



Introduction To R

Roy Francis • 16-Sep-2021

<https://royfrancis.github.io/course-r>

roy.francis@nbis.se

Contents

- Getting Started
- Variables & Operators
- Data Types
- Datatype Conversion
- Functions
- Control Structures
- R Packages
- Base Graphics
- Grid Graphics
- Input & Output
- Rmarkdown
- Tidyverse
- Bioconductor
- Exercises/Lab
- Help & Learning R

Topics

- R & RStudio environment
- Running code, scripting, sourcing script
- Variables and operators
- Data types & data type conversion
- Reusing code using functions
- Base and grid graphics
- Input & output of text & graphics
- Reproducible analyses, Rmarkdown, notebooks and reports
- Tidyverse: Modern R programming paradigm

What? Why R?

R is a language and environment for statistical computing and graphics.

- Command line interface

Pros

- Data analysis
- Statistics
- High quality graphics
- Huge number of packages
- R is popular
- Reproducible research
- RStudio IDE
- FREE! Open source

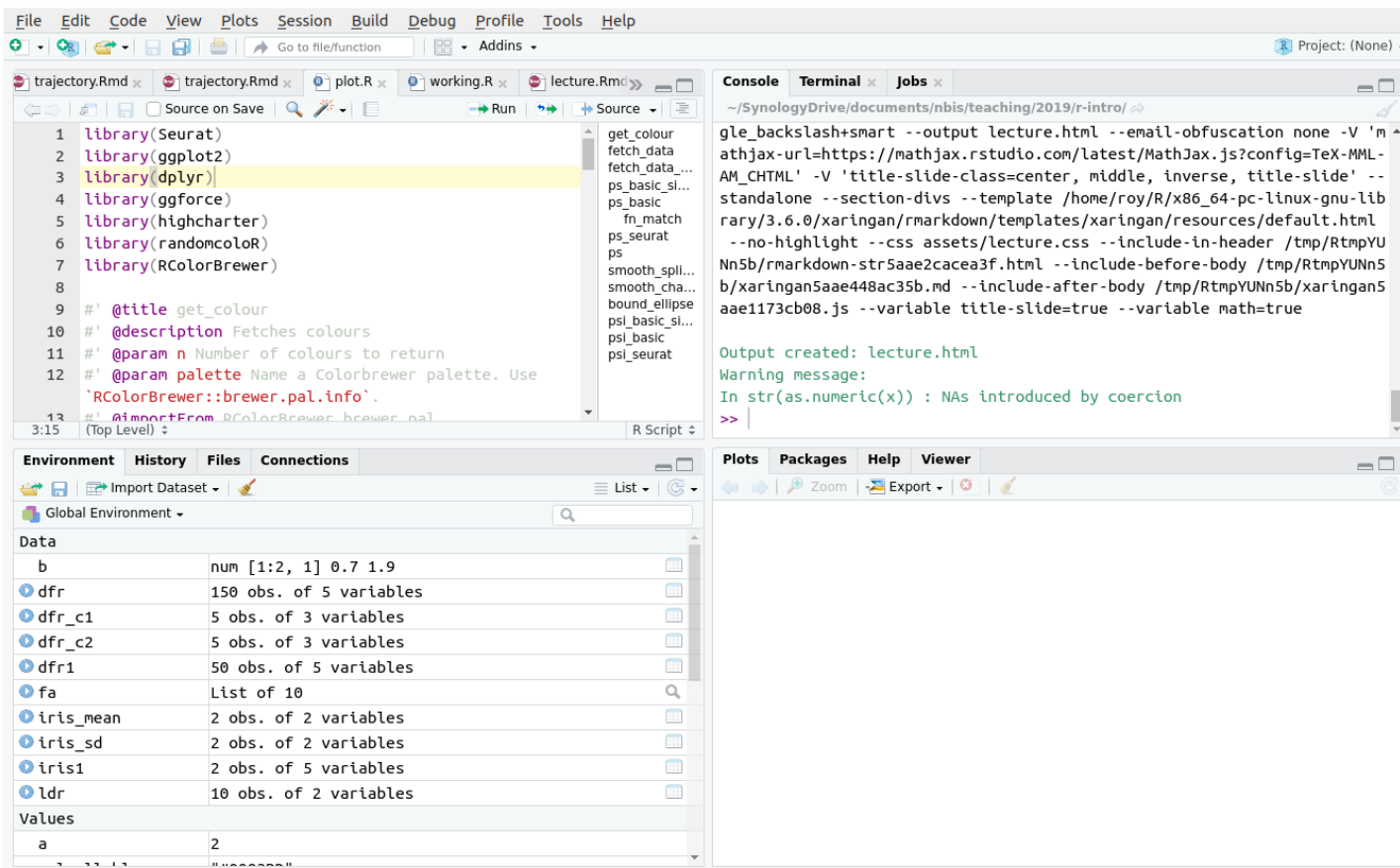


Cons

- Steep learning curve
- Not elegant/consistent
- Slow

Installation

- Install R from [r-project.org](https://www.r-project.org).
- Install **RStudio IDE**
- Code editor, highlighting, projects, version control, package building, debugger, profiler



The screenshot displays the RStudio IDE interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The toolbar contains icons for file operations, running code, and other functions. The main editor window shows a script with R code for loading libraries (Seurat, ggplot2, dplyr, ggforce, highcharter, randomcolor, RColorBrewer) and a function definition for 'get_colour'. The console on the right shows the output of the 'get_colour' function, including a warning message about NAs introduced by coercion. The environment pane at the bottom left lists the global environment and data objects, including 'b', 'dfr', 'dfr_c1', 'dfr_c2', 'dfr1', 'fa', 'iris_mean', 'iris_sd', 'iris1', 'ldr', and 'a'.

```
1 library(Seurat)
2 library(ggplot2)
3 library(dplyr)
4 library(ggforce)
5 library(highcharter)
6 library(randomcolor)
7 library(RColorBrewer)
8
9 #' @title get_colour
10 #' @description Fetches colours
11 #' @param n Number of colours to return
12 #' @param palette Name a Colorbrewer palette. Use
13 #' @importFrom RColorBrewer brewer_pal
14
15 get_colour
16 fetch_data
17 fetch_data
18 ps_basic_sl...
19 ps_basic
20 fn_match
21 ps_seurat
22 ps
23 smooth_spl...
24 smooth_cha...
25 bound_ellipse
26 psi_basic_sl...
27 psi_basic
28 psi_seurat
```

Output created: lecture.html
Warning message:
In str(as.numeric(x)) : NAs introduced by coercion

Object	Type	Size
b	num [1:2, 1]	0.7 1.9
dfr	150 obs. of 5 variables	
dfr_c1	5 obs. of 3 variables	
dfr_c2	5 obs. of 3 variables	
dfr1	50 obs. of 5 variables	
fa	List of 10	
iris_mean	2 obs. of 2 variables	
iris_sd	2 obs. of 2 variables	
iris1	2 obs. of 5 variables	
ldr	10 obs. of 2 variables	

Interaction

- Execute commands directly in Console
- Ready console shows `>`
- Console shows `+` when waiting for information
- Press `Esc` to escape from `+` to `>`
- Save commands by writing scripts
- Run lines using `Ctrl + Enter`
- Run entire script using `Ctrl + Shift + Enter`

Variables & Operators

- Assign variables using `<-`, `=` or `->`

```
x <- 4
x = 4
x
```

```
## [1] 4
```

- Arithmetic operators

```
x <- 4; y <- 2;
x + y # add
x - y # subtract
x * y # multiply
x / y # divide
x %% y # modulus
x ^ y # power
```

```
## [1] 6
## [1] 2
## [1] 8
## [1] 2
## [1] 0
## [1] 16
```

