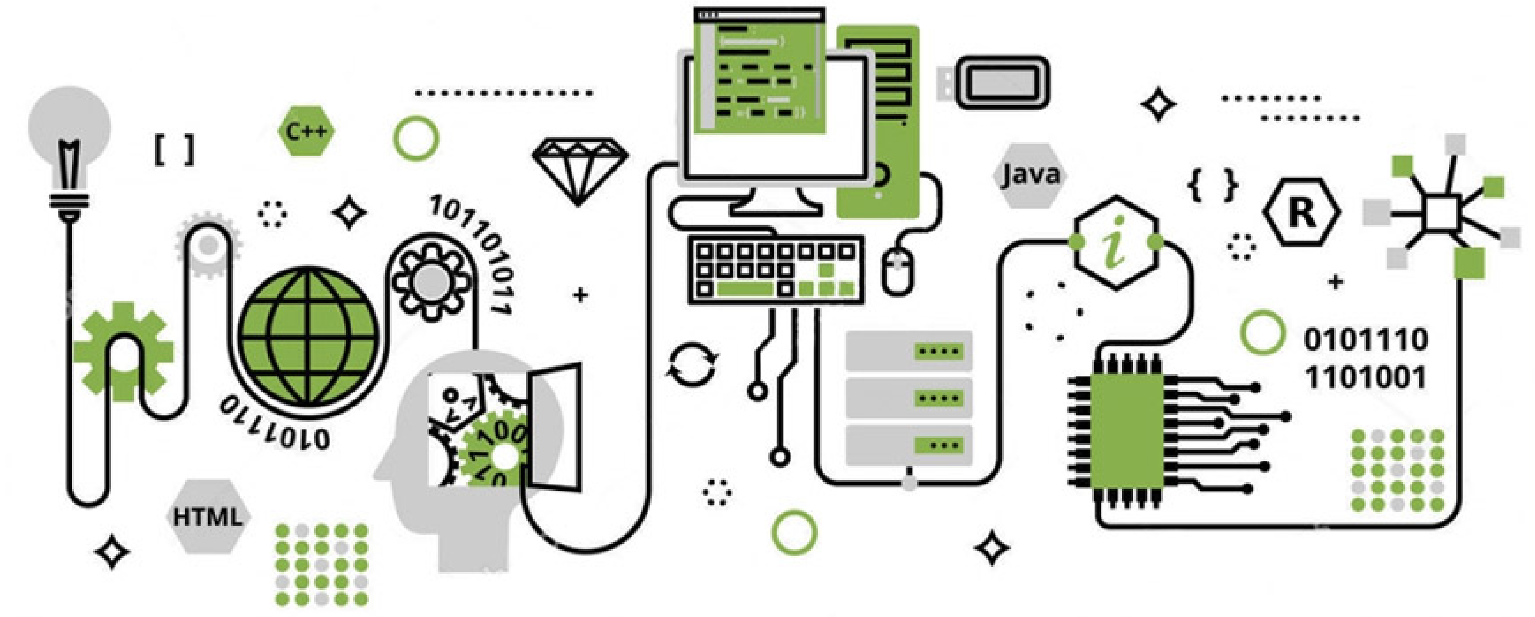


Introduction To R

Introduction to Bioinformatics

Roy Francis • 07-Nov-2019
NBIS • SciLifeLab

Contents

Topics

- Familiarise with R & RStudio environment
- Running code, scripting, sourcing script
- Variables and operators
- Data types & datatype conversion
- Creating and running 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

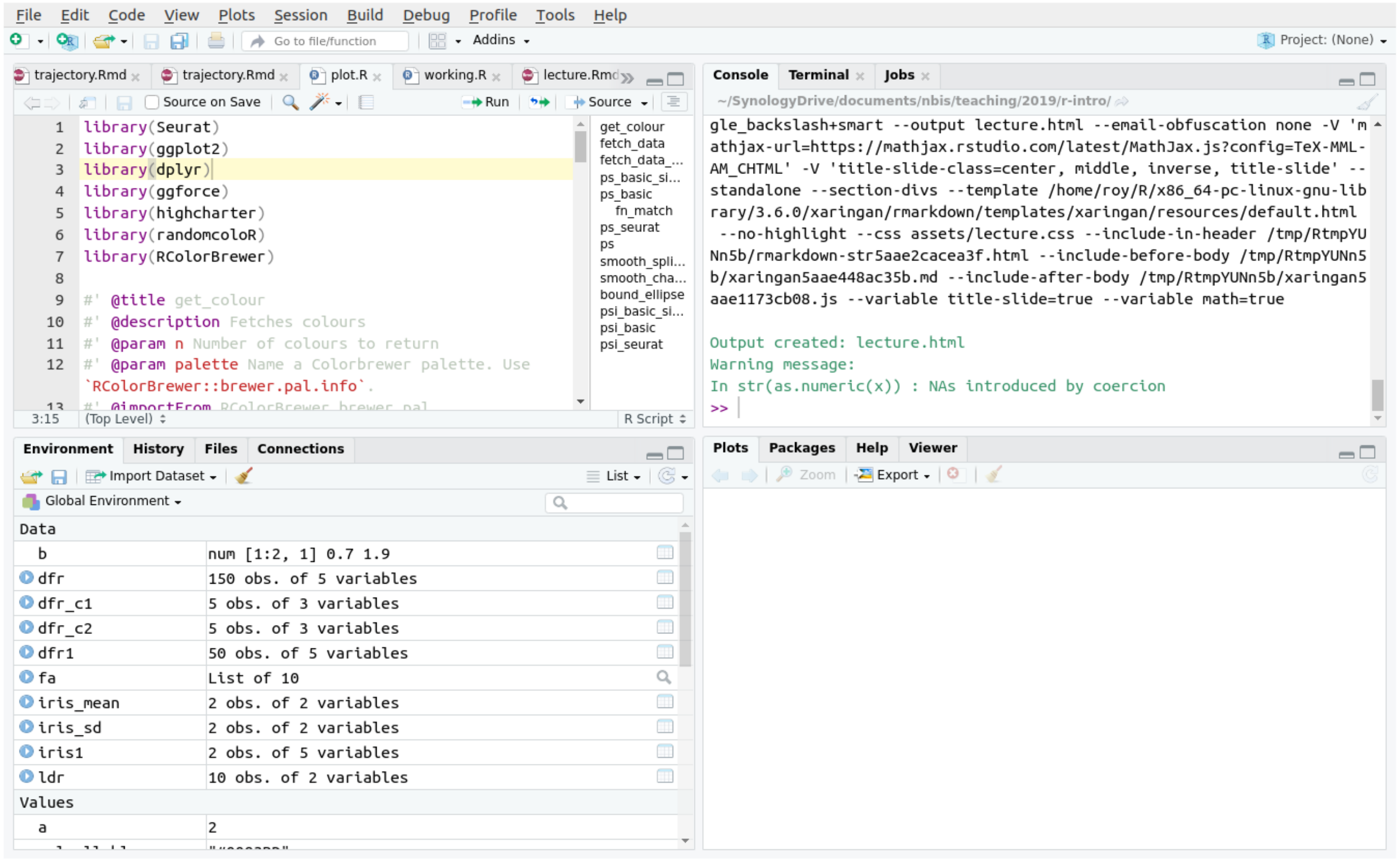
- Data analysis
- Statistics
- High quality graphics
- Huge number of packages
- R is popular
- Reproducible research
- RStudio IDE
- FREE! Open source

