

# R for Infectious Disease Modelling

Yang Liu, PhD  
[yang.liu@lshtm.ac.uk](mailto:yang.liu@lshtm.ac.uk)

Shanghai, China  
August, 2019

LONDON  
SCHOOL of  
HYGIENE  
& TROPICAL  
MEDICINE



- Have a brief overview of the fundamentals of R
- R in your work flow
  - Import and export
  - Tidy and transform
  - Visualization
- Finding Solutions/ Self-learning

“Lecture\_CodeAlong.R” is useful during this lecture.



# Brief Review of R

LONDON  
SCHOOL of  
HYGIENE  
& TROPICAL  
MEDICINE



# What is R all about?

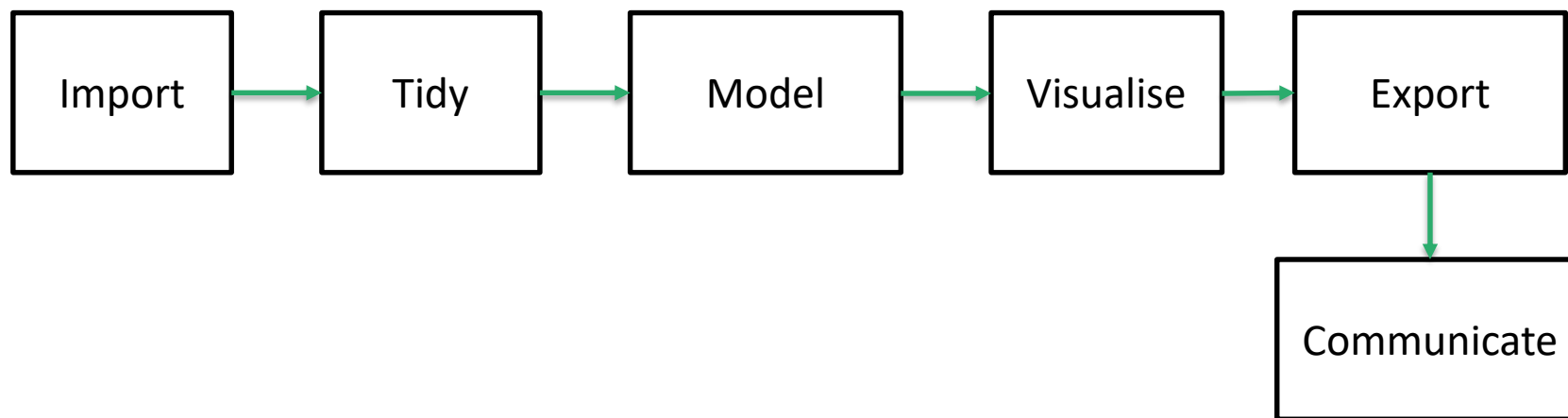


# Why though?

1. R can do a lot of things:
  - Read and process different data types and file types
  - Data cleaning
  - Web-scraping; interacting with websites
  - Statistical analysis
  - Mathematical modelling
  - Data visualization
  - Document processing, e.g. Markdown
  - Dashboards; reports generation
  - Mapping and spatial modelling

# Why though?

## 1. R can do a lot of things:



# Why though?

2. R plays well with other software/ programming languages/ systems/ platforms:
  - Python
  - C++
  - Jags (Just another Gibbs sampler)
  - OpenBUGS
  - git

# Why though?

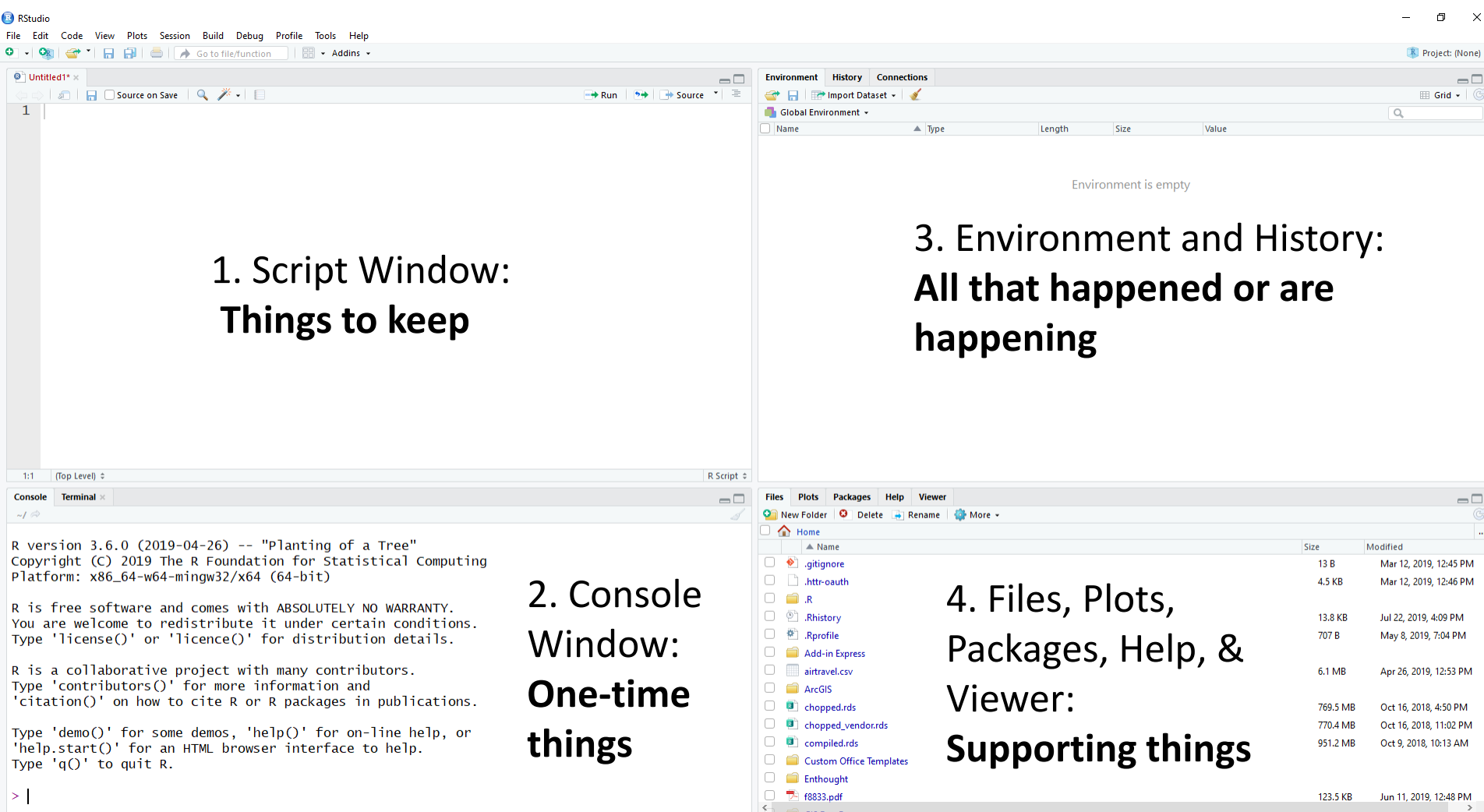
3. R is a community driven project:
  - Open source
  - Constantly evolving
  - Channels to interact



## 4. Last but not the least:



# What does it look like?



**1. Script Window: Things to keep**

```
1 |
```

**2. Console Window: One-time things**

```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

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.

> |
```

**3. Environment and History: All that happened or are happening**

Environment is empty

**4. Files, Plots, Packages, Help, & Viewer: Supporting things**

Name	Size	Modified
.gitignore	13 B	Mar 12, 2019, 12:45 PM
.gitignore	4.5 KB	Mar 12, 2019, 12:46 PM
.R		
.Rhistory	13.8 KB	Jul 22, 2019, 4:09 PM
.Rprofile	707 B	May 8, 2019, 7:04 PM
Add-in Express		
airtravel.csv	6.1 MB	Apr 26, 2019, 12:53 PM
ArcGIS		
chopped.rds	769.5 MB	Oct 16, 2018, 4:50 PM
chopped_vendor.rds	770.4 MB	Oct 16, 2018, 11:02 PM
compiled.rds	951.2 MB	Oct 9, 2018, 10:13 AM
Custom Office Templates		
Enthought		
f8833.pdf	123.5 KB	Jun 11, 2019, 12:48 PM

# Let's run a line now!



To run codes, you can:

- (1) Type it/ copy & paste it in the console window and press enter;**
- (2) Type it in the script window, either a. put the cursor on the target line or b. highlight the target code with your mouse, and then press ctrl + enter;**
- (3) Type it in the script window, select it using your mouse, and then click the “Run” button top right;**
- (4) Click the “Source” button on top right, which will run the whole script.**

# Starting from objects

## Level 1 (Highest)

- List

```
lv1_1 <- list(lv3_1,  
             lv3_2,  
             lv3_3,  
             lv3_4,  
             lv3_5)#list
```

## Level 2

- Vector, matrix, data frame...

```
lv2_1 <- c(1:10) #vector, 1D  
lv2_2 <- matrix(lv2_1, ncol = 2) #matrix, 2D  
lv2_3 <- data.frame(lv2_2) #dataframe, 2D
```

## Level 3 (Lowest)

- Double/ integer (numeric), text (character), Logical, Factor (categorical), Date...

```
lv3_1 <- 2 #double/ integer  
lv3_2 <- "Shanghai" #character/ text  
lv3_3 <- TRUE #logical  
lv3_4 <- factor(c("animal")) #factor  
lv3_5 <- as.Date("2019-07-30") #date
```

```
print(lv3_1)  
class(lv1_1)  
lv2_1[1]  
lv2_2[1,2]
```

# Overall Indexing

Level 1



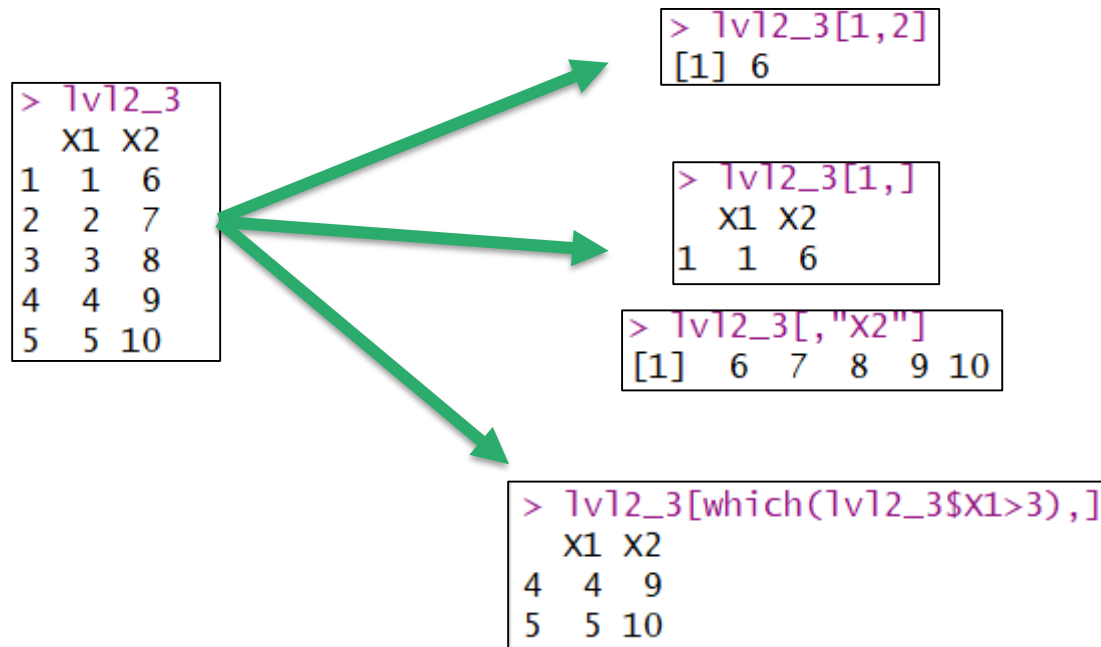
Level 2



Level 3



Subsetting can be done in many different ways depending on your needs. To name a few:



There is an additional type of object we did not talk about – functions. Instead of storing data in various formats, functions store “rules”:

```
> circle_area <- function(r){  
+   area_tmp <- pi * r * r  
+   return(area_tmp)  
+ }  
> circle_area(2)  
[1] 12.56637
```

Generally there are three sources of functions: (1) base R; (2) specific packages; and (3) write your own!