- Logical operators return `TRUE` or `FALSE`

```
x == y # equal to?
x != y # not equal to?
x > y # greater than?
x < y # less than?
x >= y # greater than or equal to?
x <= y # less than or equal to?
```

```
## [1] FALSE
## [1] TRUE
## [1] TRUE
## [1] FALSE
## [1] TRUE
## [1] FALSE
```

```
T | F # OR
T & F # AND
```

```
## [1] TRUE
## [1] FALSE
```

- `||`, `&&`, `!`, `any()`, `all` for logical vectors

Variables & Operators

- `:` operator is used for generating regular sequences
- `::` & `:::` are used for accessing functions
- `%*%` used for matrix multiplication
- `%in%` used as a set operator

```
"a" %in% c("x", "p", "a", "c")
```

```
## [1] TRUE
```

- Avoid conflicting built-in variable names like `c`, `t`, `q` etc
- Variable names cannot start with a number

Data Type • Overview

```
## dimension    homogenous heterogenous
## 1            1D Atomic vector      List
## 2            2D      Matrix    Data.frame
## 3            nD      Array
```

- Use `typeof()` to find type of a variable

```
x <- 4; typeof(x)
```

```
## [1] "double"
```

```
y <- "this"; typeof(y)
```

```
## [1] "character"
```

```
mode(x); class(x)
str(x); structure(x)
```

```
## [1] "numeric"
## [1] "numeric"
## num 4
## [1] 4
```

Data Type • Basic

- Mode

```
mode(1.0)
mode(1L)
mode("hello")
mode(factor(1))
mode(T)
```

```
## [1] "numeric"
## [1] "numeric"
## [1] "character"
## [1] "numeric"
## [1] "logical"
```

- Type

```
typeof(1.0)
typeof(1L)
typeof("hello")
typeof(factor(1))
typeof(T)
```

```
## [1] "double"
## [1] "integer"
## [1] "character"
## [1] "integer"
## [1] "logical"
```

Data Type • Missing Values

- R explicitly handles missing data as `NA` and undefined data as `NULL` (NA vs NULL)
- NA is not 0
- NA is not ""
- NA is not FALSE
- NA is not NULL
- Operations that involve NA may or may not result in an NA

```
NA==1
sum(c(2,6,NA,6))
sum(c(2,6,NA,6),na.rm=TRUE)
NA|NA
NA|TRUE
NA&TRUE
NULL|TRUE
```

```
## [1] NA
## [1] NA
## [1] 14
## [1] NA
## [1] TRUE
## [1] NA
## logical(0)
```

Data Type • Vector • Create

- Vector stores multiple values
- Concatenate variables, values and vectors using the function `c()`

```
x <- c(2,3,4,5,6)
y <- c("a","c","d","e")
x
y
```

```
## [1] 2 3 4 5 6
## [1] "a" "c" "d" "e"
```

- Few different ways to create vectors.

```
c(2,3,5,6)
2:8
seq(2,5,by=0.5)
rep(1:3,times=2)
```

```
## [1] 2 3 5 6
## [1] 2 3 4 5 6 7 8
## [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
## [1] 1 2 3 1 2 3
```

Data Type • Vector • Access

- Access vectors using the `[]` operator.

```
x[1]; y[3]
```

```
## [1] 2  
## [1] "d"
```

- Function `c()` to specify multiple positions.

```
x[c(1,3)]
```

```
## [1] 2 4
```

- Vectorised operation

```
x <- c(2,3,4,5); y <- c(9,8,7,6)  
x+y
```

```
z <- c("a", "an", "a", "a"); k <- c("boy", "apple", "girl", "mess")  
paste(z,k)
```

```
## [1] 11 11 11 11  
## [1] "a boy" "an apple" "a girl" "a mess"
```

Data Type • Vector

- Verify data type

```
x <- c(2,3,4,5)
z <- c("a", "an", "a", "a")
```

```
mode(x)
mode(z)
str(x)
str(z)
```

```
## [1] "numeric"
## [1] "character"
##  num [1:4] 2 3 4 5
##  chr  [1:4] "a" "an" "a" "a"
```

```
is.atomic(x)
is.numeric(x)
is.character(z)
```

```
## [1] TRUE
## [1] TRUE
## [1] TRUE
```

Data Type • Factor

- Factors store categorical data

```
x <- factor(c("a","b","b","c","c"))  
class(x)  
str(x)
```

```
## [1] "factor"  
## Factor w/ 3 levels "a","b","c": 1 2 2 3 3
```

- Factor  has 3 categories (3 levels)

```
levels(x)
```

```
## [1] "a" "b" "c"
```

- Verify if an R object is a factor

```
is.factor(x)
```

```
## [1] TRUE
```

Data Type • Matrix • Create

- Create a matrix from vector

```
x <- matrix(c(2,3,4,5,6,7))
x
```

```
##      [,1]
## [1,]    2
## [2,]    3
## [3,]    4
## [4,]    5
## [5,]    6
## [6,]    7
```

- Matrix has rows and columns

```
dim(x) # dimensions
nrow(x) # number of rows
ncol(x) # number of columns
```

```
## [1] 6 1
## [1] 6
## [1] 1
```

- Specify rows and columns

```
x <- matrix(c(2,3,4,5,6,7),nrow=3,
             ncol=2,byrow=TRUE)
x
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    4    5
## [3,]    6    7
```

```
str(x)
```

```
##  num [1:3, 1:2] 2 4 6 3 5 7
```

- Verify if an R object is a matrix

```
is.matrix(x)
```

```
## [1] TRUE
```


Data Type • Matrix • Access

- Access matrix using `[]` operator as `[row,col]`

```
x[2,2]
```

```
## [1] 5
```

- Get whole row/col using `[row,]` or `[,col]`

```
x[1,]  
x[,2]
```

```
## [1] 2 3  
## [1] 3 5 7
```

- Use `drop=FALSE` to retain a matrix as `[row,col,drop=FALSE]`

```
x[1,,drop=F]
```

```
##      [,1] [,2]  
## [1,]    2    3
```

Data Type • Matrix • Label

- Add row/column names

```
rownames(x) <- c("a", "b", "c")
colnames(x) <- c("k", "p")
x
```

```
##   k p
## a 2 3
## b 4 5
## c 6 7
```

- Get row/column labels

```
rownames(x)
colnames(x)
dimnames(x)
```

```
## [1] "a" "b" "c"
## [1] "k" "p"
## [[1]]
## [1] "a" "b" "c"
##
## [[2]]
## [1] "k" "p"
```

- Access using labels

```
x["b",]
x[, "p"]
```

```
## k p
## 4 5
## a b c
## 3 5 7
```

Data Type • List

- Create using `list()`.

```
x <- list(c(2,3,4,5),c("a","b","c","d"),
         factor(c("a","a","b")),
         matrix(c(3,2,3,5,6,7),ncol=2))
x
```

```
## [[1]]
## [1] 2 3 4 5
##
## [[2]]
## [1] "a" "b" "c" "d"
##
## [[3]]
## [1] a a b
## Levels: a b
##
## [[4]]
##      [,1] [,2]
## [1,]    3    5
## [2,]    2    6
## [3,]    3    7
```

```
typeof(x); class(x);
```

```
## [1] "list"
## [1] "list"
```

- Access lists using `[]` and `[[[]]`

```
x[1]
```

```
## [[1]]
## [1] 2 3 4 5
```

- Lists are recursive

```
x <- list(list(list(list())))
str(x)
```

```
## List of 1
## $ :List of 1
## ..$ :List of 1
## .. ..$ : list()
```

Data Type • `data.frame` • Create

```
dfr <- data.frame(x = 1:3, y = c("a", "b", "c")); dfr
```

```
##   x y  
## 1 1 a  
## 2 2 b  
## 3 3 c
```

```
str(dfr)
```

```
## 'data.frame':   3 obs. of  2 variables:  
## $ x: int  1 2 3  
## $ y: chr  "a" "b" "c"
```