Cons

- Steep learning curve
- Not elegant/consistent
- Slow

Getting Started • Installation

- Install R from r-project.org.
- Install RStudio IDE
- Code editor, highlighting, projects, version control, package building, debugger, profiler



```r
library(Seurat)
library(ggplot2)
library(dplyr)
library(ggforce)
library(highcharter)
library(randomcoloR)
library(RColorBrewer)

#' @title get_colour
#' @description Fetches colours
#' @param n Number of colours to return
#' @param palette Name a Colorbrewer palette. Use
`RColorBrewer::brewer.pal.info`.
#' @importFrom RColorBrewer brewer.pal
```

```
gle_backslash+smart --output lecture.html --email-obfuscation none -V 'm
athjax-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 /home/roy/R/x86_64-pc-linux-gnu-lib
rary/3.6.0/xaringan/rmarkdown/templates/xaringan/resources/default.html
 --no-highlight --css assets/lecture.css --include-in-header /tmp/RtmpYU
Nn5b/rmarkdown-str5aae2cacea3f.html --include-before-body /tmp/RtmpYUNn5
b/xaringan5aae448ac35b.md --include-after-body /tmp/RtmpYUNn5b/xaringan5
aae1173cb08.js --variable title-slide=true --variable math=true

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

```
Data
  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
Values
  a          2
```

Getting Started • 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 variable names like `c`, `t` etc
- Variable names cannot start with a number

Data Type • Overview

- 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 `x` 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]
x[,2,drop=F]
```

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

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
```

- 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
```

```
str(dfr)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

- 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")
```

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
```

- Few other conversion functions

`as.matrix()`, `as.data.frame()`, `as.integer()`, `as.Date()`

Functions • Built-In • Math

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

```
##  [1]  80 130 196  42  83 105  37 168  89 116
```

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

```
## [1] 1046
## [1] 104.6
## [1] 97
## [1] 37
##  [1] 4.382027 4.867534 5.278115 3.737670 4.418841 4.653960 3.610918
##  [8] 5.123964 4.488636 4.753590
##  [1] 5.540622e+34 2.872650e+56 1.323483e+85 1.739275e+18 1.112864e+36
##  [6] 3.989520e+45 1.171914e+16 9.151093e+72 4.489613e+38 2.388691e+50
##  [1]  8.944272 11.401754 14.000000  6.480741  9.110434 10.246951  6.082763
##  [8] 12.961481  9.433981 10.770330
##  [1]  80 130 196  42  83 105  37 168  89 116
##  [1]  37  42  80  83  89 105 116 130 168 196
```

Functions • Built-In • String

```r
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

```
a <- 1:6; b <- 8:10

d <- a*b
e <- log(d)
f <- sqrt(e)
f
```

```
## [1] 1.442027 1.700109 1.844234 1.861649 1.951067 2.023449
```

- Custom function definition

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

- Re-use function

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

```
## [1] 1.576359 1.744856 1.861649
```

- Function names must not start with number

Control Structure • `if`

- Conditional statements using `if()`

```r
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

```r
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

```r
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
```