# R relies on packages, what about that?

Base R contains a large number of functions already that can meet simple needs. But for more advanced techniques, you sometimes need to rely on packages.

```
install.packages("EpiDynamics")  
library("EpiDynamics")  
require("EpiDynamics")  
EpiDynamics::SIR
```



Line 1: Install a package from Internet

Line 2 and 3: Both are used to activate a package. “Require” is more commonly used within a function. “Library” is the right way to go in most cases.

Line 4: Here, you are calling the function SIR from the package EpiDynamics. This is very useful when you have the same function names in different packages.



Advanced options:

- (1) pacman
- (2) devtools::install\_github



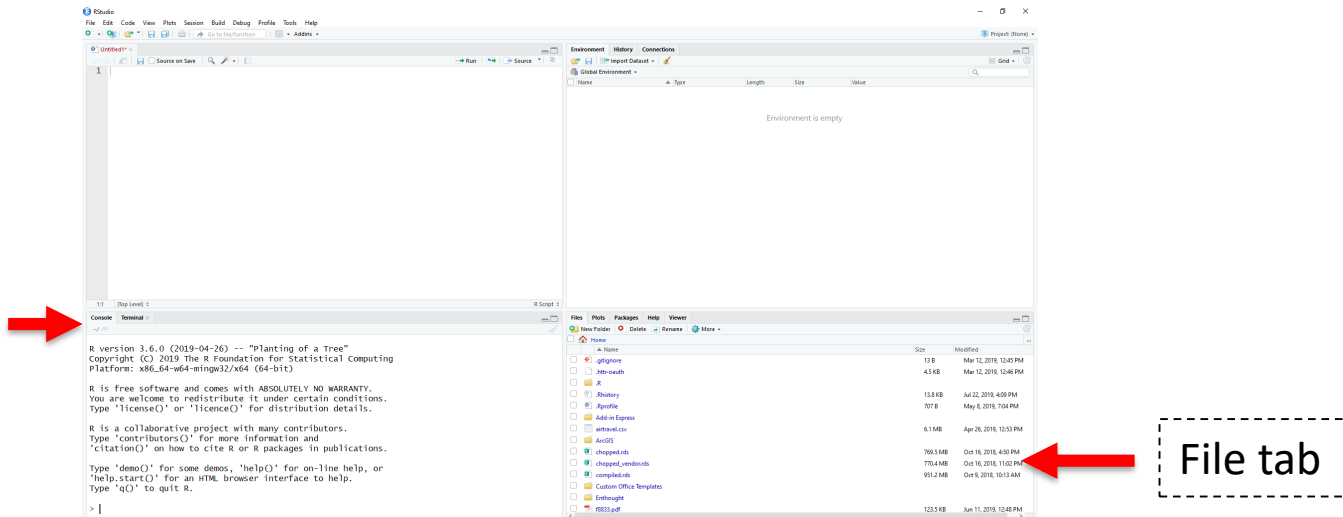
# Interacting with R



R is a piece of software. It only looks at where you tell it to look. These commands will help you get an idea of where R is looking, or tell R where to look at.

```
getwd()  
list.files()  
setwd("~/SOMWHERE ELSE")
```

Alternatively:



# RECAP: What else just happened?

Symbol	Implication



# Questions?

LONDON  
SCHOOL of  
HYGIENE  
& TROPICAL  
MEDICINE

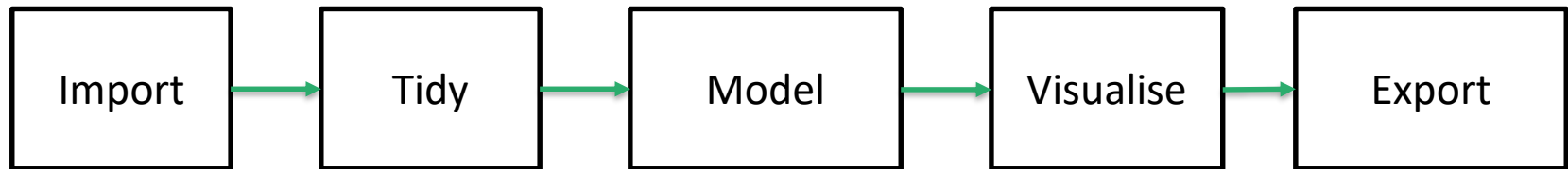


# R in your workflow

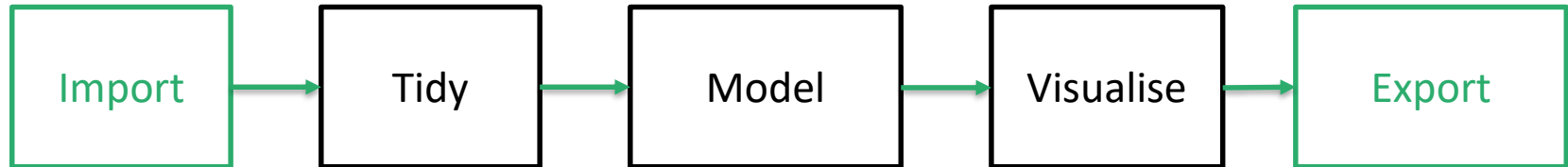
LONDON  
SCHOOL of  
HYGIENE  
& TROPICAL  
MEDICINE

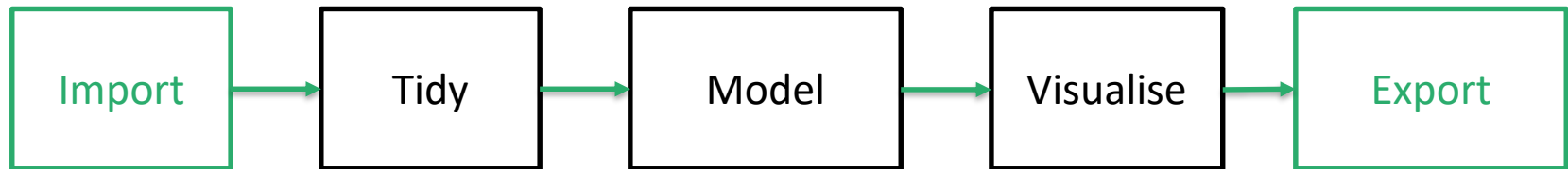


# Your flow of work



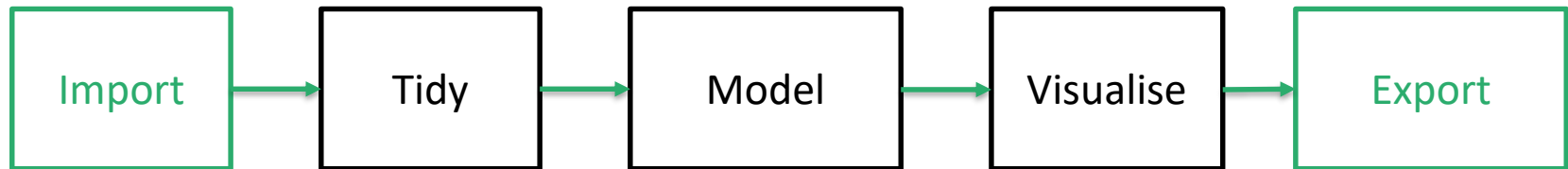
# Import & Export





R is capable of working with many different file types. In my personal experience, I have worked with:

- Regular Documents: \*.txt, \*.csv, \*.xlsx, \*.pdf;
- Geographic Information: \*.shp, \*.tif;
- R code: \*.r, \*.rmd;
- R objects: \*.rds, \*.rdata;
- Images: \*.png, \*.jpeg;
- Large data files: \*.ncdf4;
- Web content: \*.html, \*.xml.




R is capable of working with many different file types. In my personal experience, I have worked with:

- Regular Documents: \*.txt, \*.xlsx ;
- Geographic Information: \*.shp
- R code: \*.r
- R objects: \*.rds, \*.rdata;
- Images:
- Large data files:
- Web content:



These three are file types unique to R.

- .r files contain **r scripts**. These are the files where you codes will be saved. → 
- .rdata can save **multiple r objects**. Before you close R, sometimes the software will ask you if you'd like to the current workspace to an .rdata file. Proceed with caution. → `save()`
- .rds can only save **a single r object**. The advantage is that it can maintain the object class. It's also good at compressing file size. → `write_rds(); saveRDS(); read_rds(); readRDS()`

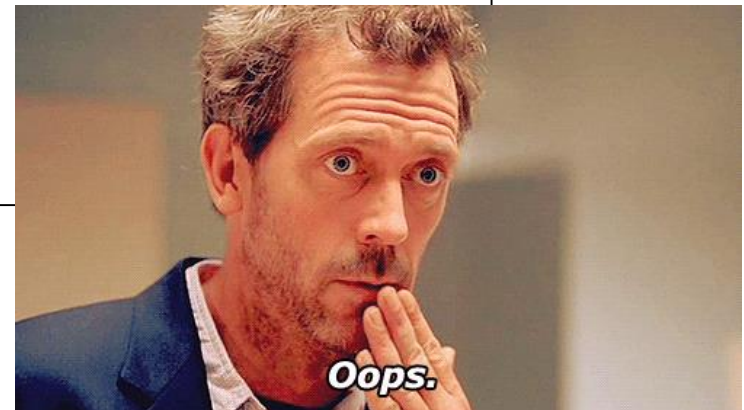
These two are probably the most common types of file type import. You might have worked with them via **Notepad** or **Excel**.

```
> lvl2_3
  x1 x2
1  1  6
2  2  7
3  3  8
4  4  9
5  5 10
> write.table(lvl2_3, file = "lvl2_3.txt")
> read.table("lvl2_3.txt")
  x1 x2
1  1  6
2  2  7
3  3  8
4  4  9
5  5 10
```