- Use `stringsAsFactors=FALSE` to avoid auto factor conversion

```
dfr <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = F)  
str(dfr)  
is.data.frame(dfr)
```

```
## 'data.frame':   3 obs. of  2 variables:  
## $ x: int  1 2 3  
## $ y: chr  "a" "b" "c"  
## [1] TRUE
```

Data Type • `data.frame` • Access

- Access using `[]` or `$` operator

```
dfr$x  
dfr$y
```

```
## [1] 1 2 3  
## [1] "a" "b" "c"
```

- `head()` / `tail()` functions show first/last six lines
- Subset a `data.frame` using `subset()`

```
subset(dfr,dfr$y=="a")
```

```
##   x y  
## 1 1 a
```

Data Type • Conversion

```
x <- c(1,2,3); str(x)
```

```
##  num [1:3] 1 2 3
```

- Convert to character

```
y <- as.character(x); str(y)
```

```
##  chr [1:3] "1" "2" "3"
```

- Character coerced (if possible) to number

```
x <- c("1","2","hello"); str(x)
```

```
##  chr [1:3] "1" "2" "hello"
```

```
str(as.numeric(x))
```

```
##  num [1:3] 1 2 NA
```

Data Type • Conversion

Many other conversion functions

- `as.matrix()`
- `as.data.frame()`
- `as.integer()`
- `as.Date()`
- `as.factor()`
- `as.list()`
- `as.vector()`

Functions • Built-In • Math

```
# generate 10 random numbers between 1 and 200
x <- sample(x=1:200,10); x;
```

```
## [1] 7 108 40 120 107 151 24 156 111 74
```

```
sum(x) # sum
mean(x) # mean
min(x) # min
log(x) # log
exp(x) # exponent
sqrt(x) # square-root
round(x) # round
sort(x) # sort
```

```
## [1] 898
## [1] 89.8
## [1] 7
## [1] 1.945910 4.682131 3.688879 4.787492 4.672829 5.017280 3.178054 5.049856
## [9] 4.709530 4.304065
## [1] 1.096633e+03 8.013164e+46 2.353853e+17 1.304181e+52 2.947878e+46
## [6] 3.788495e+65 2.648912e+10 5.622626e+67 1.609487e+48 1.373383e+32
## [1] 2.645751 10.392305 6.324555 10.954451 10.344080 12.288206 4.898979
## [8] 12.489996 10.535654 8.602325
## [1] 7 108 40 120 107 151 24 156 111 74
## [1] 7 24 40 74 107 108 111 120 151 156
```


Functions • Built-In • String

```
paste("hello","kitty") # join
grep("hell","hello") # find a pattern
nchar("hello") # number of characters
toupper("hello") # to uppercase
tolower("HELLO") # to lowercase
sub("ell","ipp","hello") # replace pattern
substr("hello",start=1,stop=3) # substring
strsplit("sunny&bunny&funny","&") # split a character
```

```
## [1] "hello kitty"
## [1] 1
## [1] 5
## [1] "HELLO"
## [1] "hello"
## [1] "hippo"
## [1] "hel"
## [[1]]
## [1] "sunny" "bunny" "funny"
```

- `print()` & `cat()` are useful for printing messages
- `\n` newline character

Functions • Custom

Code can be re-used by converting them to functions.

```
a <- 1:6; b <- 8:10  
d <- a*b  
e <- log(d)  
e
```

```
## [1] 2.079442 2.890372 3.401197 3.465736 3.806662 4.094345
```

- Custom function definition

```
my_function <- function(a,b){  
  d <- a*b  
  e <- log(d)  
  return(e)  
}
```

- Re-use function

```
my_function(a=2:4,b=6:8)
```

```
## [1] 2.484907 3.044522 3.465736
```

- Function names must not start with number

Control Structure • **if**

- Conditional statements using **if()**

```
a <- 2; b <- 5;  
if(a < b) print(paste(a,"is smaller than",b))
```

```
## [1] "2 is smaller than 5"
```

- Use **else** for alternative output

```
if(a < b) {  
  print(paste(a,"is smaller than",b))  
}else{  
  print(paste(b,"is smaller than",a))  
}
```

```
## [1] "2 is smaller than 5"
```

- Chain **if else** statements

```
grade <- "B"  
  
if(grade == "A"){  
  print("Grade is Excellent!")  
}else if(grade == "B"){  
  print("Grade is Good.")  
}else if (grade == "C") {  
  print("Grade is Alright.")  
}
```

```
## [1] "Grade is Good."
```

Control Structure • **for**

- Use **for()** loop for known number of iterations

```
for (i in 1:5){  
  print(i)  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5
```

- Use **while()** loop for unknown number of iterations

```
i <- 1  
while(i < 5){  
  print(i)  
  i <- i+1  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4
```

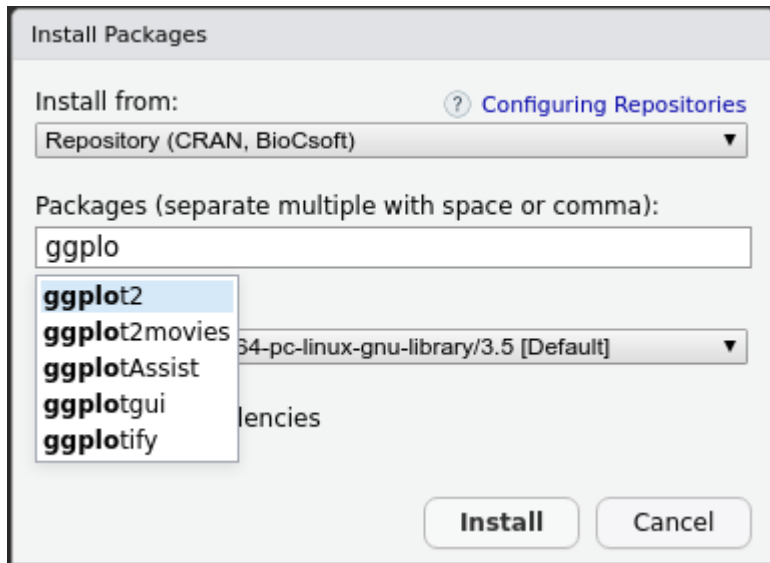
R Packages

- **CRAN** (The Comprehensive R Archive Network); Use `install.packages()`

```
install.packages("ggplot2",dependencies=TRUE)
```

- For local packages, use `type="source"`

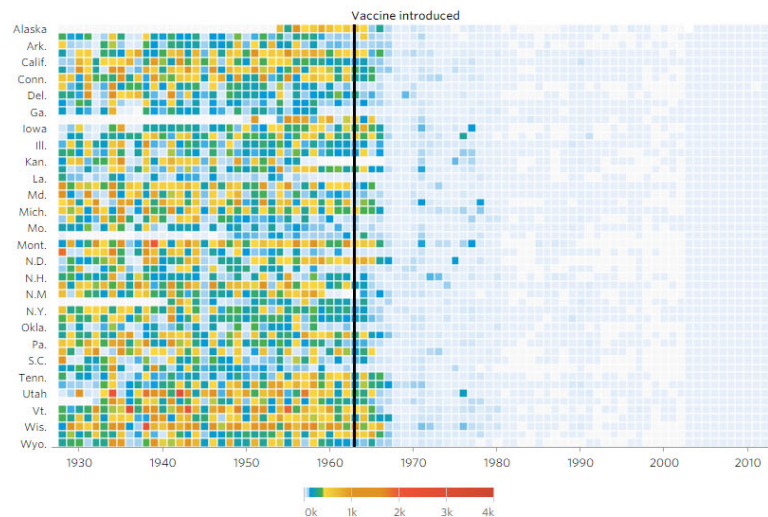
```
install.packages(path="./dir/package.zip",type="source")
```