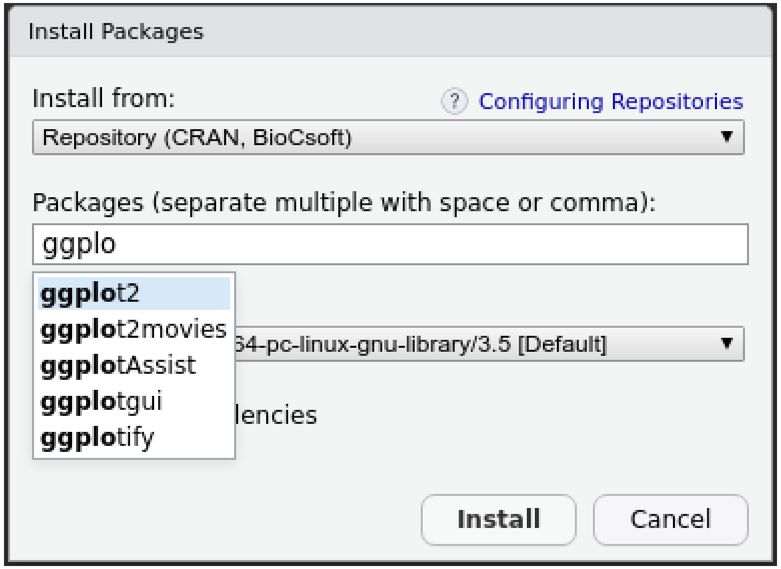
R Packages

- 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")
```


Install Packages

Install from:

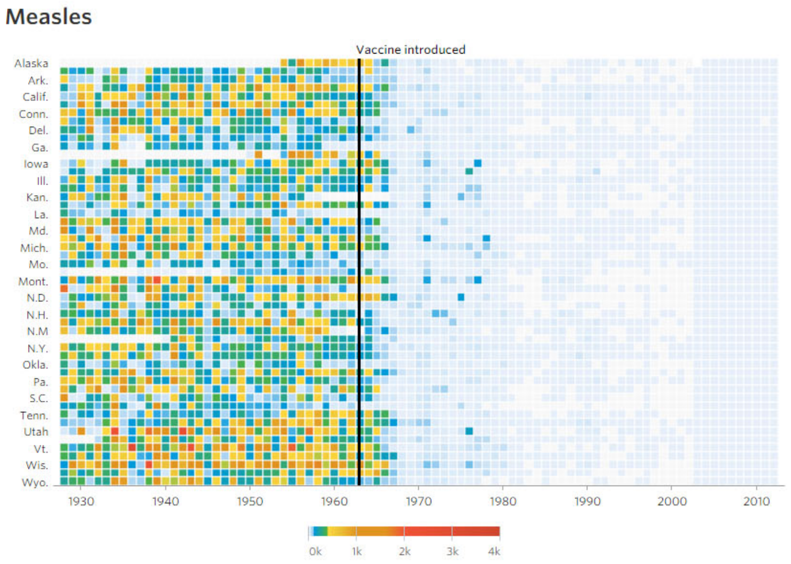
Repository (CRAN, BioCsoft)

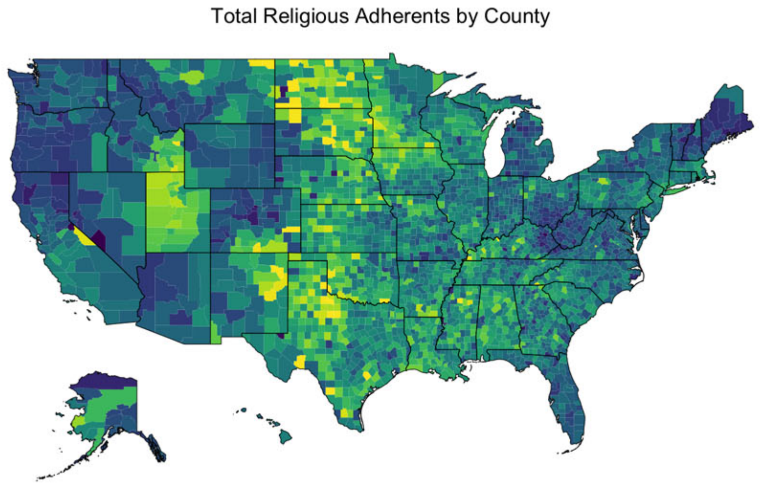
Packages (separate multiple with space or comma):

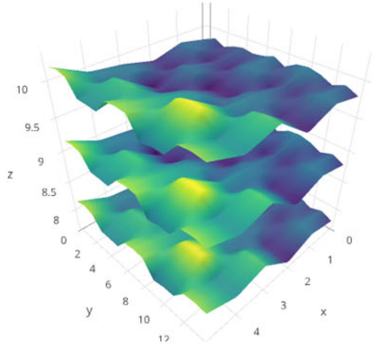
ggplo

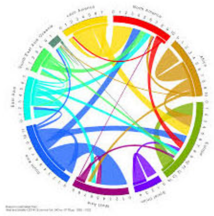
- Bioconductor for Biology/Bioinformatics packages; Use `BiocManager::install()`
- For GitHub packages, Use `devtools::install_github()`