These two are probably the model common types of file type import. You might have worked with them via **Notepad** or **Excel**.

```
> #install.packages("xlsx")
> #install.packages("readxl")
> lv12_3["X3"] <- c("苹果", "葡萄", "香蕉", "哈密瓜", "菠萝")
> lv12_3
```

	X1	X2	X3
1	1	6	<U+82F9><U+679C>
2	2	7	<U+8461><U+8404>
3	3	8	<U+9999><U+8549>
4	4	9	<U+54C8><U+5BC6><U+74DC>
5	5	10	<U+83E0><U+841D>



# .txt and .xlsx

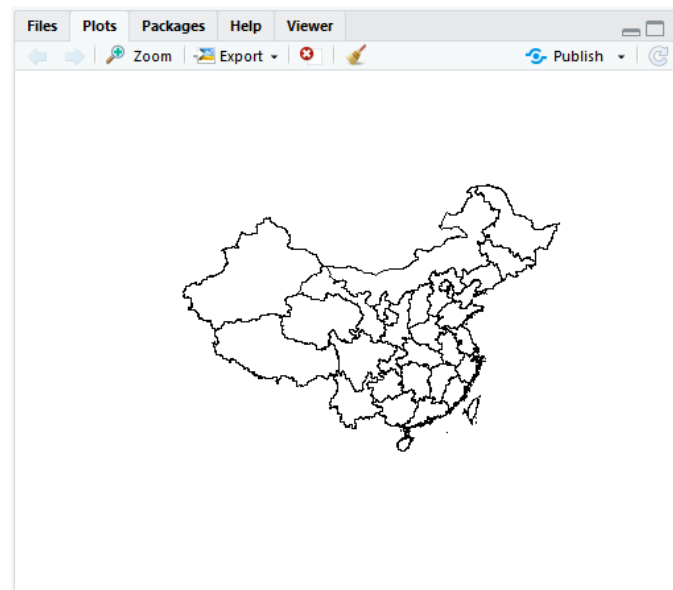
```
> Sys.setlocale("LC_ALL", "Chinese")
[1] "LC_COLLATE=Chinese (Simplified)_China.936;LC_CTYPE=Chinese (Simplified)_China.
IC=C;LC_TIME=Chinese (Simplified)_China.936"
> lv12_3
  X1 X2   X3
1  1  6  苹果
2  2  7  葡萄
3  3  8  香蕉
4  4  9  哈密瓜
5  5 10  菠萝
> xlsx::write.xlsx(lv12_3, file = "lv12_3.xlsx")
> readxl::read_excel("lv12_3.xlsx")
New names:
* `` -> ...1
# A tibble: 5 x 4
  ...1     X1     X2 X3
<chr> <dbl> <dbl> <chr>
1  1         1     6 苹果
2  2         2     7 葡萄
3  3         3     8 香蕉
4  4         4     9 哈密瓜
5  5         5    10 菠萝
```

Shape files contains geographic information about locations in the format of polygon (area), point, or raster (grid-based).

```
install.packages("rgdal")  
library(rgdal)  
chn <- readOGR("C:/Users/eideyliu/Downloads/2014Shapefile_ChinaCDC_20140325/shp", "sheng")  
plot(chn)
```

Advanced options:

- (1) sf
- (2) Mapview
- (3) Leaflet



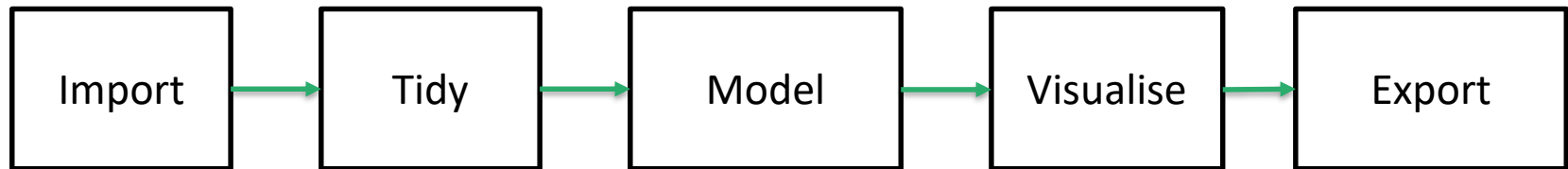
## Regular Documents

- `readxl`
- `writexl`
- `xlsx`

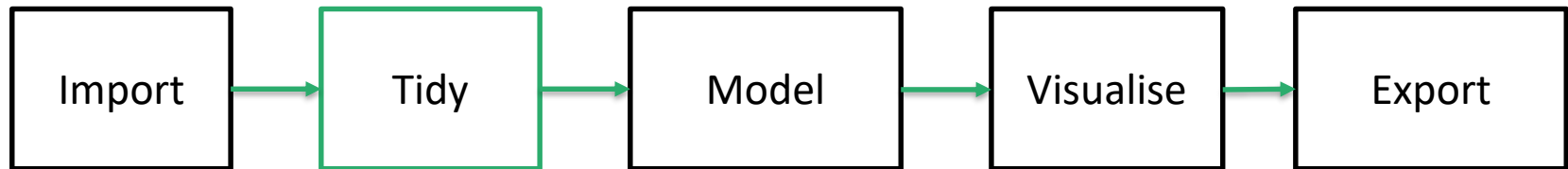
## Geographic Information

- `rgdal`
- `sf`

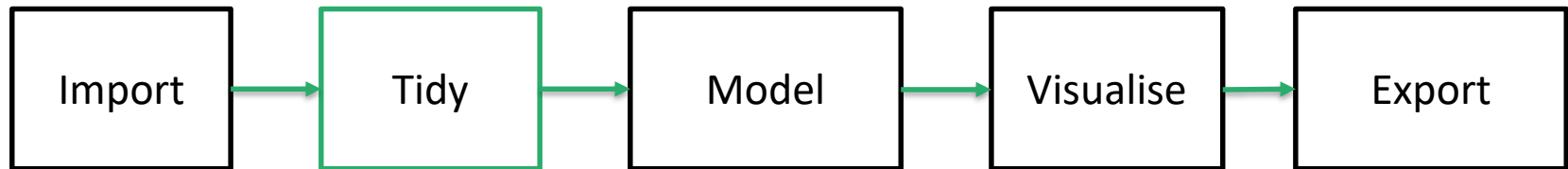
# Your flow of work



# Your flow of work

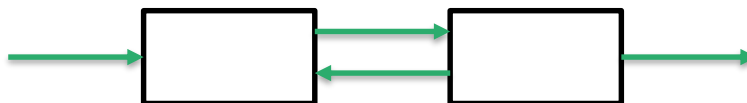






Tidying your data entails:

- **Convert/ generate** new data from raw data so that they become meaningful to your analysis.
- **Organize** your data in a way that is well structured, consistent, and easy to work with for the Modelling and Visualization steps;



# Converting between classes

## Level 1 (Highest)

- List

## Level 2

- Vector, matrix, data frame, sf...

## Level 3 (Lowest)

- Double/ integer (numeric), text (character), Logical, Factor (categorical), Date...

The general rule is if you are trying to **convert into objects that allows for the same or higher dimensions**, you can convert between classes using functions with the `as.CLASSNAME()`, e.g., `as.numeric()`; `as.matrix()`.

# Converting between classes

## Level 1 (Highest)

- List

## Level 2

- Vector, matrix, data frame, sf...

## Level 3 (Lowest)

- Double/ integer (numeric), text (character), Logical, Factor (categorical), Date...

However, it will need to depend on specific cases:

- Integer  $\rightarrow$  text?
- Text  $\rightarrow$  Integer?
- Integer  $\rightarrow$  Logical?
- Logical  $\rightarrow$  Integer?
- Vector  $\rightarrow$  Matrix?
- Matrix  $\rightarrow$  Vector?

It's important to keep an on if conversion processes are doing what you want them to do.

IF statements are a one-time thing:

- a **IF** statement means something will happen for all when certain conditions are met.

```
> 1v12_3$X3
[1] "苹果"    "葡萄"    "猴子"    "哈密瓜" "马"
> is_fruit <- function(type){
+   if(type %in% c("苹果","葡萄","哈密瓜")){
+     return(T)
+   } else {return(F)}
+ }
> is_fruit("苹果")
[1] TRUE
> is_fruit("马")
[1] FALSE
>
```

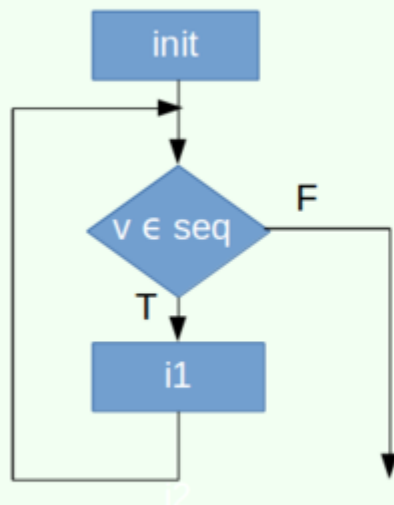
There are three basic types of loops in R:

- a **FOR** loop is used when you want SOMETHING to happen **for a range** of conditions;
- a **REPEAT** loop is used when you want SOMETHING to happen **for as long as** certain conditions are met;
- a **WHILE** loop is used when you want SOMETHING to happen **for as long as** certain conditions are **not** met.

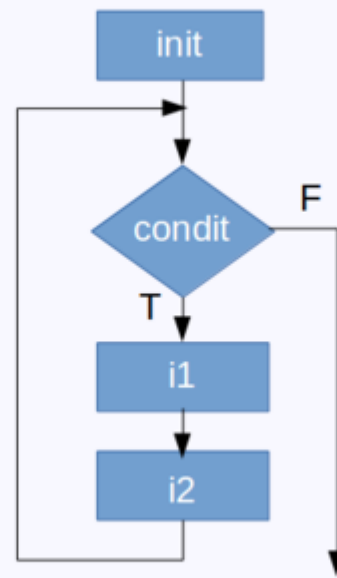
# Loops



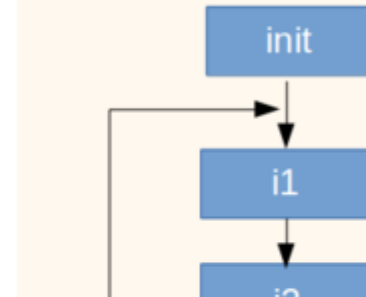
## For loop



## while loop



## repeat loop



<https://www.datacamp.com/community/tutorials/tutorial-on-loops-in-r>  
Carlo Fanara, 2018



```
> library(tidyverse)
-- Attaching packages ----- tidyverse 1.2.1 --
v ggplot2 3.2.0      v purrr   0.3.2
v tibble  2.1.2      v dplyr   0.8.1
v tidyr   0.8.3      v stringr 1.4.0
v readr   1.3.1      v forcats 0.4.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

Package	Usage
ggplot2	Visualization
tibble	Data storage
tidyr	Data cleaning
readr	Import data
purrr	Application of functions to lists of objects; advance conversion
dplyr	Data wrangling
stringr	Work with texts
forcats	For categorical variables





# Useful Commands in Tidyverse

- `select`
- `mutate`
- `filter`
- `gather`
- `spread`
- `join`

# Let's try it out with a toy model!

Disease ***Birdy Pox*** is a (fictional) infectious disease with a transmission rate (Beta) of 2 and a recovery rate (Gamma) of 0.15. This outbreak within a population of 1,000,000 is a result of 1 infectious individual (initial condition). We observe the outbreak progression for 70 days.