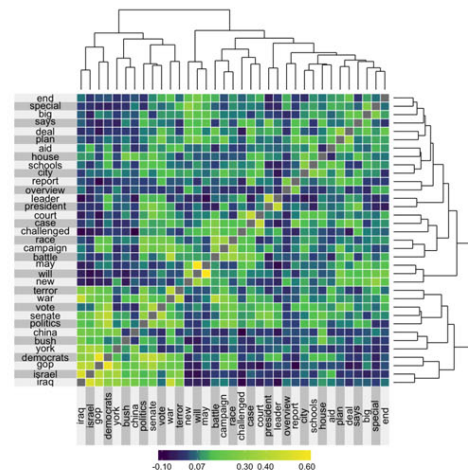
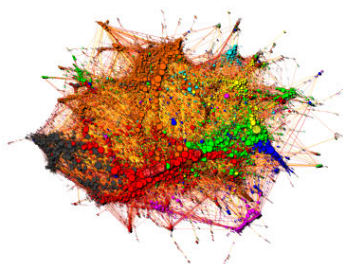
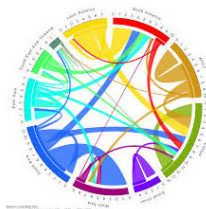
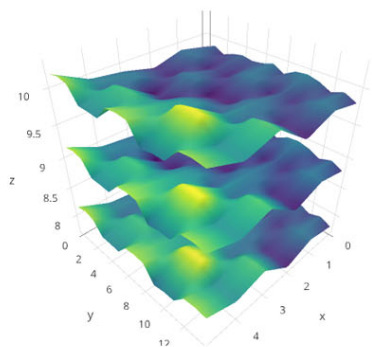
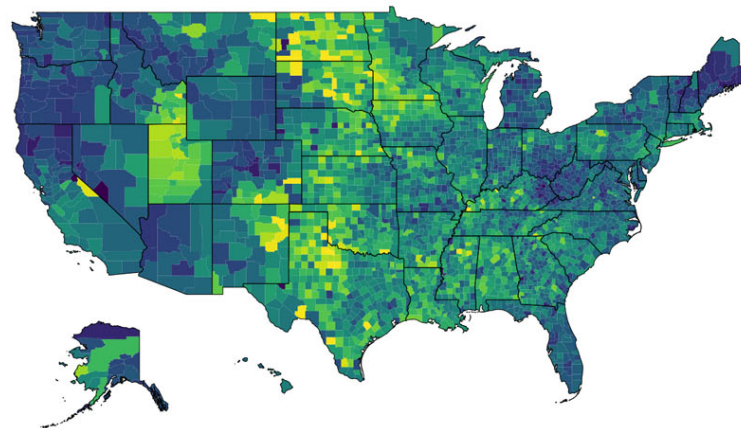
- **Bioconductor** for Biology/Bioinformatics packages; Use `BiocManager::install()`
- For GitHub packages, Use `devtools::install_github()`

Graphics

Measles

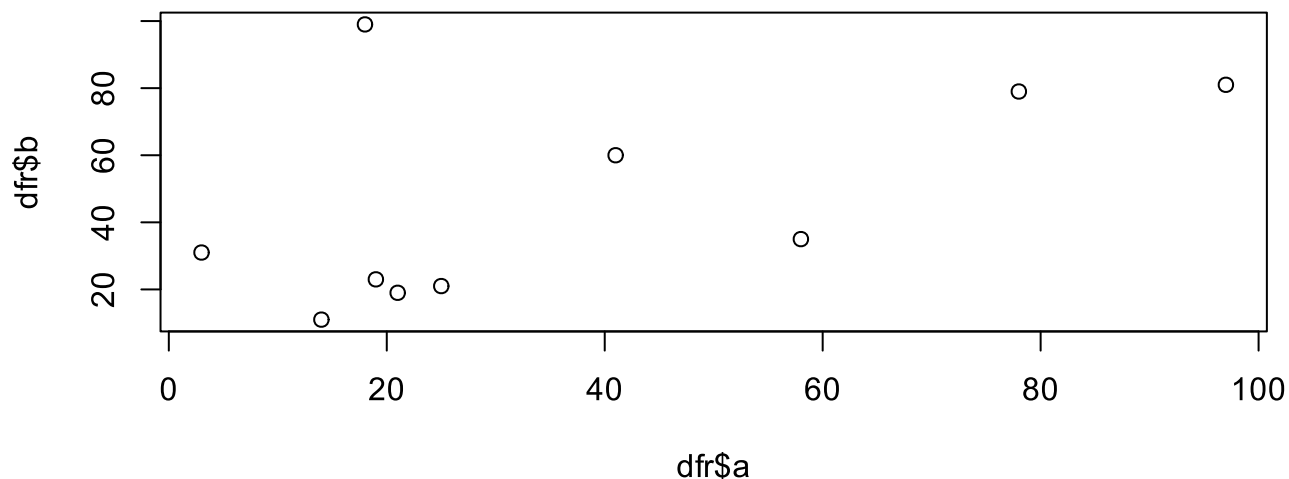


Total Religious Adherents by County



Graphics • Base

```
dfr <- data.frame(a=sample(1:100,10),b=sample(1:100,10))  
plot(dfr$a,dfr$b)
```



- Add axes labels etc

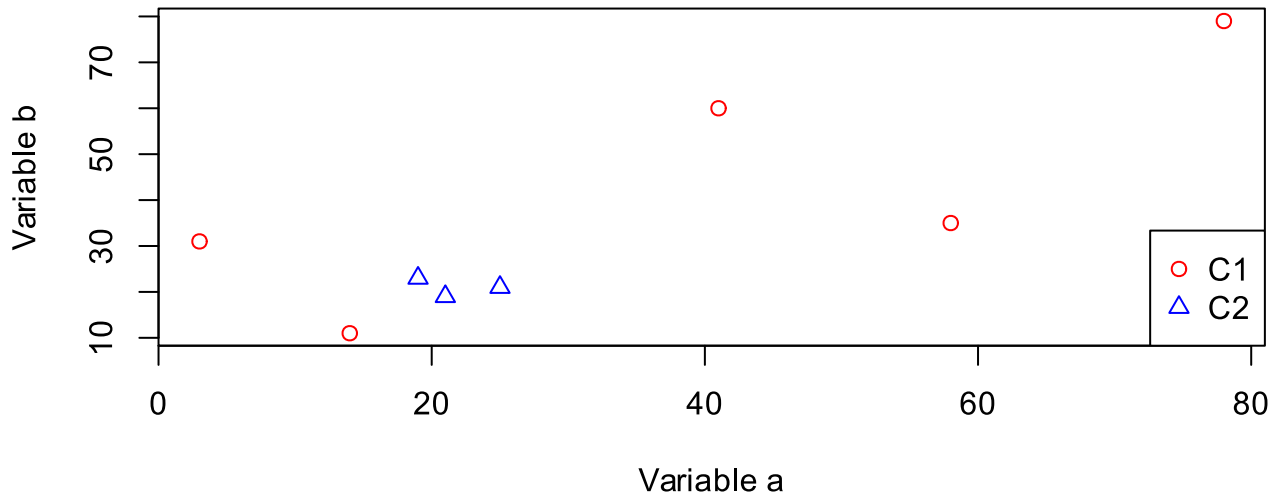
```
plot(dfr$a,dfr$b,xlab="Variable a",ylab="Variable b")  
plot(dfr$a,dfr$b,xlab="Variable a",ylab="Variable b",type="b")
```