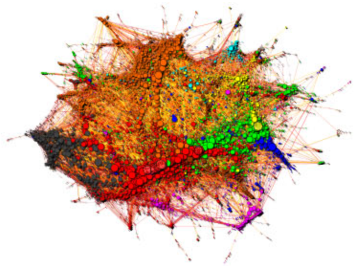
Graphics

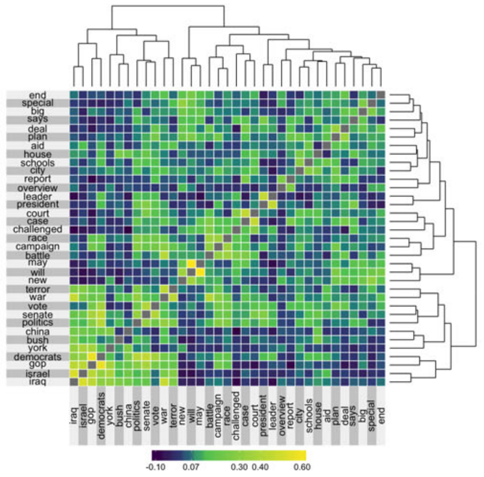

Measles
Vaccine introduced
Alaska
Ark.
Calif.
Conn.
Del.
Ga.
Iowa
Ill.
Kan.
La.
Md.
Mich.
Mo.
Mont.
N.D.
N.H.
N.M
N.Y.
Okla.
Pa.
S.C.
Tenn.
Utah
Vt.
Wis.
Wyo.
1930
1940
1950
1960
1970
1980
1990
2000
2010
0k
1k
2k
3k
4k


Total Religious Adherents by County



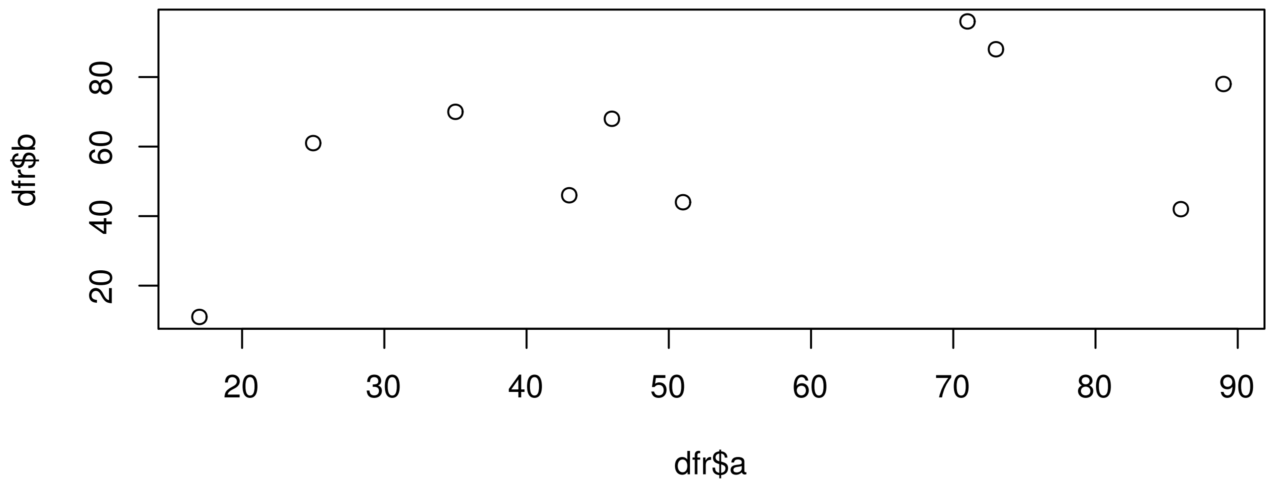






Graphics • Base

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

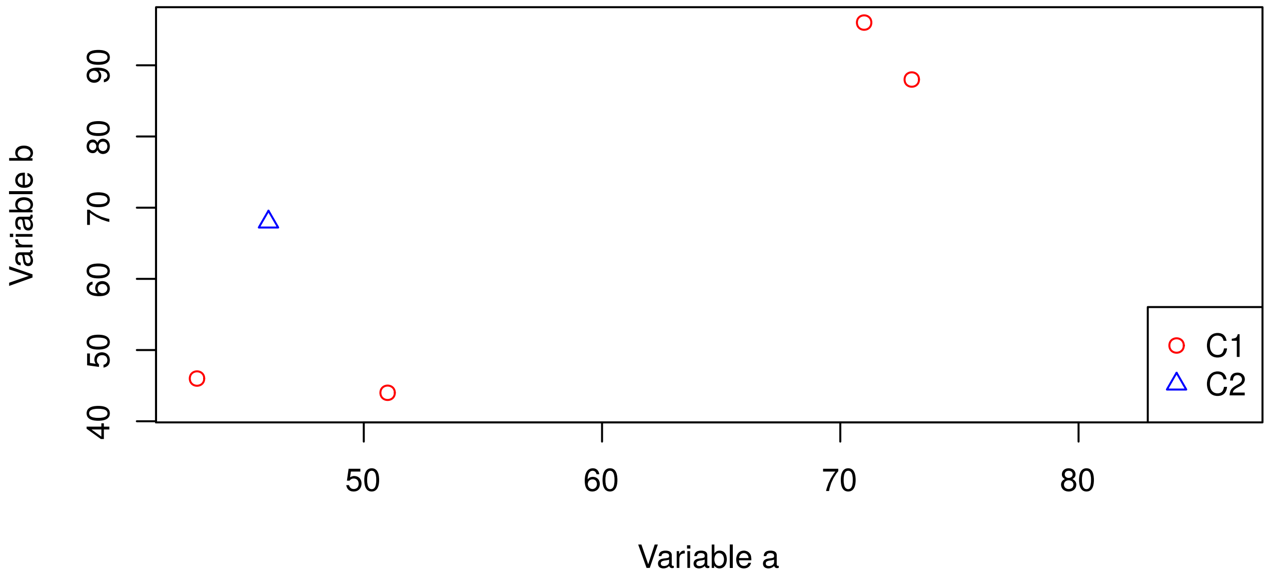


- Add axes labels etc

