YIMING LI, 15 MAR 2017

THE BEGINNER'S GUIDE TO R

WARNING: COMPLETELY FOR BEGINNERS!

IN TODAY'S GUIDE...

1. What is R? Why R?
2. Installation and "Hello World!" in R
3. R data types – vectors, matrices and data frames
4. R operators and managing a data frame
5. I/O and basic graphs in R
6. Pop quiz

* A set of materials used in this workshop could be download via: http://web.hku.hk/~liym1018/projects.html#RWorkshop

WARNING: COMPLETELY FOR BEGINNERS!

IN TODAY'S GUIDE...

1. What is R? Why R?
2. Installation and "Hello World!" in R
3. R data types – vectors, matrices and data frames
4. R operators and managing a data frame
5. I/O and basic graphs in R
6. Pop quiz

What is  ?

(From Wikipedia)

- **R** is an **open source programming language** and **software environment** for **statistical computing** and **graphics**.
- The **R** language is widely used among **statisticians** and **data miners** for **developing statistical software** and **data analysis**.

What is R?

(From Wikipedia)

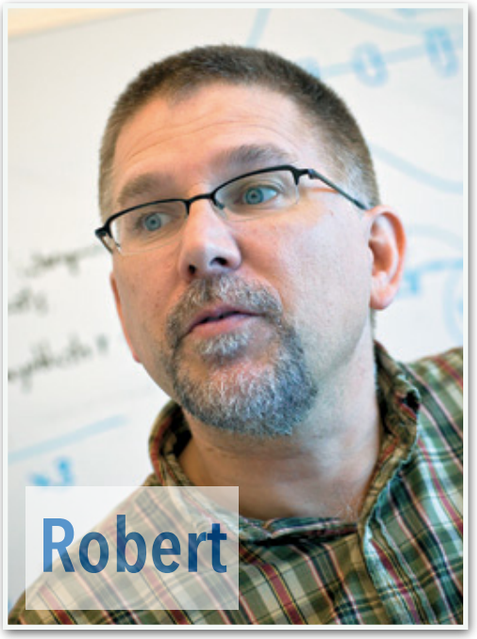
- **R** is an **open source programming language** and **software environment** for **statistical computing** and **graphics**.
- The **R** language is widely used among **statisticians** and **data miners** for **developing statistical software** and **data analysis**.
- **R** was created by **R**oss Ihaka and **R**obert Gentleman as an implementation of the S programming language (initial version: 1995; stable beta version: 2000).

What is R?

(From Wikipedia)

- **R** is an **open source programming language** and **software environment** for **statistical computing** and **graphics**.
- The **R** language is widely used among **statisticians** and **data miners** for **developing statistical software** and **data analysis**.
- **R** was created by **R**oss Ihaka and **R**obert Gentleman as an implementation of the S programming language (initial version: 1995; stable beta version: 2000).

What is R?



(From Wikipedia)

- **R** is an **open source programming language** and **software environment** for **statistical computing** and **graphics**.
- The **R** language is widely used among **statisticians** and **data miners** for **developing statistical software** and **data analysis**.
- **R** was created by **R**oss Ihaka and **R**obert Gentleman as an implementation of the S programming language (initial version: 1995; stable beta version: 2000).

Why R?

- **R** is **free**.
  - R is available as **Free Software** under the terms of the Free Software Foundation's GNU General Public License in source code form.

Why R?

- **R** is **free**.
  - R is available as **Free Software** under the terms of the Free Software Foundation's GNU General Public License in source code form.

Why R?

- **R** is **free**.
  - R is available as **Free Software** under the terms of the Free Software Foundation's GNU General Public License in source code form.
  - ***Free as in "free speech", not "free beer"!***
  - The users have the **freedom** to **run**, **copy**, **distribute**, **study**, **change** and **improve** the software.

Why R?

- **R** is **free**.
  - R is available as **Free Software** under the terms of the Free Software Foundation's GNU General Public License in source code form.
- ***Free as in "free speech", not "free beer"!***
- The users have the **freedom** to **run**, **copy**, **distribute**, **study**, **change** and **improve** the software.

Why R?

- **R** is **statistical**.
- Use R for **data analysis**.

Why R?

- **R** is **statistical**.
- Use R for **data analysis**.
  - Multiple linear regression

```
fit <- lm(y ~ x1 + x2 + x3, data=mydata)
```

Why R?

- **R** is **statistical**.
- Use R for **data analysis**.
  - Multiple linear regression

```
fit <- lm(y ~ x1 + x2 + x3, data=mydata)
```

  - One-way ANOVA

```
fit <- aov(y ~ A, data=mydataframe)
```

Why R?