Graphics • Base

```
dfr$cat <- rep(c("C1","C2"),each=5) # add category

# subset data
dfr_c1 <- subset(dfr,dfr$cat == "C1")
dfr_c2 <- subset(dfr,dfr$cat == "C2")

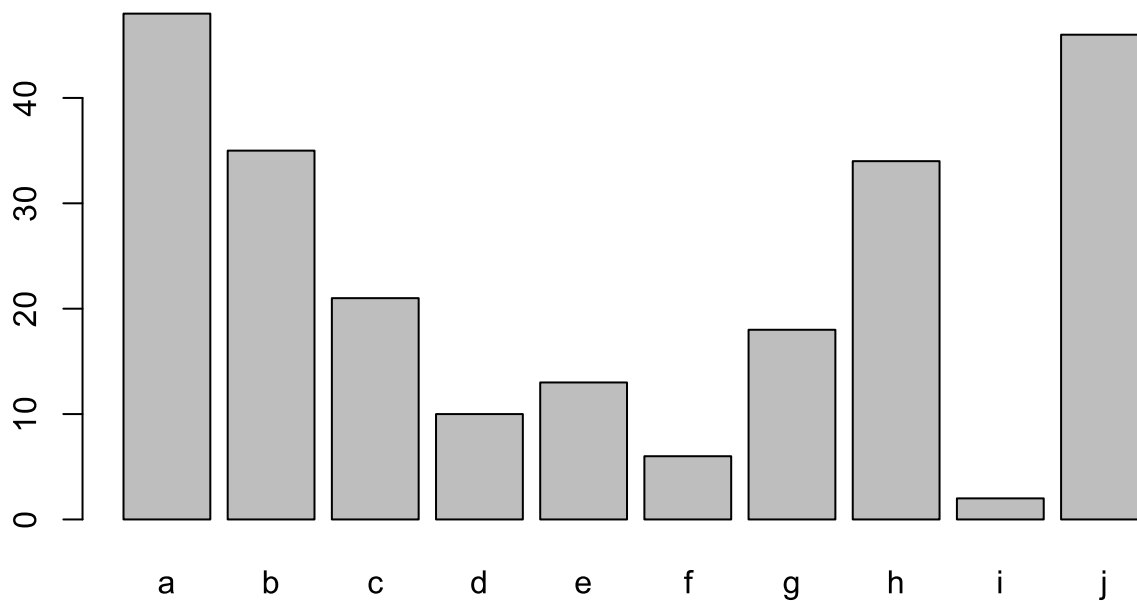
plot(dfr_c1$a,dfr_c1$b,xlab="Variable a",ylab="Variable b",col="red",pch=1)
points(dfr_c2$a,dfr_c2$b,col="blue",pch=2)
legend(x="bottomright",legend=c("C1","C2"),
      col=c("red","blue"),pch=c(1,2))
```



Graphics • Base

- Barplot

```
ldr <- data.frame(a=letters[1:10],b=sample(1:50,10))  
barplot(ldr$b,names.arg=ldr$a)
```



ggplot2:

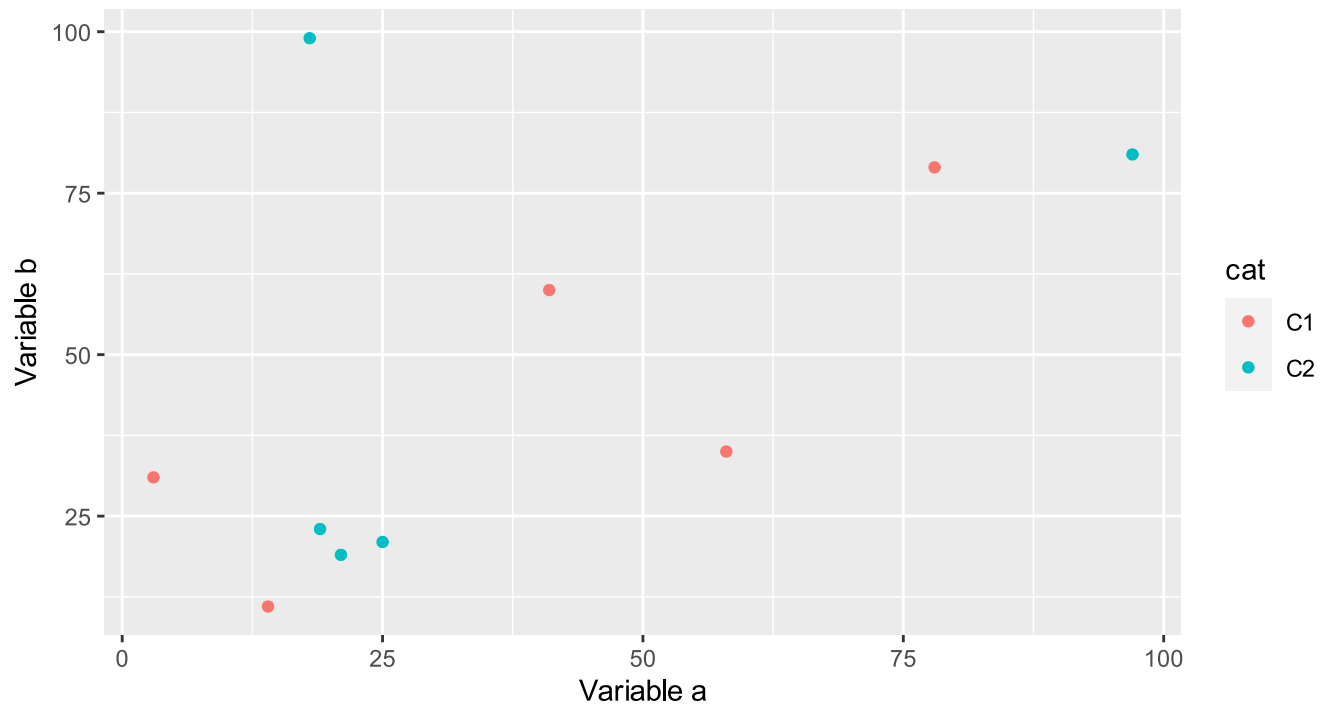
VISUAL DATA
EXPLORATION



Graphics • **ggplot2**

```
library(ggplot2)

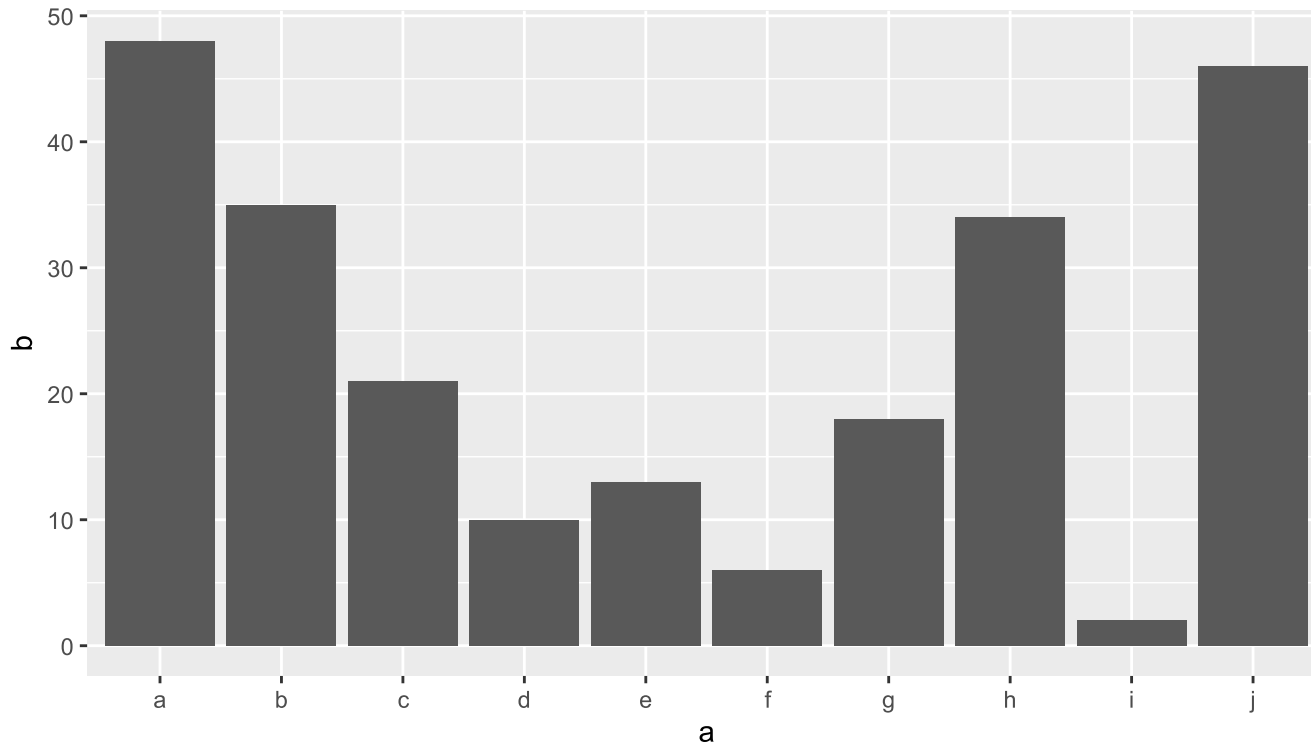
ggplot(dfr, aes(x=a, y=b, colour=cat))+
  geom_point()+
  labs(x="Variable a", y="Variable b")
```