```r
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

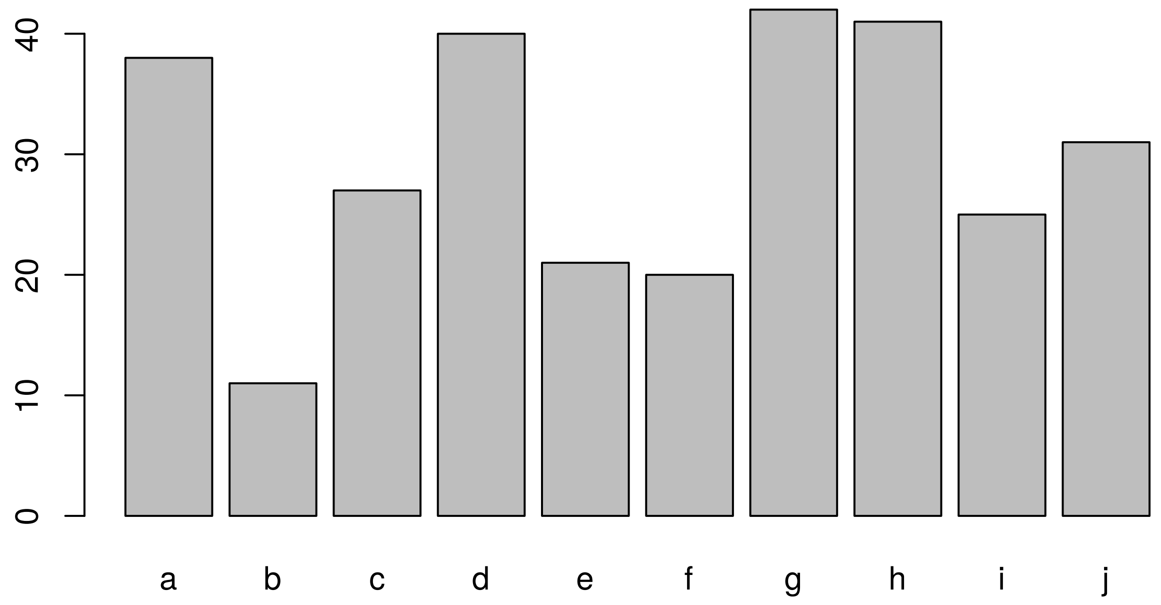
```r
dfr$cat <- rep(c("C1","C2"),each=5) # add category

# 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

