	`Min = 1.00. 1st Qu. = 3.25, Median = 5.50, Mean = 5.50, 3rd Qu. = 7.75, Max = 10.0`

Quiz

Copy the following two vectors:

```r
age <- c(22, 24,25, 25, 22, 21, 28, 23, 24, 27)
welcome <- c("Welcome", "to", "this", "course!")
```

Use `R` to generate:

1. The unique values in `age`.
2. The length of welcome
3. The mean of age.
4. Round the mean of age to 0 decimals. *Hint*: You can use the `round` function, and see how it works using `?round`.
5. Can you compute the mean of `welcome`. Why/Why not?

Character vector functions

If you look at the output of `welcome`, each word is included in separate quotation marks.

```
welcome
## [1] "Welcome" "to"       "this"     "course!"
```

This is because `R` treats each of these words as a separate element in a vector.

There are times when we might want to tell `R` to collapse the string into a single element. We can do this using the `paste()` function, specifying the option `collapse = TRUE`.

```
welcome2 <- paste(welcome, collapse = " ")
welcome2
## [1] "Welcome to this course!"

welcome3 <- paste(welcome, collapse = "")
welcome3
## [1] "Welcometothiscourse!"
```

Missing values

A lot of descriptive functions will throw up an error when there are missing values.

```
num_vec <- c(5, NA, 15, 20, 25, NA)
sum(num_vec)
## [1] NA
mean(num_vec)
## [1] NA
```

Descriptive functions include the argument `na.rm = TRUE`, which explicitly tells `R` to ignore missing values.

```
sum(num_vec, na.rm = TRUE)
## [1] 65
mean(num_vec, na.rm = TRUE)
## [1] 16.25
```

`is.na` is a logical operation that allows us to identify missing values.

```
is.na(num_vec)
## [1] FALSE  TRUE FALSE FALSE FALSE  TRUE
```

A note about vectors

Vectors can include either character values or numeric values, not both!

```
x <- rep(c(5, "a"), times = 2)
x
## [1] "5" "a" "5" "a"

x / 2
## Error in x/2: non-numeric argument to binary operator
```

With `x_num`, `R` automatically treats the vector as a character vector, because it includes some characters.

If we force `R` to treat `x` as a numeric vector, it will replace all non-numeric elements with `NA`.

```
as.numeric(x)
## Warning: NAs introduced by coercion
## [1]  5 NA  5 NA
```

Complete Exercise 2 on the worksheet

Indexing vectors with `[]`

Often we don't want to retrieve the whole vector, but only a specific element.

We can do this using `[]`.

Numeric indexing

`a[index]`, where `a` is the vector, and `index` is a vector of index values.

```
colors <- colors()

# What is the first color?
colors[1]
## [1] "white"

# What are the first 5 colors
colors[1:3]
## [1] "white"         "aliceblue"    "antiquewhite"
```

Logical indexing

When indexing a vector with a logical index, `R` will only return values for which the index is `TRUE`.

```r
years <- c(2010, 2005, 2012, 2013, 2001)

# select all years above 2010
years[years>2010]
## [1] 2012 2013

# select all years larger than 2002 and smaller than 2013
years[years > 2002 & years < 2013]
## [1] 2010 2005 2012
```

`R` actually interprets TRUE values as 1 and FALSE values as 0.

This allows us to quickly answer questions like:

```r
#How many observations in years are greater than 2005?
sum(years > 2005)
## [1] 3

# What's the proportion of observations in years greater than 2005
```

Quiz

```r
# Generates a standard normal distribution
x_norm <- rnorm(1000, mean = 0, sd = 1)
```

1. Get the 10th and 20th observation.
2. Save all observations below 0 in a new variable called `x_norm_neg`.
3. How many observations are below 0?
4. Get the proportion of values below -2 and above 2.