Graphics • **ggplot2**

- Barplot

```
ggplot(ldr, aes(x=a, y=b)) +  
  geom_bar(stat="identity")
```



Input & Output • Text

```
dfr <- read.table("iris.txt",header=TRUE,stringsAsFactors=F)
head(dfr); str(dfr)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1         5.1         3.5         1.4         0.2   setosa
## 2         4.9         3.0         1.4         0.2   setosa
## 3         4.7         3.2         1.3         0.2   setosa
## 4         4.6         3.1         1.5         0.2   setosa
## 5         5.0         3.6         1.4         0.2   setosa
## 6         5.4         3.9         1.7         0.4   setosa
## 'data.frame':   150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species      : chr  "setosa" "setosa" "setosa" "setosa" ...
```

```
dfr1 <- dfr[dfr$Species == "setosa",]
write.table(dfr1,"iris-setosa.txt",sep="\t",row.names=F,quote=F)
```

`sep="\t"` sets tab delimiter, `row.names=F` avoids printing rownames, `quote=F` avoids quotes around strings.

Input & Output • Image

- Create data

```
dfr <- data.frame(a=sample(1:100,10),b=sample(1:100,10))
```

- Base plot

```
png(filename="plot-base.png")  
plot(dfr$a,dfr$b)  
dev.off()
```

- ggplot method 1

```
p <- ggplot(dfr,aes(a,b)) + geom_point()  
  
png(filename="plot-ggplot-1.png")  
print(p)  
dev.off()
```

- ggplot method 2

```
ggsave(filename="plot-ggplot-2.png",plot=p)
```

R objects

- Save R objects as compressed native R formats
- Save/Read a single object as .Rds format

```
dfr <- data.frame(a=sample(1:100,10),b=sample(1:100,10))  
saveRDS(dfr,"data.Rds")  
dfr <- readRDS("data.Rds")
```

- Save one or more objects as .Rda/.Rdata format

```
save(dfr,"data.Rdata")  
save(dfr,dfr2,"data.Rdata")  
load("data.Rdata")
```

- Save entire workspace

```
save.image(file="workspace.Rdata")  
load("workspace.Rdata")
```

Rmarkdown

TEXT. CODE. OUTPUT.
(GET IT TOGETHER, PEOPLE.)



Reproducible Analyses

- Manually steps = poor reproducibility
- Rerunning analyses
- Adding new data
- Transferring projects
- Collaborative work
- Eliminate copy-paste errors

Recommendations

- Single document with analyses, code and results
- Self-contained portable project
- Avoid manual steps
- Results are linked to code used to produce them
- Contextual narrative to workflow
- Version control of documents

Reproducibility in R

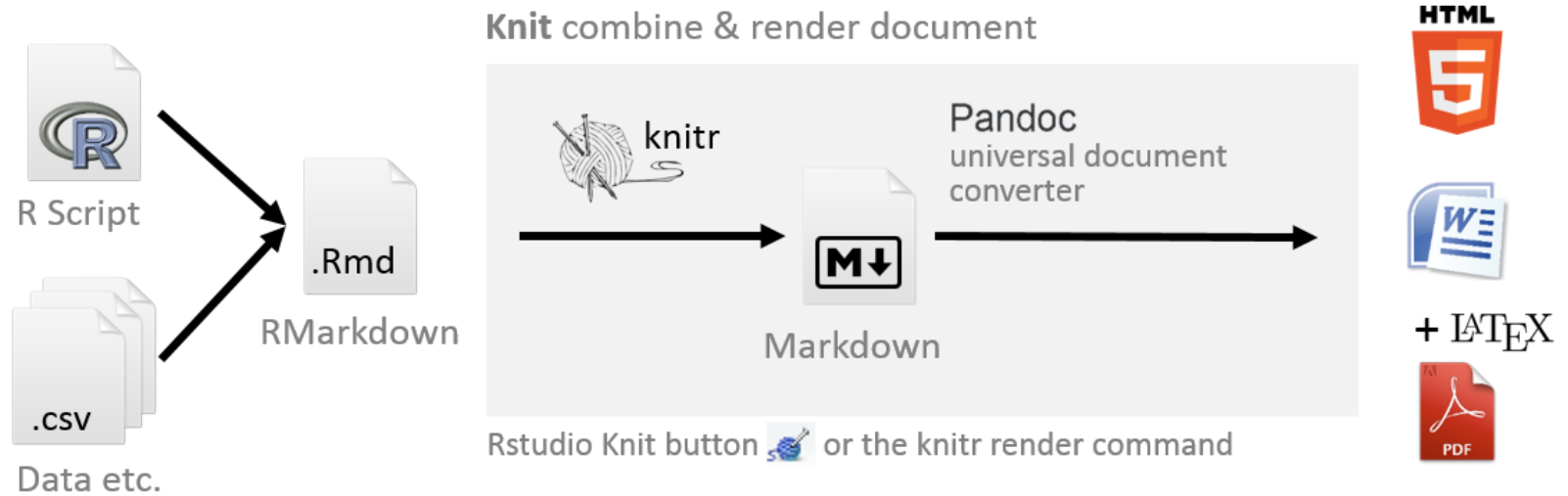
- Manage R versions carefully
- Install packages from repositories

```
install.packages(); devtools::install_github(); BiocManager::install()
```

- Package management using `renv`
- Version control using git (RStudio integrated)
- RStudio (Syntax highlighting, Debugging, Projects etc)
- Running external programs from R

```
system("./plink --file --flag1 --flag2 --out bla")
```