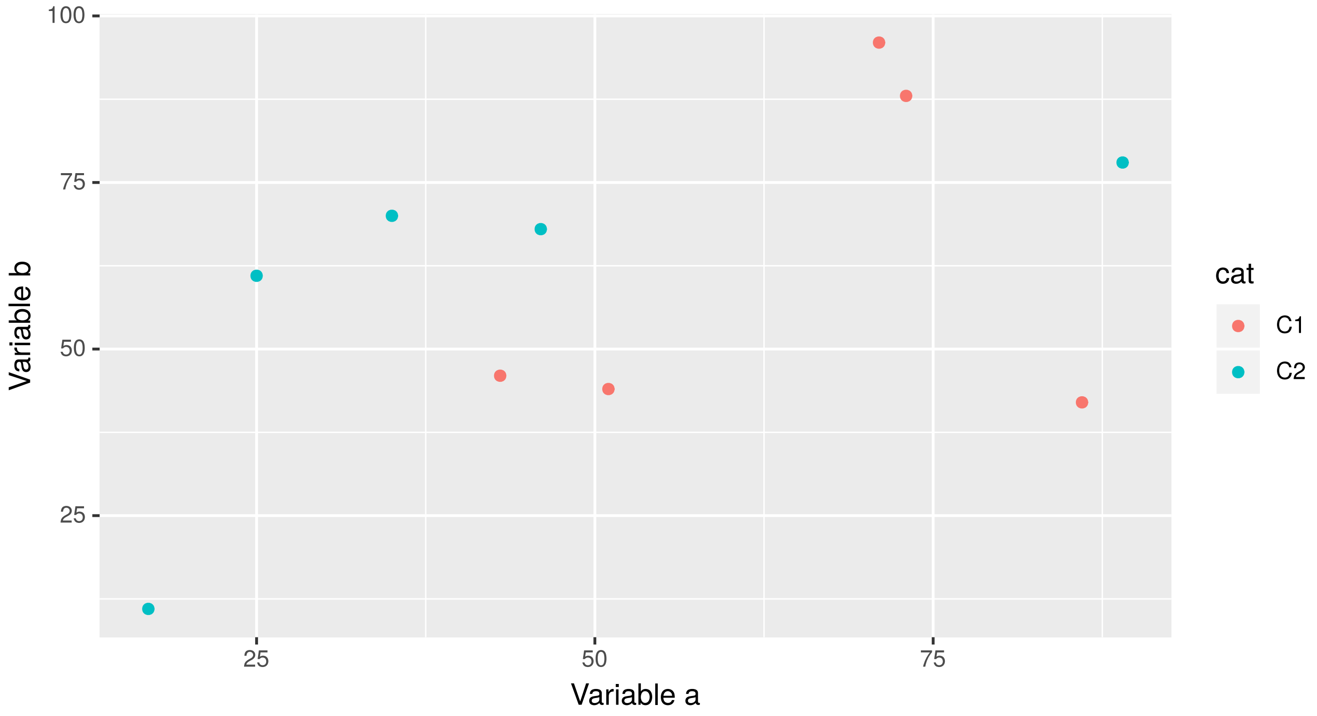
- Barplot

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



Graphics • `ggplot2`

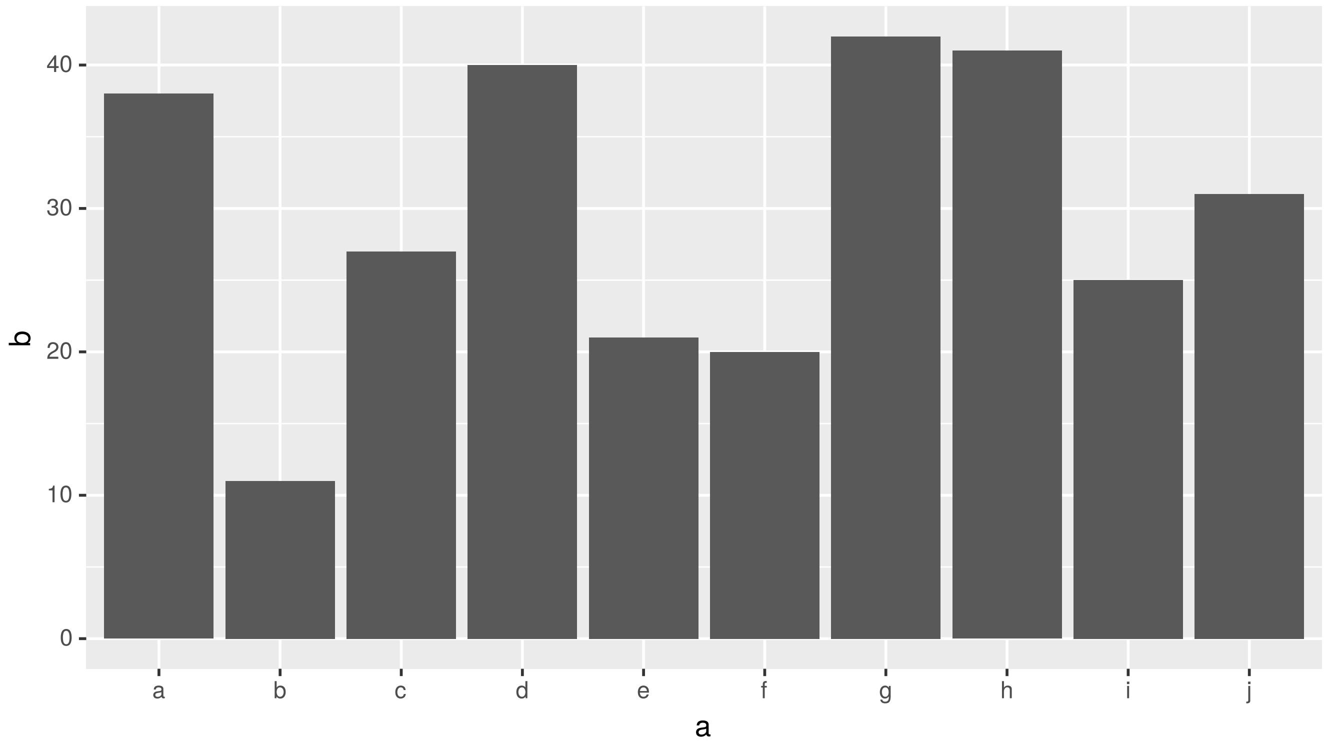
```r
library(ggplot2)