```
parameters <- c(beta = 2, gamma = 0.15)
initials <- c(S = 1 - 1e-06, I = 1e-06, R = 0)
res <- SIR(pars = parameters,
           init = initials,
           time = 0:70)$results
head(res)
```



# Let's try it out with a toy model!

Disease ***Birdy Pox*** is a (fictional) infectious disease with a transmission rate (Beta) of 2 and a recovery rate (Gamma) of 0.15. This outbreak within a population of 1,000,000 is a result of 1 infectious individual (initial condition). We observe the outbreak progression for 70 days.



# Let's try it out with a toy model!

```
> res
  time      S      I      R
1    0 9.999990e-01 1.000000e-06 2.000000e-06
2    1 9.999912e-01 8.174281e-06 2.581702e-06
3    2 9.999426e-01 5.313143e-05 6.227017e-06
4    3 9.996285e-01 3.437448e-04 2.979590e-05
5    4 9.976319e-01 2.190313e-03 1.797414e-04
6    5 9.851226e-01 1.375322e-02 1.126135e-03
7    6 9.126303e-01 8.051224e-02 6.859456e-03
8    7 6.259288e-01 3.389319e-01 3.514135e-02
9    8 2.220655e-01 6.650748e-01 1.128617e-01
10   9 5.315178e-02 7.267538e-01 2.200964e-01
11  10 1.314886e-02 6.620028e-01 3.248503e-01
12  11 3.803566e-03 5.783245e-01 4.178740e-01
13  12 1.295759e-03 5.000625e-01 4.986437e-01
14  13 5.115105e-04 4.311264e-01 5.683641e-01
15  14 2.296210e-04 3.713327e-01 6.284397e-01
```

The function `select` helps you trim your working table based on needs: keeping the relevant variables, getting rid of the ones that are not needed. **What if we are only interested in I?**

```
> select(res, time, I)
  time      I
1    0 1.000000e-06
2    1 8.174281e-06
3    2 5.313143e-05
4    3 3.437448e-04
5    4 2.190313e-03
6    5 1.375322e-02
7    6 8.051224e-02
8    7 3.389319e-01
9    8 6.650748e-01
10   9 7.267538e-01
```

The function `mutate` helps you convert/ compute based on variables that already exist in the table provided. **What if we want to convert the unit from time from days to weeks?**

```
> mutate(res, time_wk = time/7)
```

	time	S	I	R	time_wk
1	0	9.999990e-01	1.000000e-06	2.000000e-06	0.0000000
2	1	9.999912e-01	8.174281e-06	2.581702e-06	0.1428571
3	2	9.999426e-01	5.313143e-05	6.227017e-06	0.2857143
4	3	9.996285e-01	3.437448e-04	2.979590e-05	0.4285714
5	4	9.976319e-01	2.190313e-03	1.797414e-04	0.5714286
6	5	9.851226e-01	1.375322e-02	1.126135e-03	0.7142857
7	6	9.126303e-01	8.051224e-02	6.859456e-03	0.8571429
8	7	6.259288e-01	3.389319e-01	3.514135e-02	1.0000000
9	8	2.220655e-01	6.650748e-01	1.128617e-01	1.1428571
10	9	5.315178e-02	7.267538e-01	2.200964e-01	1.2857143

The function `filter` is a form of subsetting. It is capable of processing multiple filter based on multiple conditions.

**What if we are only interested in looking at the time when proportion of I is greater than 0.2?**

```
> dplyr::filter(res, I>0.15)
```

	time	S	I	R
1	7	6.259288e-01	0.3389319	0.03514135
2	8	2.220655e-01	0.6650748	0.11286167
3	9	5.315178e-02	0.7267538	0.22009639
4	10	1.314886e-02	0.6620028	0.32485030
5	11	3.803566e-03	0.5783245	0.41787396
6	12	1.295759e-03	0.5000625	0.49864373
7	13	5.115105e-04	0.4311264	0.56836407
8	14	2.296210e-04	0.3713327	0.62843973
9	15	1.151378e-04	0.3197142	0.68017268
10	16	6.356784e-05	0.2752280	0.72471046
11	17	3.808738e-05	0.2369143	0.76304961
12	18	2.453377e-05	0.2039264	0.79605103
13	19	1.679474e-05	0.1755282	0.82445701
14	20	1.210954e-05	0.1510826	0.84890725

The function `filter` is a form of subsetting. It is capable of processing multiple filter based on multiple conditions.

**What if we are only interested in looking at this same table but only at the end of each week?**

```
> dplyr::filter(res, time%%7 == 0)
```

	time	S	I	R
1	0	9.999990e-01	1.000000e-06	0.00000200
2	7	6.259288e-01	3.389319e-01	0.03514135
3	14	2.296210e-04	3.713327e-01	0.62843973
4	21	9.143379e-06	1.300406e-01	0.86995227
5	28	2.962983e-06	4.550924e-02	0.95448980
6	35	1.997260e-06	1.592595e-02	0.98407405
7	42	1.739746e-06	5.573243e-03	0.99442702
8	49	1.657703e-06	1.950337e-03	0.99805001
9	56	1.629915e-06	6.825114e-04	0.99931786
10	63	1.620302e-06	2.388387e-04	0.99976154
11	70	1.616951e-06	8.358084e-05	0.99991680



The function **gather** changes tables from wide to long form.

- A wide table looks like the following:

Time	S	I	R
0	a	c	e
1	b	d	f

The function **gather** changes tables from wide to long form. Using the same example, a long table would look like something like this:


Time	key	value
0	S	a
1	S	b
0	I	c
1	I	d
0	R	e
1	R	f

# gather and spread

The function **gather** changes tables from wide to long form. To implement:

```
> tmp <- gather(res, key = key, value = value, -time)
> tmp
```

	time	key	value
1	0	S	9.999990e-01
2	1	S	9.999912e-01
3	2	S	9.999426e-01
4	3	S	9.996285e-01
5	4	S	9.976319e-01
6	5	S	9.851226e-01
7	6	S	9.126303e-01
8	7	S	6.259288e-01
9	8	S	2.220655e-01
10	9	S	5.315178e-02
11	10	S	1.314886e-02



The function `spread` changes tables from long to wide form. To implement:

```
> spread(tmp, key = key, value = value)
```

	time	I	R	S
1	0	1.000000e-05	2.000000e-05	9.999990e-01
2	1	8.174281e-06	2.581702e-06	9.999912e-01
3	2	5.313143e-05	6.227017e-06	9.999426e-01
4	3	3.437448e-04	2.979590e-05	9.996285e-01
5	4	2.190313e-03	1.797414e-04	9.976319e-01
6	5	1.375322e-02	1.126135e-03	9.851226e-01
7	6	8.051224e-02	6.859456e-03	9.126303e-01
8	7	3.389319e-01	3.514135e-02	6.259288e-01
9	8	6.650748e-01	1.128617e-01	2.220655e-01
10	9	7.267538e-01	2.200964e-01	5.315178e-02

A fix:

```
> class(tmp$key)
[1] "character"
> tmp$key <- factor(tmp$key, levels = c("S","I","R"))
> tmp <- tmp[order(tmp$key),]
> class(tmp$key)
[1] "factor"
> spread(tmp, key = key, value = value)
```


	time	S	I	R
1	0	9.999990e-01	1.000000e-06	2.000000e-06
2	1	9.999912e-01	8.174281e-06	2.581702e-06
3	2	9.999426e-01	5.313143e-05	6.227017e-06
4	3	9.996285e-01	3.437448e-04	2.979590e-05
5	4	9.976319e-01	2.190313e-03	1.797414e-04
6	5	9.851226e-01	1.375322e-02	1.126135e-03
7	6	9.126303e-01	8.051224e-02	6.859456e-03
8	7	6.259288e-01	3.389319e-01	3.514135e-02
9	8	2.220655e-01	6.650748e-01	1.128617e-01
10	9	5.315178e-02	7.267538e-01	2.200964e-01

The function `join` merges two tables based on certain column.

Y	X1
0	a
1	b

Y	X2
0	c
2	d

“full\_join”



Y	X1	x2
0	a	c
1	c	NA
2	NA	d

The function `join` merges two tables based on certain column.

```
> holiday <- data.frame(time = c(4,23),
+                         holiday = 1)
> tmp <- full_join(res, holiday, by = "time")
> tmp[is.na(tmp$holiday), "holiday"] <- 0
> head(tmp)
```

	time	S	I	R	holiday
1	0	0.99999990	1.0000000e-06	0.0000000e+00	0
2	1	0.99999912	8.174281e-06	5.817023e-07	0
3	2	0.9999426	5.313143e-05	4.227017e-06	0
4	3	0.9996285	3.437448e-04	2.779590e-05	0
5	4	0.9976319	2.190313e-03	1.777414e-04	1
6	5	0.9851226	1.375322e-02	1.124135e-03	0

# Advance option: %>%

These are read as “**pipes**”. You can think of these as “and then”. It chains functions together. While doing data cleaning, you may want to do multiple things to the same object – this is when pipes become particularly useful.

If  $f()$  and  $g()$  are both functions:

- $g(f(x))$
- $x \%>\% f \%>\% g$

These two expressions mean the exact same things.



# Advance option: %>%

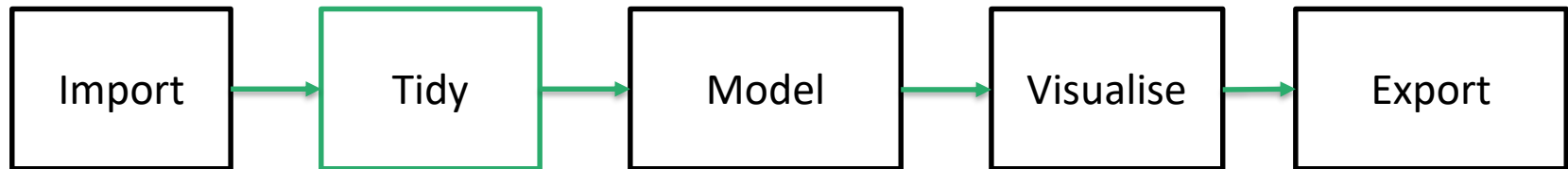
We want to first filter the original table to only look at the estimates at every week-end, and then we want to convert the table to a long table:

```
tmp <- dplyr::filter(res, time%%7 == 0)
tmp <- gather(tmp, key = state, value = proportion, -time)
```

VS

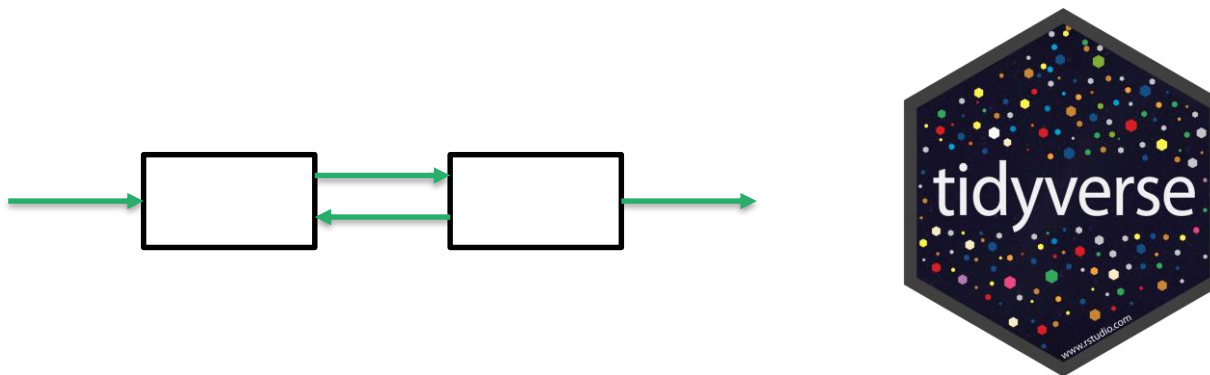
```
res %>%
  filter(., time%%7 == 0) %>%
  gather(., key = state, value = proportion, -time) -> tmp
```

# RECAP: Import & Export

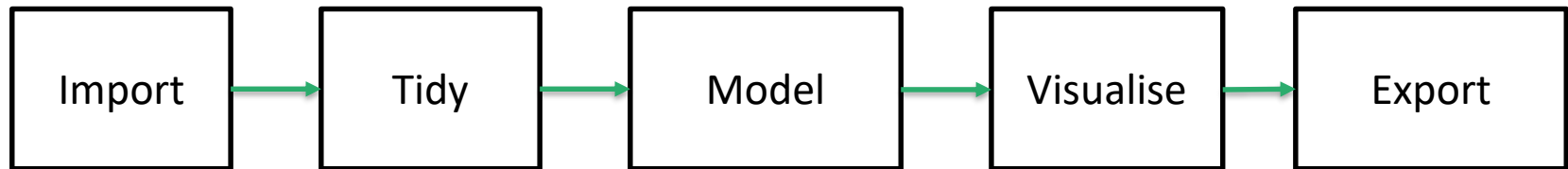


Tidying your data entails:

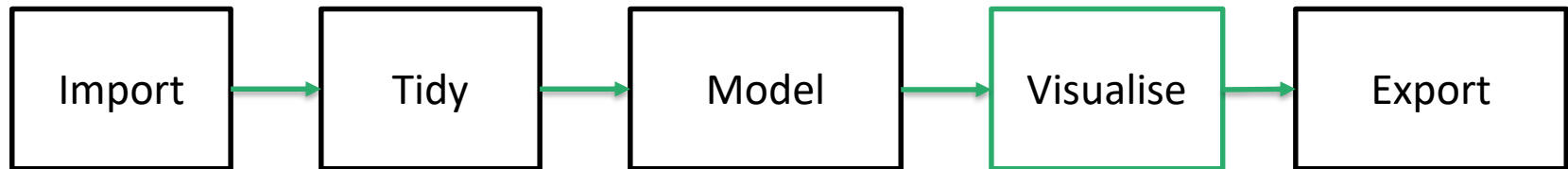
- **Convert/ generate** new data from raw data so that they become meaningful to your analysis.
- **Organize** your data in a way that is well structured, consistent, and easy to work with for the Modelling and Visualization steps;



# Your flow of work

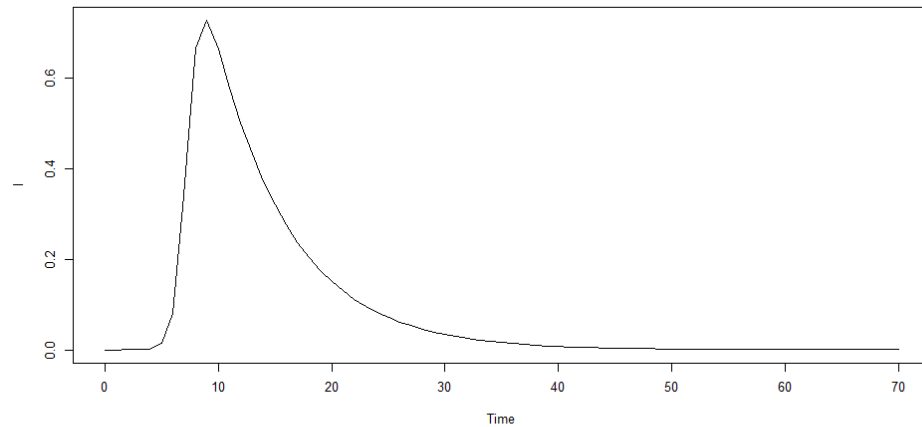


# Your flow of work

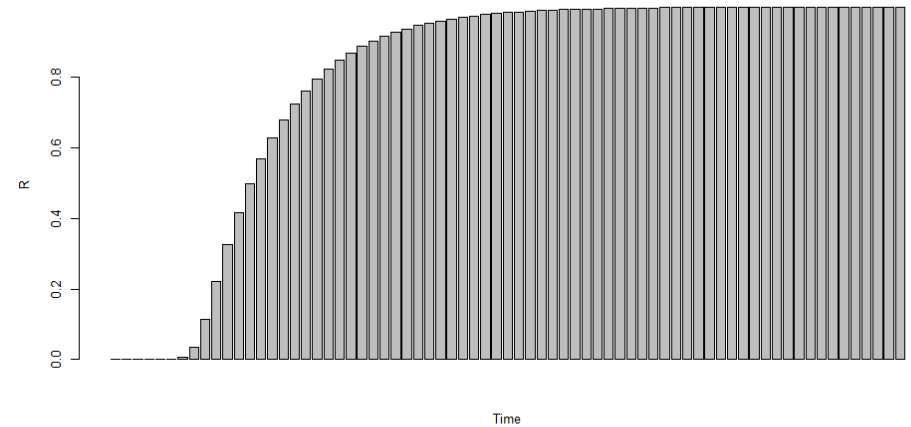


# Base R plots

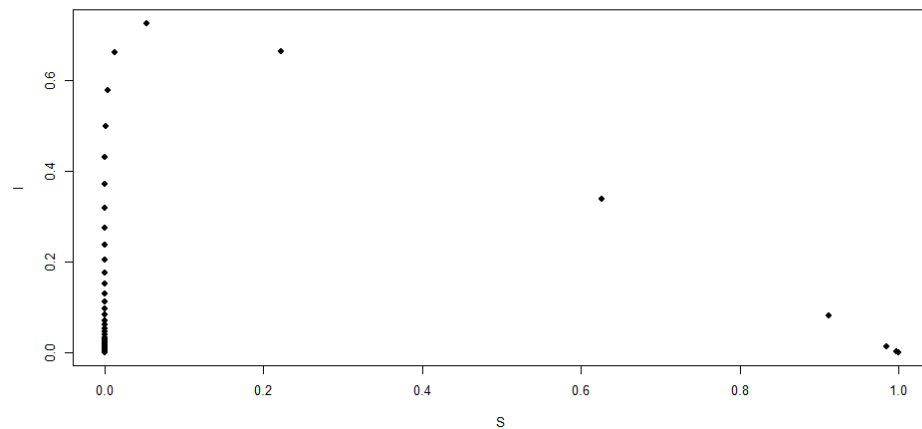
Time-series of I



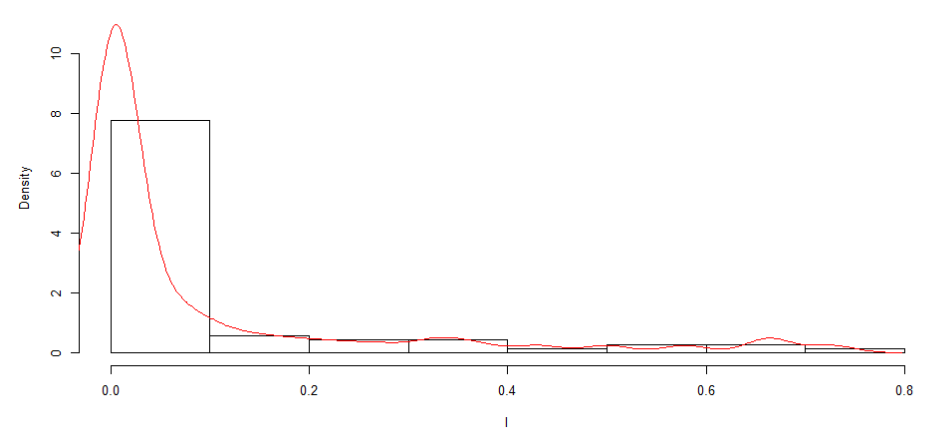
Time-series of R



Relationship between S and I



Histogram of I



```
par(mfrow=c(2,2))
#
#line plot
plot(x = res$time, y = res$I, type = "l", xlab = "Time", ylab = "I", main = "Time-series of I")
#bar plot
barplot(res$R, xlab = "Time", ylab = "R", main = "Time-series of R")
#scatter plot
plot(x = res$S, y = res$I, xlab = "S", ylab = "I", main = "Relationship between S and I", pch = 19)
#histogram + density line
hist(res$I, breaks = 10, freq = F, ylim=c(0, 11), xlab = "I", main = "Histogram of I")
lines(density(res$I), col = "red")
```

**Method 1:** Use your mouse. In the bottom right panel, in the "plot" label, the drop down menu "Export" will lead the users to an interface that does the job.

**Method 2:** Use code.

```
png("sample.png")
plot(x = res$time,
      y = res$I,
      type = "l",
      xlab = "Time",
      ylab = "I",
      main = "Time-series of I")
dev.off()
```

Part 1: Create an empty "container"

Part 2: Plot


Part 3: Tell R you are done

The function `ggplot()` uses grammar of graphics. Essentially, it uses code to make pictures. Let's go back to ***Birdy Pox*** and see what we can do.





ggplot works with long-form dataset.