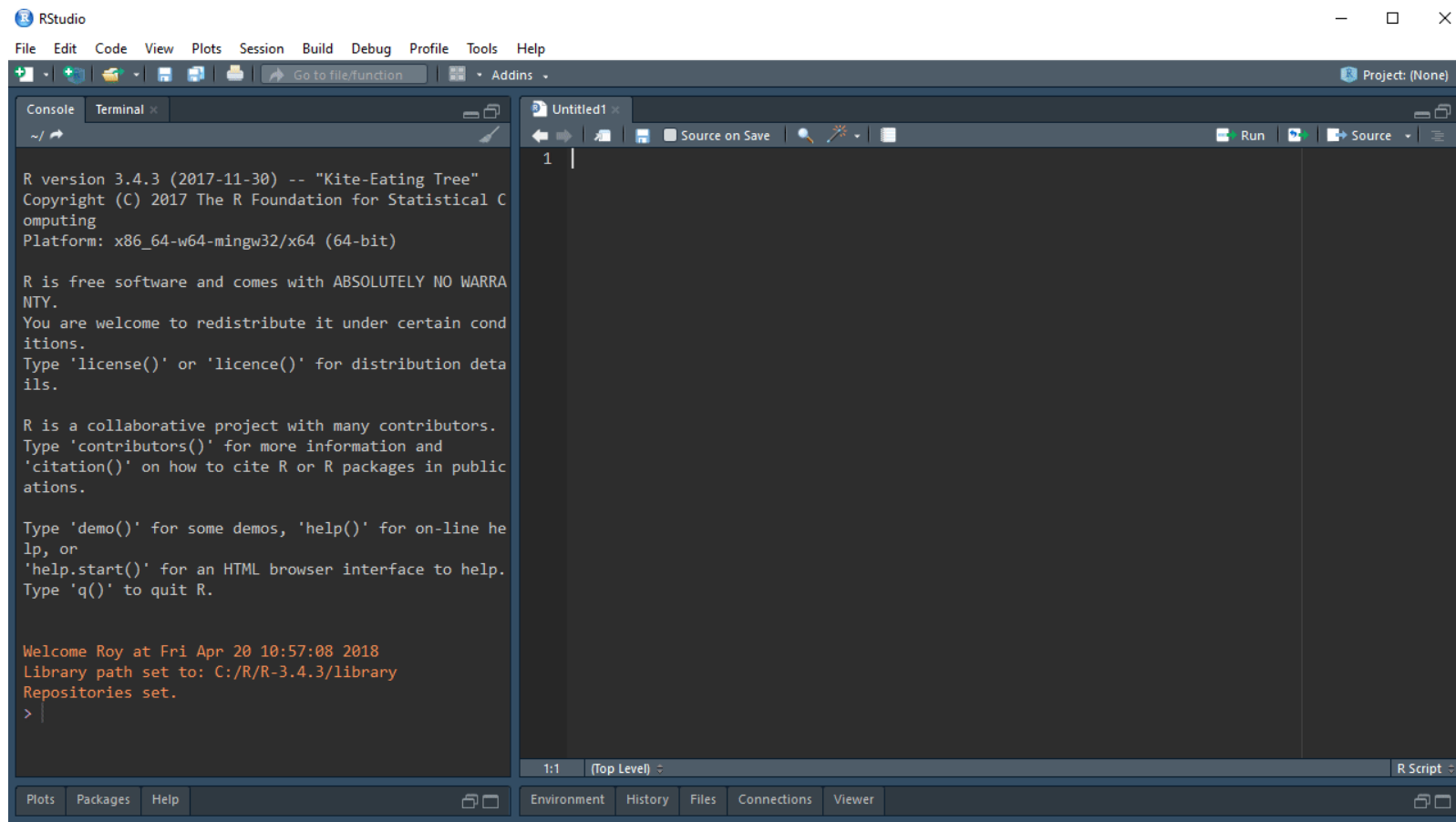
Document Converter



- Document converter (Reports, Presentations, Articles etc)
- Rmd -> md -> HTML|PDF|docx

RStudio Notebook

Create a new .Rmd document




- Text and code can be written together
- Inline R output (text and figures)

Rmarkdown Guide

- Plain text format for readability
- Support of pure language (HTML, Latex etc) for complex formatting
- Rmarkdown = Markdown + R chunks
- Create a file that ends in `.Rmd`
- Add YAML matter to top

```
---  
title: "This is a title"  
output:  
  rmarkdown::html_document  
---
```

- In RStudio `File > New File > R Markdown` opens up an Rmd template
- Render interactively using the **Knit** button 
- Render using command `rmarkdown::render("report.Rmd")`

Rmarkdown Guide

```
### Heading 3  
#### Heading 4
```

```
_italic text_  
__bold text__  
`code text`  
~~strikethrough~~  
2^10^  
2~10~
```

```
- bullet point
```

```
Link to [this](somewhere.com)
```

```

```

Heading 3

Heading 4

italic text

bold text

`code text`

~~strikethrough~~

2¹⁰

2₁₀

- bullet point

Link to [this](#)



Rmarkdown Guide

- R code can be executed inline

Today's date is ``r date()``

Today's date is Thu Sep 16 11:10:22 2021

- R code can be executed in code chunks

```
```${r}  
date()
```
```

- By default shows input code and output result.

```
date()
```

```
## [1] "Thu Sep 16 11:10:22 2021"
```

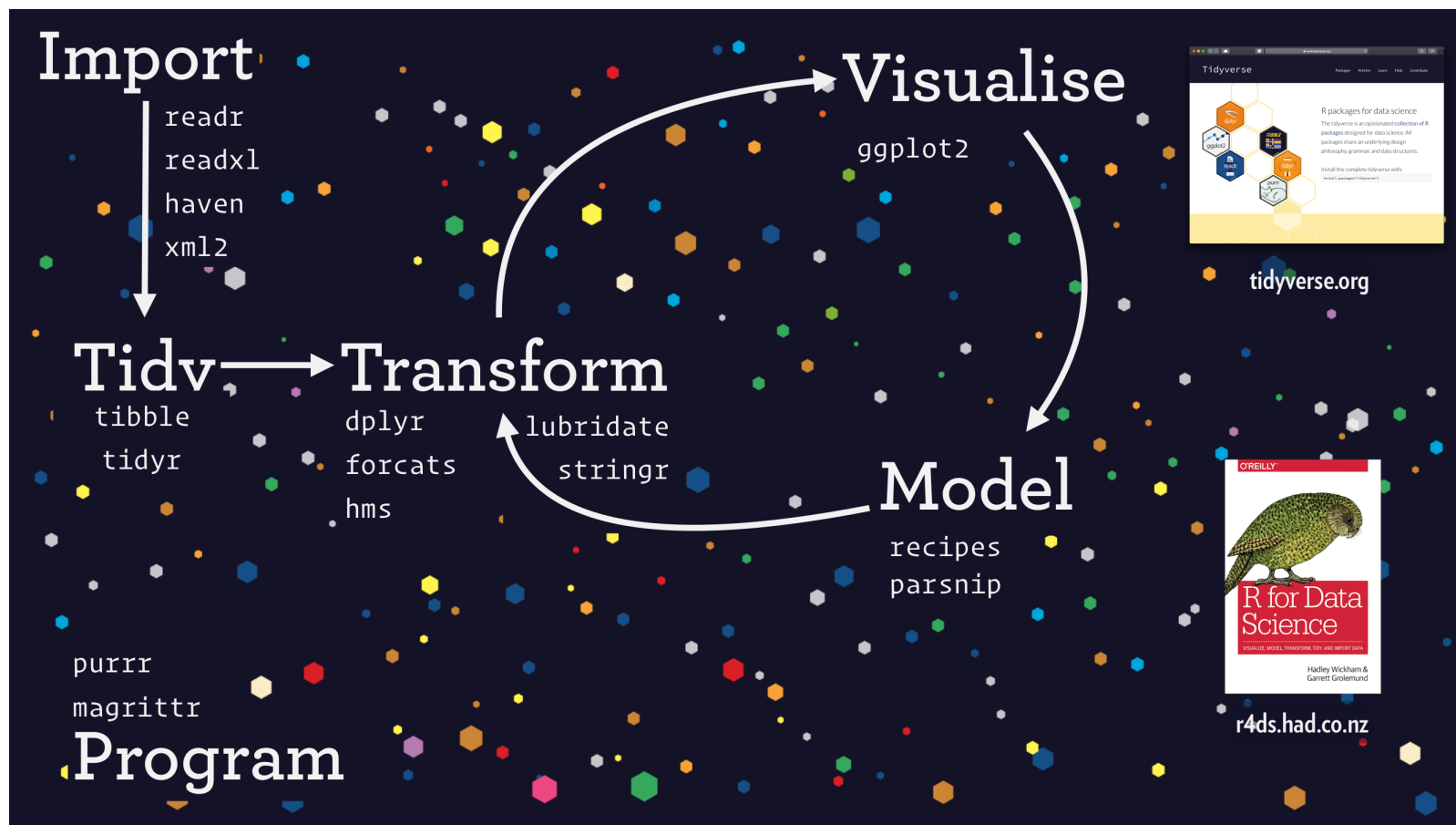
- Many arguments to **customise chunks**
 - Set `eval=FALSE` to not evaluate a code chunk
 - Set `echo=FALSE` to hide input code
 - Set `results="hide"` to hide output
- **R Markdown reference**