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)
```

```
## '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")
```

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
- Contexual narrative to workflow
- Version control of documents

Data Management Ecosystem

- Track edits and collaborate coding. Eg: *Git*
- Share and track code. Eg: *GitHub*
- Notebooks, documentation and reports. Eg: *Latex, Jupyter, Rmarkdown*
- Package and environment manager. Eg: *Packrat, Conda, Virtualenv*
- Workflow manager Eg: *Snakemake, Nextflow*
- Virtual machines. Eg: *VMWare, VirtualBox*
- Containerised computing environment. Eg: *Docker, Singularity, Vagrant*
- Big data analyses. Eg: *Hadoop, Spark*
- Workflow orchestration. Eg: *Openstack, Kubernetes, Terraform*

Reproducibility in R

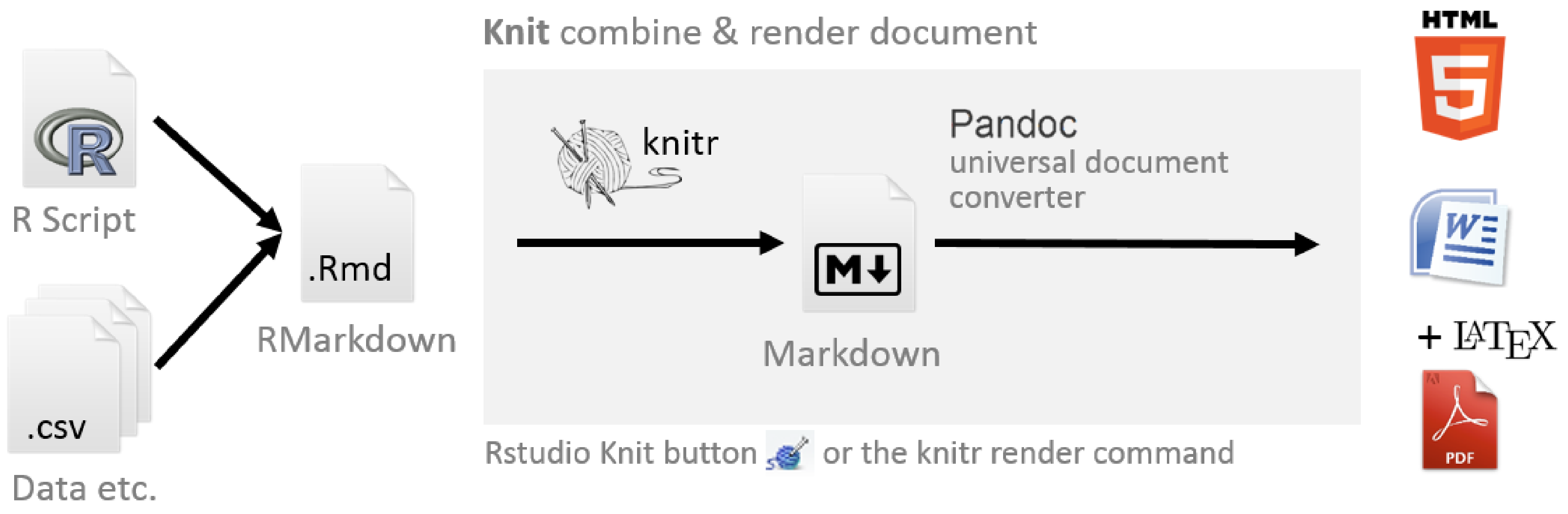
- Manage R versions carefully
- Install packages from repositories

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

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

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

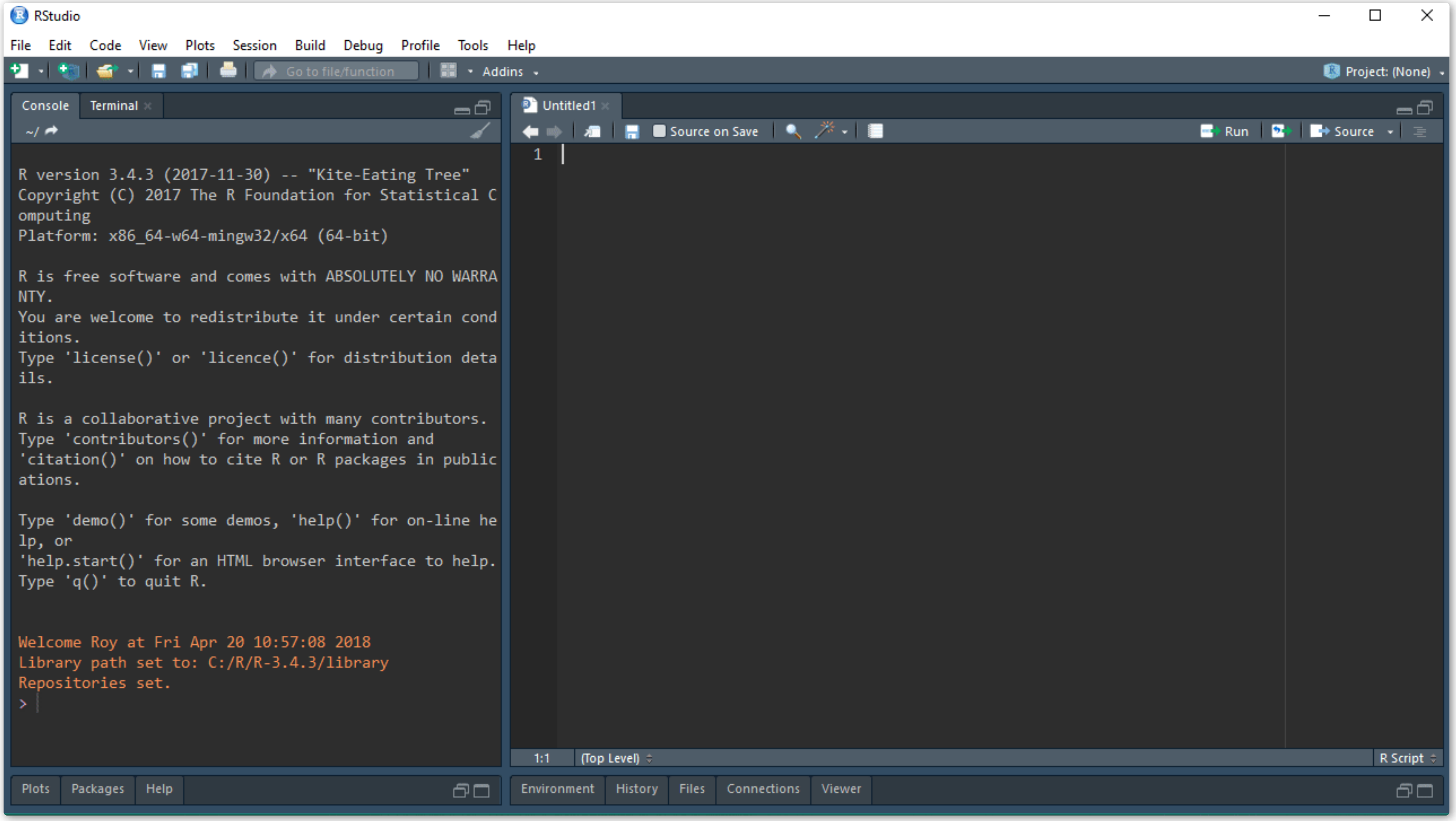
Document Converter


Knit combine & render document
R Script
.csv
Data etc.
.Rmd
RMarkdown
knitr
Markdown
Pandoc
universal document converter
HTML
+ LaTeX
PDF
Rstudio Knit button or the knitr render command

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

RStudio Notebook

Create a new .Rmd document