Quiz

1. Get the 10th and 20th observation

```
x_norm[c(10, 20)]
## [1] -0.3771047 -0.1016105
```

1. Save all observations below 0 in a new variable called `x_norm_neg`.

```
x_norm_neg <- x_norm[x_norm<0]
```

1. How many observations are below 0?

```
length(x_norm_neg)
## [1] 508
```

1. Get the proportion of values below -2 and above 2.

```
mean(x_norm > 2 | x_norm < -2)
## [1] 0.042
```

Changing values of a vector

In the example below, you know that the 4th value should have been 23, but was wrongly coded as NA.

```
age <- c(17, 21, 22, 25, NA)
age[5] <- 23
age
## [1] 17 21 22 25 23

age[age >= 18] <- "18+"
age
## [1] "17"  "18+" "18+" "18+" "18+"
```

Quiz

```r
x <- c(5, 15, NA, 25, 30)
```

1. What elements are we subsetting?

```r
x[1]
x[c(3, 4)]
x[!is.na(x)]
```

1. What is a quick way to calculate the share of missing values in x?

2. Replace the missing values in `x` using the `is.na()` operator.

Dataframes

Most of the work we do as sociologists will involve playing around with rectangular data or dataframes.

While vectors are one dimensional, dataframes have two dimensions; rows and columns.

- **columns**: variables
- **rows**: observations

name	height	mass	hair_color	skin_color
Luke Skywalker	172	77	blond	fair
C-3PO	167	75	NA	gold
R2-D2	96	32	NA	white, blue
Darth Vader	202	136	none	white
Leia Organa	150	49	brown	light
Owen Lars	178	120	brown, grey	light

Dataframes

Creating a dataframe

You can turn multiple vectors into a dataframe using the `data.frame` command.

```
# vectors
country_name <- c("Nigeria", "Gambia", "Finland", "Brazil")
country_year <- 2013
country_pop_1m <- c(173.6, 1.8, 5.4, 200.4)

# combine into dataframe
pop_df <- data.frame("country" = country_name,
                     "year" = country_year,
                     "pop_1m" = country_pop_1m,
                     stringsAsFactors = FALSE)

pop_df
##   country year pop_1m
## 1 Nigeria 2013  173.6
## 2  Gambia 2013    1.8
## 3 Finland 2013    5.4
## 4  Brazil 2013  200.4
```

Load data

Rather than constructing your own dataframe in `R`, you will often work with data that already comes in a specific format, such as a csv, excel, .txt, Stata (.dta), or other file type.

`R` can load all these different file types.

```r
# csv file
data <- read.csv(data)

# To load stata files, you have to use the foreing package
# Note that you might need to install the package
data <- foreign::read.dta(data)

# load an R data file
load("data.Rda")
```

Complete Exercise 3 on the worksheet

Inspect the data

Here are some of the most important ways to inspect a dataframe.

Function	**Description**
`head(x), tail(x)`	Print the first few rows (or last few rows).
`View(x)`	Open the entire object in a new window
`nrow(x), ncol(x), dim(x)`	Count the number of rows and columns
`names()`	Show the row (or column) names
`summary(x)`	Show the summary statistics of a dataframe

Inspect the data

```r
# get the dimensions (rows and columns) of your data
dim(pop_df)
## [1] 4 3

# view the first 5 rows of data
head(pop_df)
##   country year pop_1m
## 1 Nigeria 2013  173.6
## 2  Gambia 2013    1.8
## 3 Finland 2013    5.4
## 4  Brazil 2013  200.4

# view the last 2 rows of data
tail(pop_df, 2)
##   country year pop_1m
## 3 Finland 2013    5.4
## 4  Brazil 2013  200.4
```

Inspect the data

```
# inspect the variable names in your dataframe
names(pop_df)
## [1] "country" "year"    "pop_1m"

# generate summary statistics for each variable in your dataframe.
summary(pop_df)
##    country               year          pop_1m    
##  Length:4           Min.   :2013   Min.   :  1.8  
##  Class :character   1st Qu.:2013   1st Qu.:  4.5  
##  Mode  :character   Median :2013   Median : 89.5  
##                     Mean   :2013   Mean   : 95.3  
##                     3rd Qu.:2013   3rd Qu.:180.3  
##                     Max.   :2013   Max.   :200.4  
```

Complete Exercise 3 on the worksheet