- **R** is **statistical**.
- Use R for **data analysis**.
  - Multiple linear regression

    ```
    fit <- lm(y ~ x1 + x2 + x3, data=mydata)
    ```

  - One-way ANOVA

    ```
    fit <- aov(y ~ A, data=mydataframe)
    ```

  - Structural equation modelling

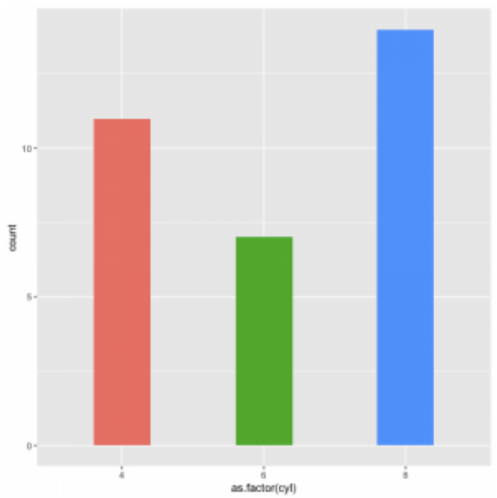
    Many packages available — `sem`, `lavaan`, `OpenMX`

Why R?

- **R** is **statistical**.
- Use R for **data visualisation**.

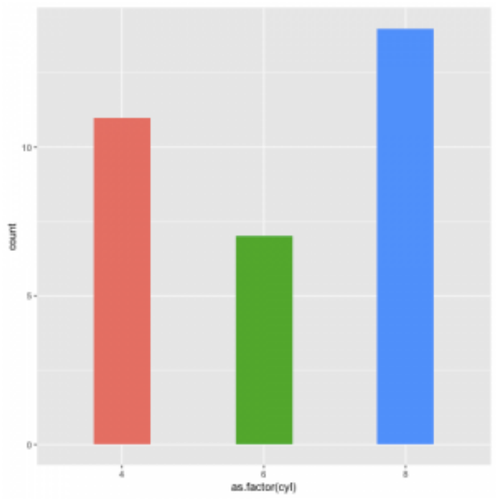
Why R?

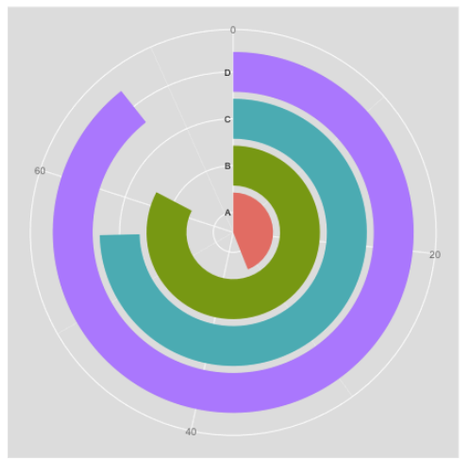
- **R** is **statistical**.
- Use R for **data visualisation**.



Why R?

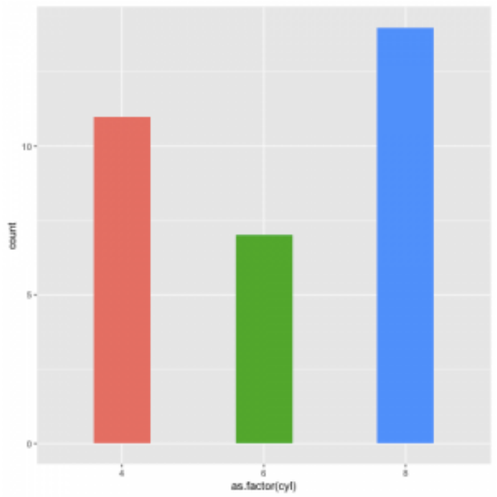
- **R** is **statistical**.
- Use R for **data visualisation**.

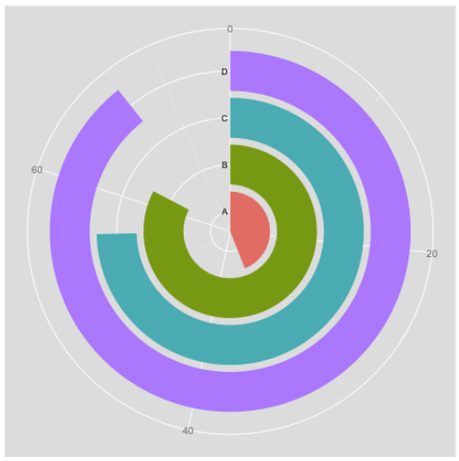


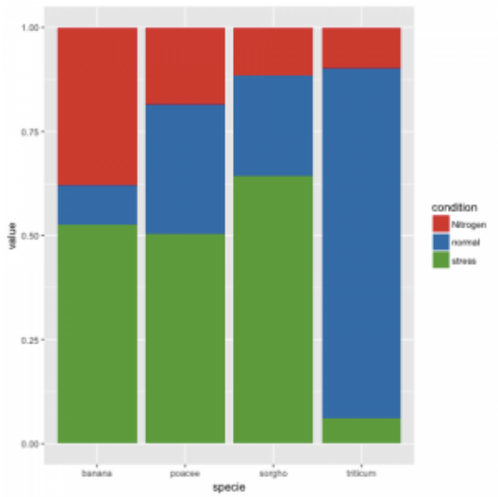


Why R?