```
tmp <- gather(res,  
              key = state,  
              value = proportion, -time)  
  
tmp <- mutate(tmp,  
              state = factor(state,  
                             levels = c("S", "I", "R"),  
                             labels = c("Susceptible", "Infectious", "Recovered"))))
```



Again, we need to make sure object classes are correct.

# ggplot: structure

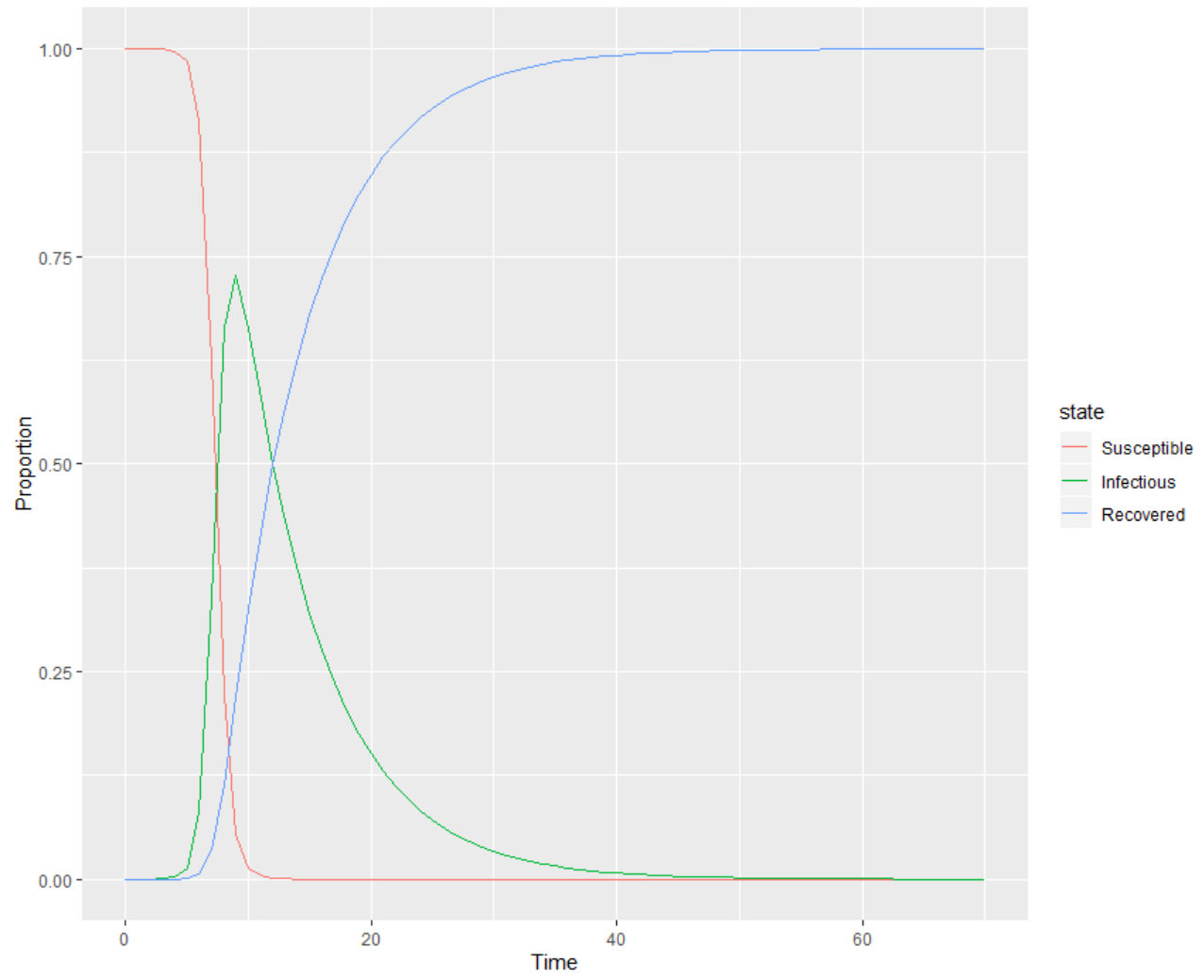
```
ggplot(tmp, aes(x = time,  
                y = proportion,  
                group = state,  
                color = state)) +  
  geom_line() +  
  labs(x = "Time", y = "Proportion")
```

Part 1: Defining the Environment

Part 2: Defining how objects are being drawn. In Rstudio console window, try typing in "geom\_"

Part 3: Defining the rest, such as labels, themes, panel arrangements, colour schemes...

# ggplot: structure



# ggplot: structure

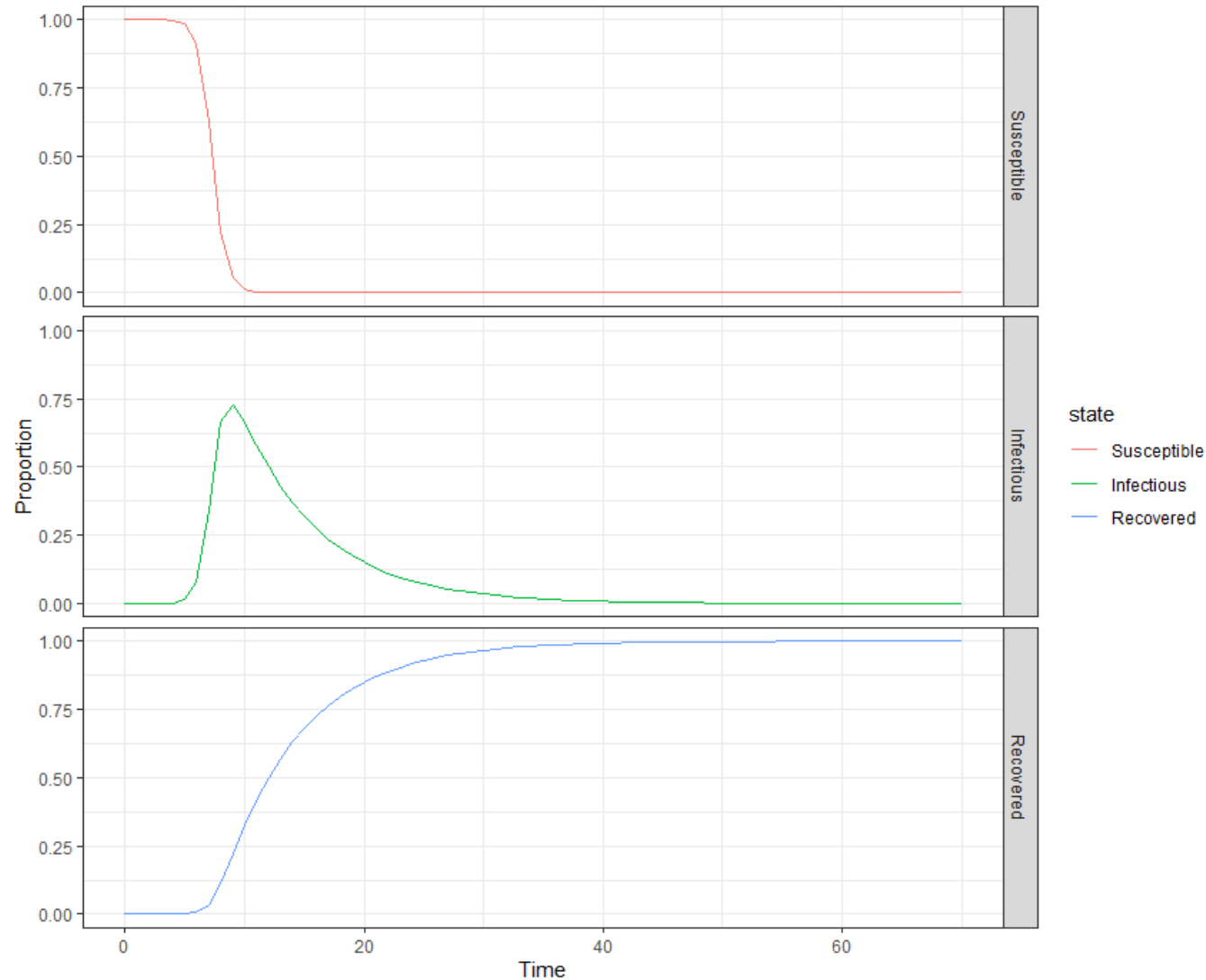
```
ggplot(tmp, aes(x = time,  
                y = proportion,  
                group = state,  
                color = state)) +  
  geom_line() +  
  labs(x = "Time", y = "Proportion") +  
  theme(legend.position = "bottom") +  
  theme_bw() +  
  facet_grid(rows = vars(state))
```

Part 1: Defining the Environment

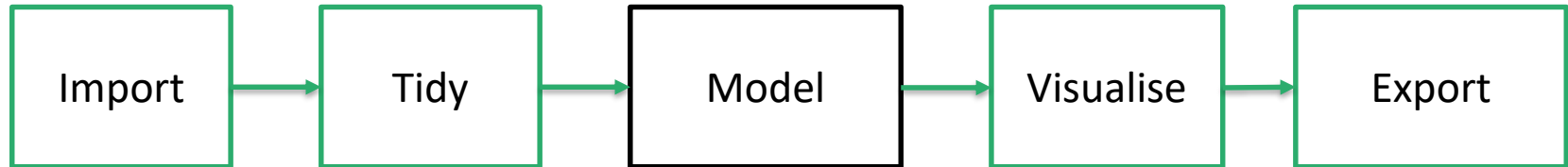
Part 2: Defining how objects are being drawn. In Rstudio console window, try typing in "geom\_"

Part 3: Defining the rest, such as labels, themes, panel arrangements, colour schemes...

# ggplot: example



```
ggplot(tmp, aes(x = time,  
                y = proportion,  
                group = state,  
                color = state)) +  
  geom_line() +  
  labs(x = "Time", y = "Proportion") +  
  theme(legend.position = "bottom") +  
  theme_bw() +  
  facet_grid(rows = vars(state))  
  