```
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"
Copyright (C) 2017 The R Foundation for Statistical C
omputing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRA
NTY.
You are welcome to redistribute it under certain cond
itions.
Type 'license()' or 'licence()' for distribution deta
ils.

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

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


Welcome Roy at Fri Apr 20 10:57:08 2018
Library path set to: C:/R/R-3.4.3/library
Repositories set.
>
```

- 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)

![](https://www.r-project.org/Rlogo.png)
```

Heading 3

Heading 4

italic text
bold text
`code text`
~~strikethrough~~
2^{10}
2_{10}

- bullet point

Link to this

Rmarkdown Guide

- R code can be executed inline

Today's date is `` `r date()` ``
Today's date is Thu Nov 7 20:46:11 2019

- R code can be executed in code chunks

~~~
```{r}
date()
```
~~~

- By default shows input code and output result.

```
date()
```

```
## [1] "Thu Nov  7 20:46:11 2019"
```

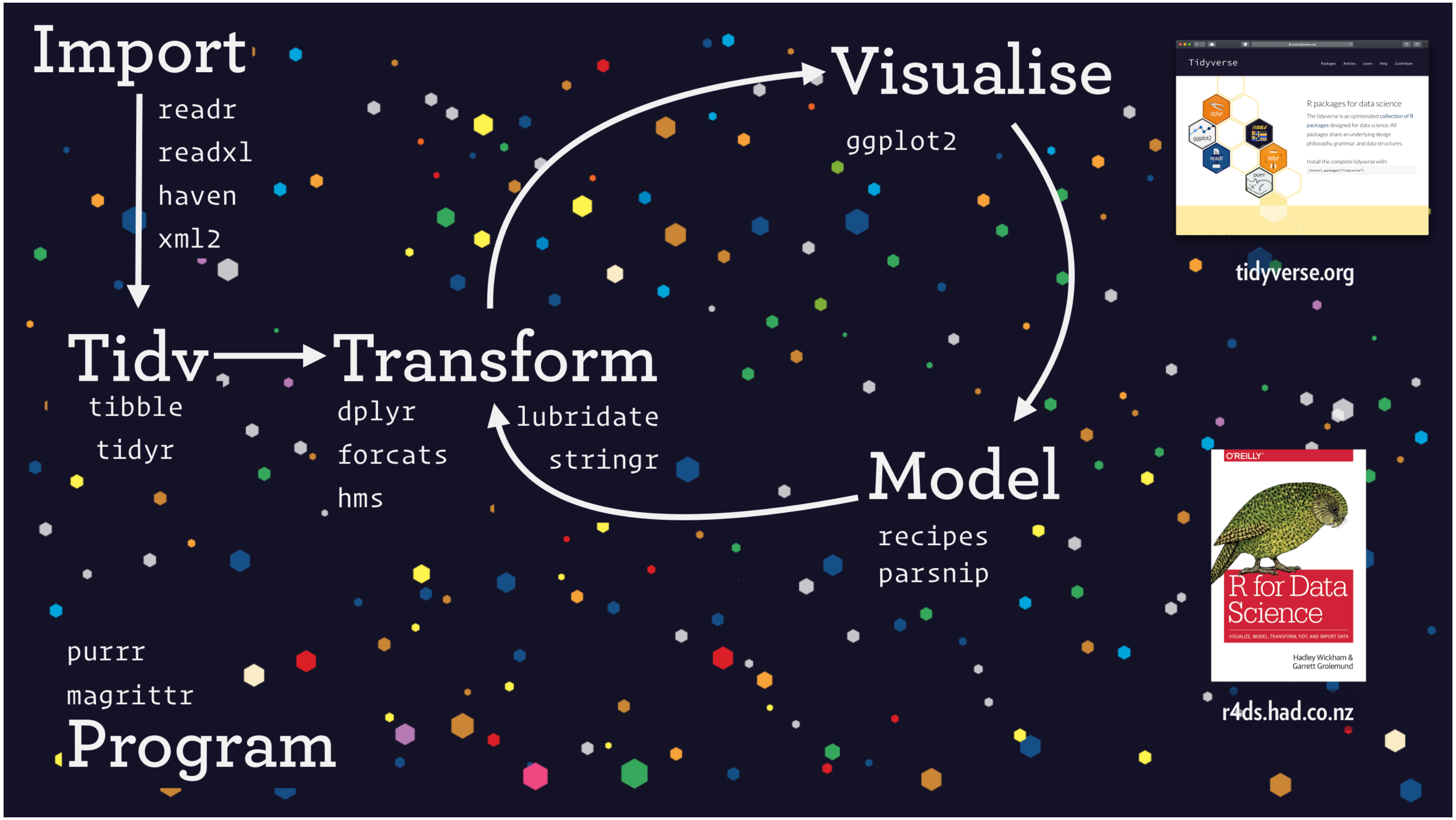
- Many arguments to customise chunks
    - Set `eval=FALSE` to not evaluate a code chunk
    - Set `echo=FALSE` to hide input code
    - Set `results="hide"` to hide output
- R Markdown reference

Tidyverse

> "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


Import
readr
readxl
haven
xml2
Tidy
tibble
tidyr
Transform
dplyr
forcats
hms
lubridate
stringr
Visualise
ggplot2
Model
recipes
parsnip
purrr
magrittr
Program
Tidyverse
R packages for data science
tidyverse.org
O'REILLY
R for Data Science
Hadley Wickham & Garrett Grolemund
r4ds.had.co.nz

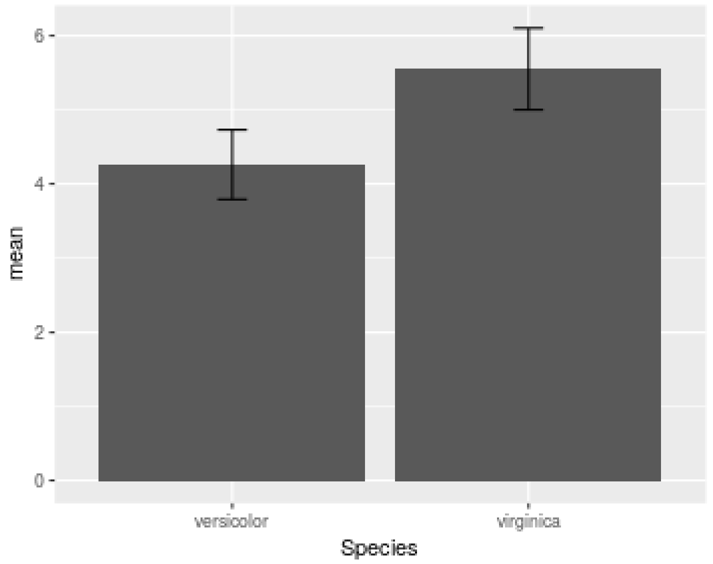
Tidyverse

- `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

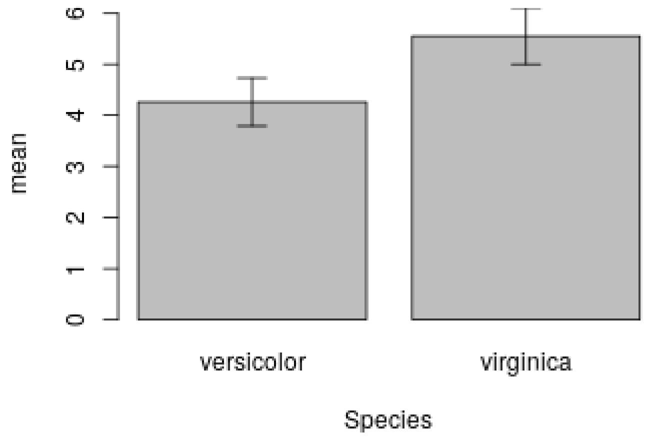
Tidyverse

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



Base R

```
iris_mean <- aggregate(Petal.Length~Species,data=
iris_sd <- aggregate(Petal.Length~Species,data=su
iris1 <- merge(iris_mean,iris_sd,by="Species")
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$mean,names.arg=iris1$Species,
arrows(x0=b,y0=iris1$ymin,y1=iris1$ymax, length=0
```



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 == "setc

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

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

# grouped summarisation
iris %>% group_by(Species) %>% summarise(mean=mea
```

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 == "se

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

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

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

Bioconductor

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

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
- 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

There are lots of resources for getting help in 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

Built on 07-Nov-2019. Graphics from freepik.com

2019 • Roy Francis • SciLifeLab • NBIS