"Language for solving data science challenges using R"

- Collection of R packages that share underlying design and grammar
- Modern, consistent and optimised functions
- Additional features compared to base R
- New code structure using new operators (Eg: pipe `%>%`)
- Tidy data & tidy evaluation

Tidyverse



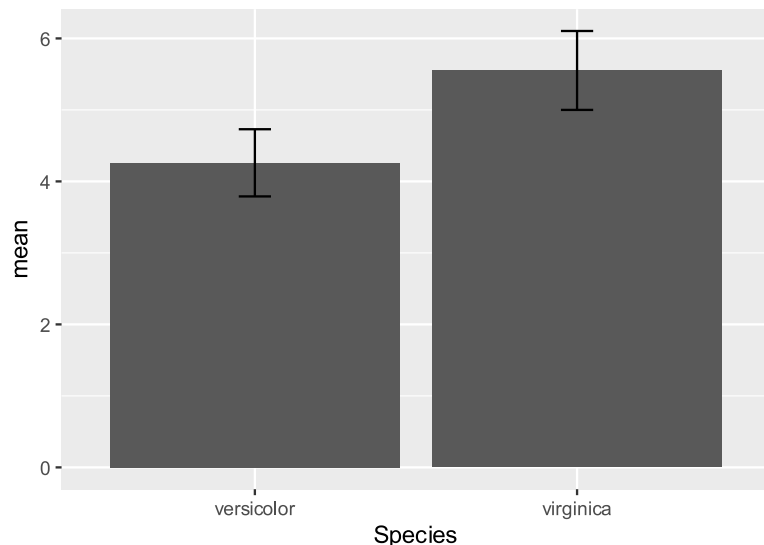
Tidyverse

- `magrittr` : Piping commands using `%>%`
- `tibble` : A better data.frame
- `readr` : Functions to import/export data
- `tidyr` : Data structuring: wide & long formats, splitting, fill missing values etc
- `dplyr` : Data selection, filtering, summarising, merging etc
- `lubridate` : Working with time
- `stringr` : Working with strings
- `forcats` : Working with factors
- `purrr` : Simpler control structures for programming
- `broom` : Model building
- `ggplot2` : Plotting

Tidyverse • Examples

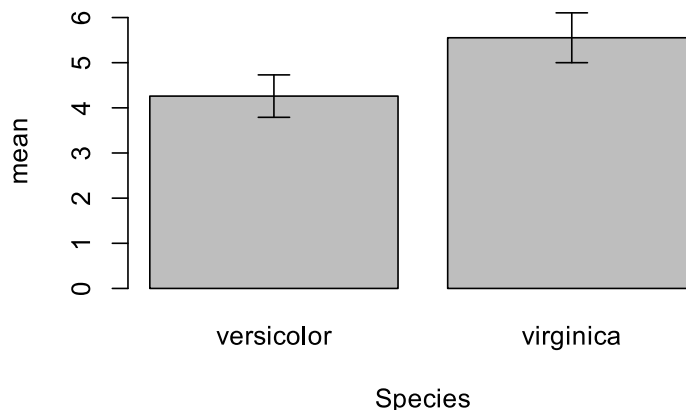
Tidyverse

```
iris %>%  
  filter(Species!="setosa") %>%  
  select(Species,Petal.Length) %>%  
  group_by(Species) %>%  
  summarise(mean=mean(Petal.Length),sd=sd(P  
  mutate(ymin=mean-sd,ymax=mean+sd) %>%  
  ggplot(aes(x=Species,y=mean,ymin=ymin,yma  
  geom_bar(stat="identity")+  
  geom_errorbar(width=0.1)
```



Base R

```
iris_mean <- aggregate(Petal.Length~Species  
iris_sd <- aggregate(Petal.Length~Species,d  
iris1 <- merge(iris_mean,iris_sd,by="Specie  
colnames(iris1) <- c("Species","mean","sd")  
iris1$Species <- factor(iris1$Species)  
iris1$ymin <- iris1$mean-iris1$sd  
iris1$ymax <- iris1$mean+iris1$sd  
{b <- barplot(iris1$mean,names.arg=iris1$Sp  
arrows(x0=b,y0=iris1$ymin,y1=iris1$ymax, le
```



Tidyverse • Examples

Tidyverse

```
# extract columns from a data.frame
select(iris, Species, Petal.Width)
select(iris, 5, 4)

# extract rows
filter(iris, Petal.Width > 0.5 & Species == "setosa")

# ordering a data.frame
arrange(iris, desc(Species), Petal.Width)

# add new computed variable
iris %>% mutate(cent=Petal.Length-mean(Petal.Length))

# grouped summarisation
iris %>% group_by(Species) %>% summarise(mean_petal=mean(Petal.Length))
```

Base R

```
# extract columns from a data.frame
iris[, c("Species", "Petal.Width")]
iris[, c(5, 4)]

# extract rows
iris[iris$Petal.Width > 0.5 & iris$Species == "setosa", ]

# ordering a data.frame
iris[order(rev(iris$Species), iris$Petal.Width), ]

# add new computed variable
iris$cent <- iris$Petal.Length-mean(iris$Petal.Length)

# grouped summarisation
aggregate(Petal.Length~Species,data=iris,FUN=mean)
```

Bioconductor



- NGS/Genomics/Biology related packages
- Package management using **BiocManager**
- Complex objects (Classes) to hold related objects
- Workflows for common tasks

Exercises

Hands-On practise material for the contents covered on this course is available here.

<https://royfrancis.github.io/course-r/lab.html>

Help

- Use `?function` to get function documentation
- Use `??name` to search for a function
- Use `args(function)` to get the arguments to a function
- Go to the package CRAN page/webpage for vignettes
- GitHub repos of packages have useful info
- **R bloggers**: Great blog to follow to keep updated with the latest in the R world as well as tutorials.
- **Stackoverflow**: Online community to find solutions to your problems.



Learning R

Tutorials

- [Introduction to R](#): Tutorial by Datacamp with excellent tutorials.
- [R programming tutorial](#): Youtube video tutorial by Derek Banas.
- [R for data science](#) Data science tutorial by Hadley wickham.
- [Data carpentry](#) Data carpentry R workshop (Medium-Advanced)

Reference

- [R Cookbook](#): General purpose reference.
- [Quick R](#): General purpose reference.
- [Awesome R](#): Curated list of useful R packages.
- [RStudio cheatsheets](#): Useful cheatsheets.
- [Advanced R](#) by Hadley Wickham (Medium-Advanced)

Links

- [Tutorialspoint List](#): Good list of resources.



Thank you! Questions?

Graphics from  freepik.com

Created: 16-Sep-2021 • Roy Francis • SciLifeLab • NBIS