ggsave("sample.png")
```



## Questions?

# Finding Answers/ Self-learning

LONDON  
SCHOOL of  
HYGIENE  
& TROPICAL  
MEDICINE



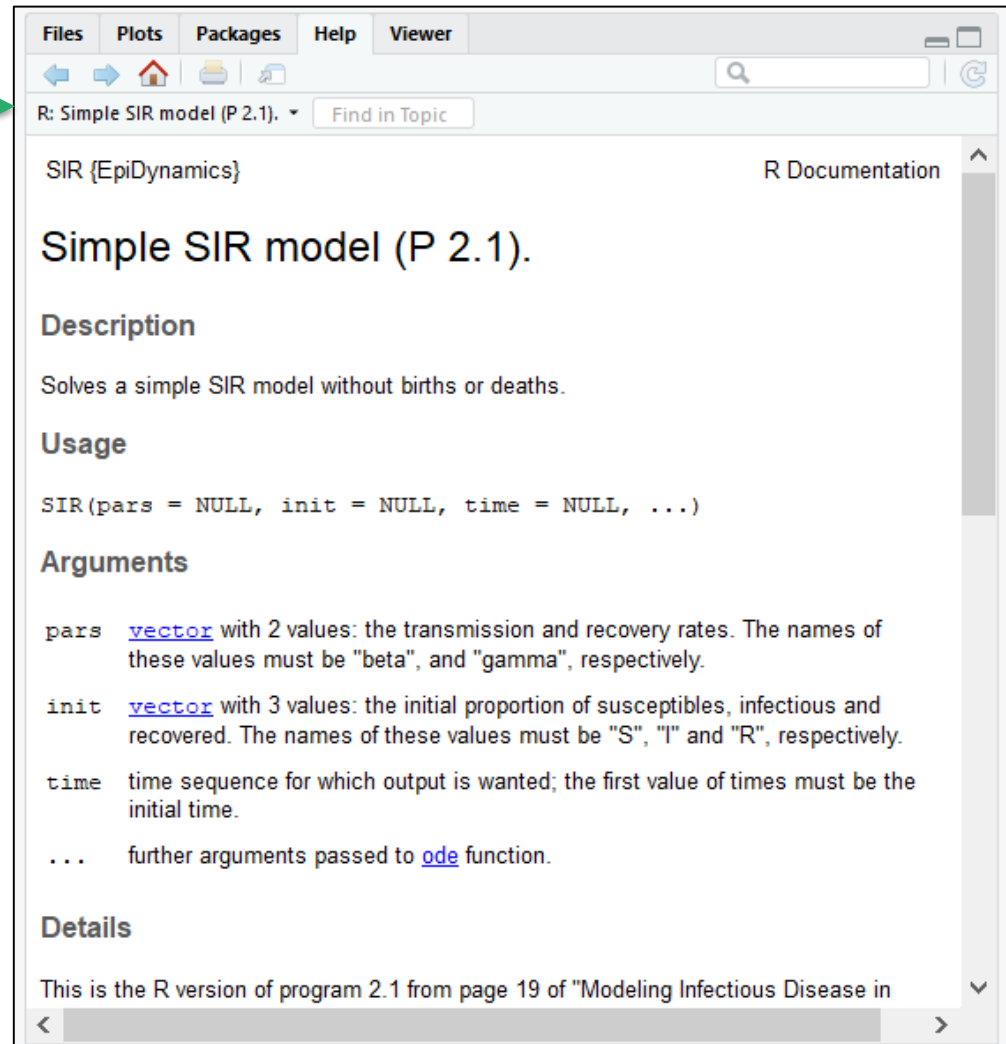


- Warning message: The code runs but something looks off.
- Error message: The code breaks somewhere. Nothing has been done.



# Within R: Option 1

> ?SIR



The screenshot shows the R help window for the `SIR` function from the `EpiDynamics` package. The window title is "R: Simple SIR model (P 2.1)". The content includes the function name, a description, usage, arguments, and details. A green arrow points from the `?SIR` command to the top of the help window.

R: Simple SIR model (P 2.1). Find in Topic

SIR {EpiDynamics} R Documentation

## Simple SIR model (P 2.1).

### Description

Solves a simple SIR model without births or deaths.

### Usage

```
SIR(pars = NULL, init = NULL, time = NULL, ...)
```

### Arguments

**pars** [vector](#) with 2 values: the transmission and recovery rates. The names of these values must be "beta", and "gamma", respectively.

**init** [vector](#) with 3 values: the initial proportion of susceptibles, infectious and recovered. The names of these values must be "S", "I" and "R", respectively.

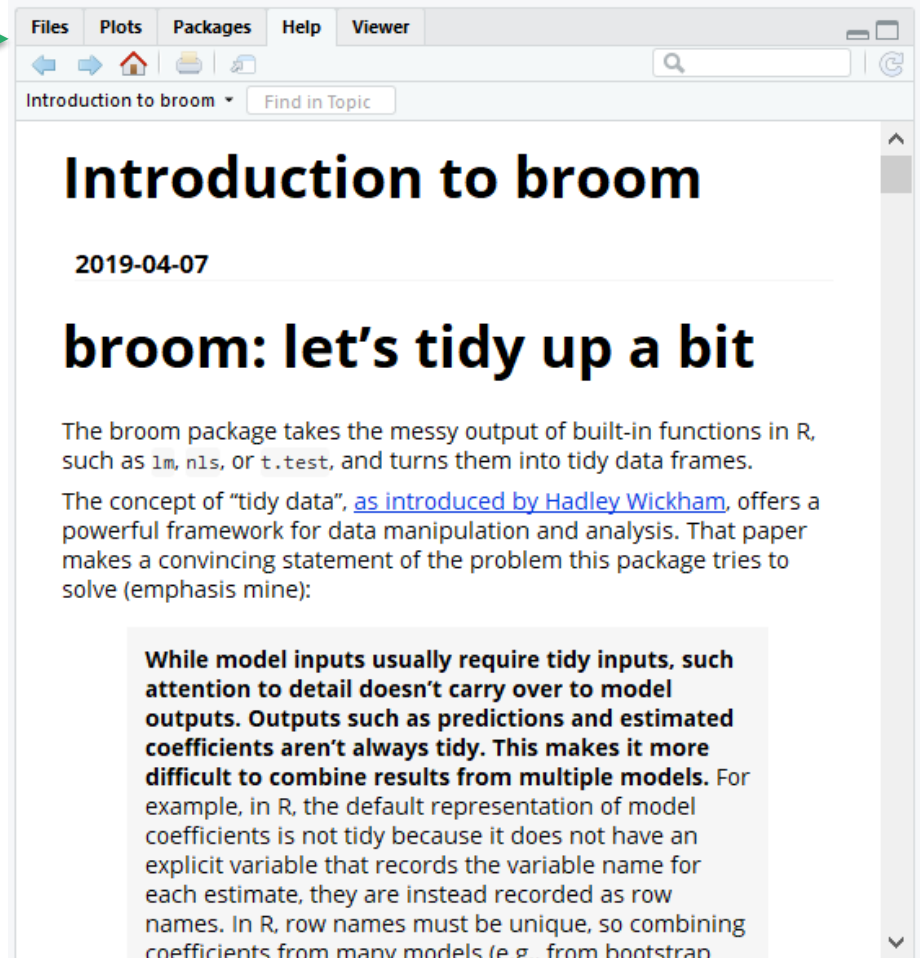
**time** time sequence for which output is wanted; the first value of times must be the initial time.

**...** further arguments passed to [ode](#) function.

### Details

This is the R version of program 2.1 from page 19 of "Modeling Infectious Disease in

```
> vignette("broom")
```



Files Plots Packages Help Viewer

Introduction to broom ▾ Find in Topic

## Introduction to broom

2019-04-07

### broom: let's tidy up a bit

The broom package takes the messy output of built-in functions in R, such as `lm`, `nls`, or `t.test`, and turns them into tidy data frames.

The concept of “tidy data”, [as introduced by Hadley Wickham](#), offers a powerful framework for data manipulation and analysis. That paper makes a convincing statement of the problem this package tries to solve (emphasis mine):

**While model inputs usually require tidy inputs, such attention to detail doesn't carry over to model outputs. Outputs such as predictions and estimated coefficients aren't always tidy. This makes it more difficult to combine results from multiple models.** For example, in R, the default representation of model coefficients is not tidy because it does not have an explicit variable that records the variable name for each estimate, they are instead recorded as row names. In R, row names must be unique, so combining coefficients from many models (e.g., from bootstrap

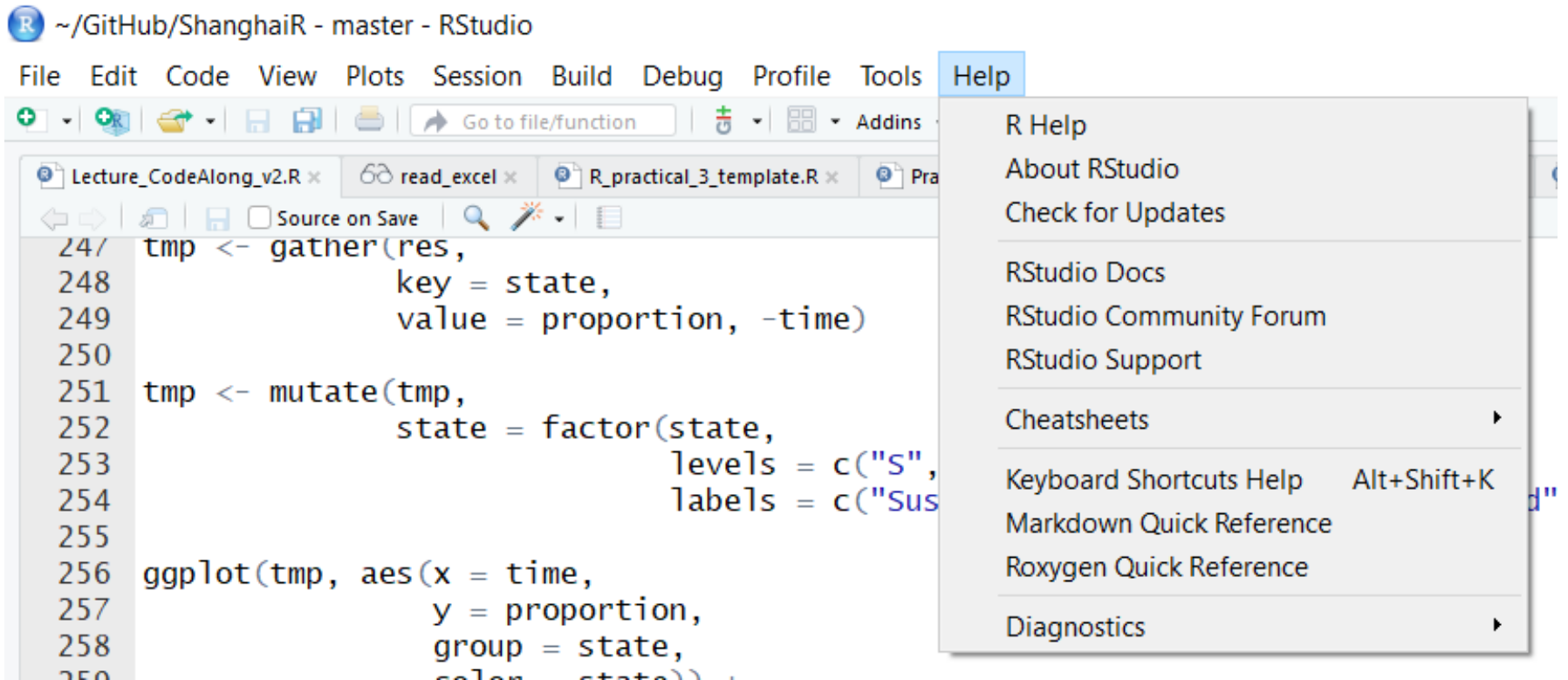
# Within R help: Option 3

> SIR

```
function (pars = NULL, init = NULL, time = NULL, ...)
{
  if (is.null(pars)) {
    stop("undefined 'pars'")
  }
  if (is.null(pars)) {
    stop("undefined 'inits'")
  }
  if (is.null(pars)) {
    stop("undefined 'time'")
  }
  function1 <- function(pars = NULL, init = NULL, time = NULL) {
    function2 <- function(time, init, pars) {
      with(as.list(c(init, pars)), {
        dS <- -beta * S * I
        dI <- beta * S * I - gamma * I
        dR <- gamma * I
        list(c(dS, dI, dR))
      })
    }
    init <- c(init["S"], init["I"], init["R"])
    output <- ode(times = time, func = function2, y = init,
      parms = pars, ...)
    return(output)
  }
  output <- function1(pars = pars, init = init, time = time)
  return(list(model = function1, pars = pars, init = init,
    time = time, results = as.data.frame(output)))
}
<bytecode: 0x00000000043785c8>
<environment: namespace:EpiDynamics>
```

Running function without  
parenthesis.

# Within R help: Option 4



The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The Help menu is open, displaying the following options: R Help, About RStudio, Check for Updates, RStudio Docs, RStudio Community Forum, RStudio Support, Cheatsheets, Keyboard Shortcuts Help (with Alt+Shift+K shortcut), Markdown Quick Reference, Roxygen Quick Reference, and Diagnostics. The background shows an R script with the following code:

```
247 tmp <- gather(res,  
248             key = state,  
249             value = proportion, -time)  
250  
251 tmp <- mutate(tmp,  
252             state = factor(state,  
253                           levels = c("S",  
254                           labels = c("Sus  
255  
256 ggplot(tmp, aes(x = time,  
257                y = proportion,  
258                group = state,  
259                color = state))
```

# Outside Help: Option 1

<https://cran.r-project.org/>



## EpiDynamics: Dynamic Models in Epidemiology

Mathematical models of infectious diseases in humans and animals. Both, deterministic and stochastic models can be simulated and plotted.

Version: 0.3.0  
Depends: R ( $\geq 3.2.2$ )  
Imports: [deSolve](#), [reshape2](#), [ggplot2](#), grid  
Published: 2015-12-03  
Author: Oswaldo Santos Baquero [aut, cre], Fernando Silveira Marques [aut]  
Maintainer: Oswaldo Santos Baquero <oswaldosant at gmail.com>  
License: [GPL-2](#) | [GPL-3](#) [expanded from: GPL ( $\geq 2$ )]  
URL: <https://github.com/oswaldosantos/EpiDynamics>  
NeedsCompilation: no  
Materials: [README](#) [NEWS](#)  
CRAN checks: [EpiDynamics results](#)

### Downloads:

Reference manual: [EpiDynamics.pdf](#)  
Package source: [EpiDynamics\\_0.3.0.tar.gz](#)  
Windows binaries: r-devel: [EpiDynamics\\_0.3.0.zip](#), r-release: [EpiDynamics\\_0.3.0.zip](#), r-oldrel: [EpiDynamics\\_0.3.0.zip](#)  
OS X binaries: r-release: [EpiDynamics\\_0.3.0.tgz](#), r-oldrel: [EpiDynamics\\_0.3.0.tgz](#)  
Old sources: [EpiDynamics archive](#)

### Linking:

Please use the canonical form <https://CRAN.R-project.org/package=EpiDynamics> to link to this page.

# Outside Help: Option 2



<https://stackoverflow.com/questions/tagged/r>

stackoverflow

ProductsCustomersUse cases

Q [r]

Log inSign up

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

TEAMS

What's this?

Q&A for Work

Questions tagged [r]

Ask Question

R is a free, open-source programming language and software environment for statistical computing, bioinformatics, visualization, and general computing. Please provide minimal and reproducible example(s) along with the desired output. Use `dput()` for data and specify all non-base packages with `library()` calls. Do not embed pictures for data or code, use indented code blocks instead. For statistics related questions, use <https://stats.stackexchange.com>.

[Learn more...](#) [Top users](#) [Synonyms \(2\)](#) [r jobs](#)

301,483 questions

NewestActiveBounties 6UnansweredMore Filter

0 votes

0 answers

3 views

R Document Term Matrix Truncating Words

Working with a document term matrix in R seems to be truncating the words. I create a document term matrix from a corpus like below: `library(tm) docs <- c("All that we are is the result of what ...`

r

tm

asked 1 min ago

tim

361 • 1 • 17

0 votes

0 answers

12 views

Build a function which gives table of distribution percentage of variable dynamically

Trying show what I want using iris for ease of understanding. I'll need two categorical columns hence I'll make one using below: Re-coding `iris$Species` into `species_rec` with only two category & ...

r

data.table

tidyverse

janitor

asked 15 mins ago

Vaibhav Singh

110 • 13

Blog

What Every Developer Should Learn Early On

DEF CON and Stack Overflow: What Our Traffic Says About Cybersecurity...

Featured on Meta

Employee profiles are now marked with a "Staff" indicator

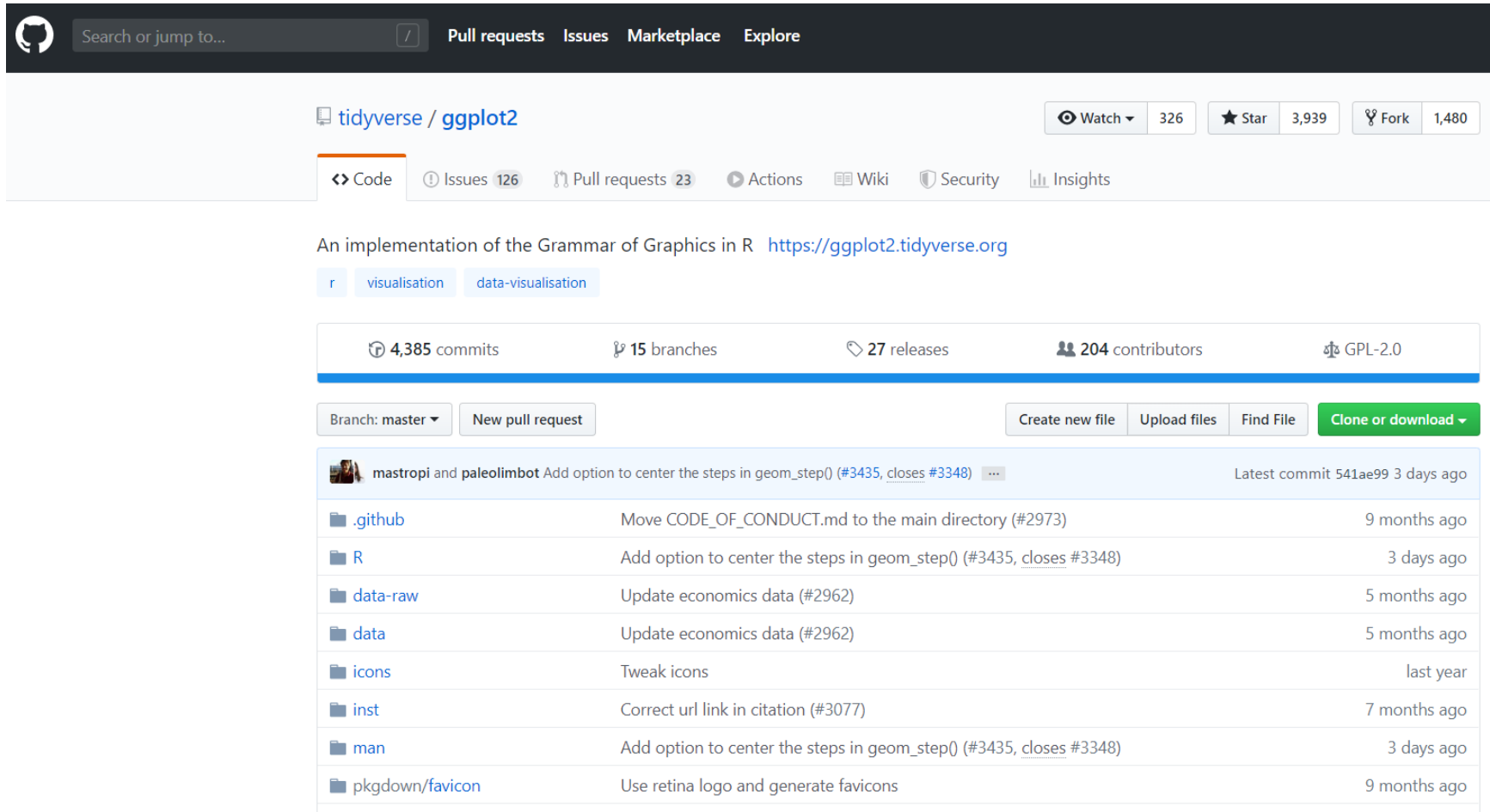
Congratulations to our 29 oldest beta sites - They're now no longer beta!

Should we burninate the [sales] tag?

Experiment: closing and reopening happens at 3 votes for the next 30 days

# Outside of R: Option 3

<https://github.com/>



The screenshot shows the GitHub repository page for `tidyverse/ggplot2`. The page header includes the GitHub logo, a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. The repository name `tidyverse / ggplot2` is displayed, along with statistics: 326 Watchers, 3,939 Stars, and 1,480 Forks. Below this, tabs for Code, Issues (126), Pull requests (23), Actions, Wiki, Security, and Insights are visible. The description states: "An implementation of the Grammar of Graphics in R" with a link to <https://ggplot2.tidyverse.org>. Tags for `r`, `visualisation`, and `data-visualisation` are present. Repository statistics show 4,385 commits, 15 branches, 27 releases, 204 contributors, and the GPL-2.0 license. Action buttons include "Branch: master", "New pull request", "Create new file", "Upload files", "Find File", and "Clone or download". A commit history table follows, showing the latest commit by `mastropi` and `paleolimbot` to "Add option to center the steps in geom\_step()" 3 days ago. The table lists files and their corresponding commit messages and dates.

File	Commit Message	Time Ago
<code>.github</code>	Move CODE_OF_CONDUCT.md to the main directory (#2973)	9 months ago
<code>R</code>	Add option to center the steps in <code>geom_step()</code> (#3435, closes #3348)	3 days ago
<code>data-raw</code>	Update economics data (#2962)	5 months ago
<code>data</code>	Update economics data (#2962)	5 months ago
<code>icons</code>	Tweak icons	last year
<code>inst</code>	Correct url link in citation (#3077)	7 months ago
<code>man</code>	Add option to center the steps in <code>geom_step()</code> (#3435, closes #3348)	3 days ago
<code>pkgdown/favicon</code>	Use retina logo and generate favicons	9 months ago



# Outside of R: Option 4



<https://r4ds.had.co.nz/>

## R for Data Science

Welcome

1 Introduction

I Explore

2 Introduction

3 Data visualisation

4 Workflow: basics

5 Data transformation

6 Workflow: scripts

7 Exploratory Data Analysis

8 Workflow: projects

II Wrangle

9 Introduction

10 Tibbles

11 Data import

12 Tidy data

13 Relational data

14 Strings

15 Factors



R for Data Science

## R for Data Science

*Garrett Grolemund*

*Hadley Wickham*

## Welcome

This is the website for “**R for Data Science**”. This book will teach you how to do data science with R: You’ll learn how to get your data into R, get it into the most useful structure, transform it, visualise it and model it. In this book, you will find a practicum of skills for data science. Just as a chemist learns how to clean test tubes and stock a lab, you’ll learn how to clean data and draw plots—and many other things besides. These are the skills that allow data science to happen, and here you will find the best practices for doing each of these things with R. You’ll learn how to use the grammar of graphics, literate programming, and reproducible research to save time. You’ll also learn how to manage cognitive resources to facilitate discoveries when wrangling, visualising, and exploring data.

O'REILLY



R for Data  
Science

VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA

Hadley Wickham &



Questions?

LONDON  
SCHOOL of  
HYGIENE  
& TROPICAL  
MEDICINE