- **R** is **statistical**.
- Use R for **data visualisation**.

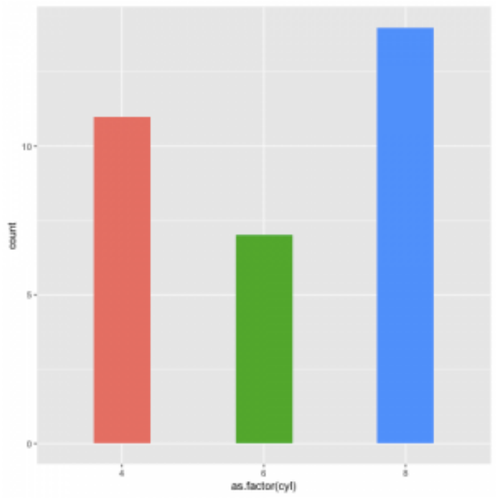


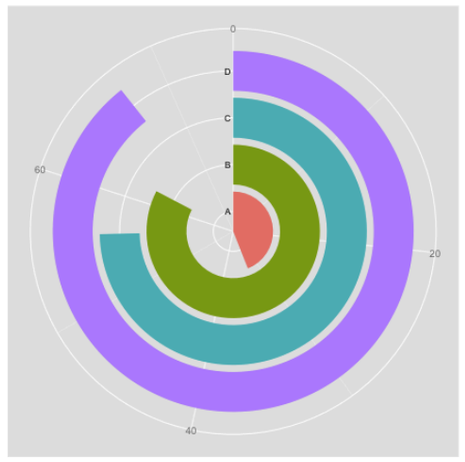


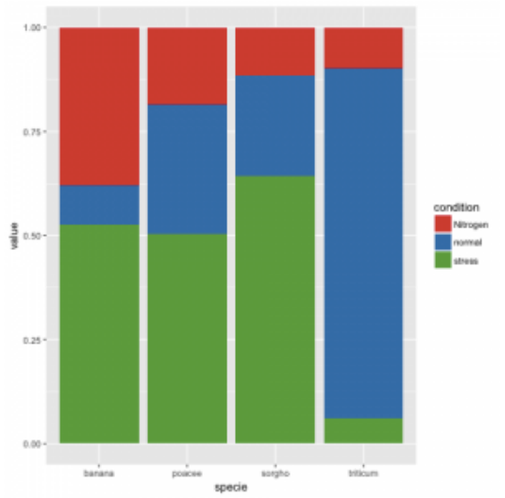


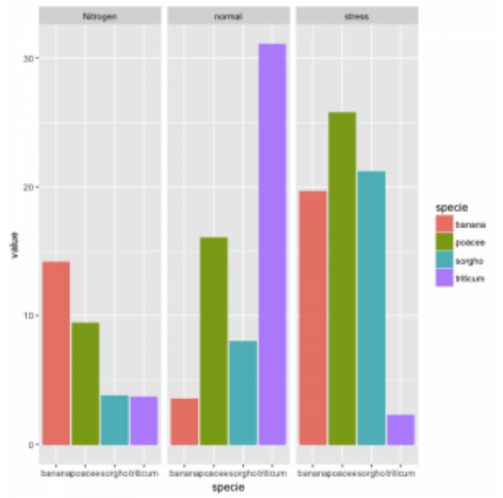
Why R?

- **R** is **statistical**.
- Use R for **data visualisation**.



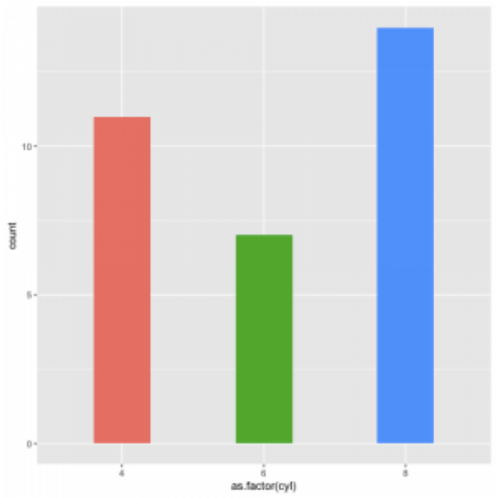


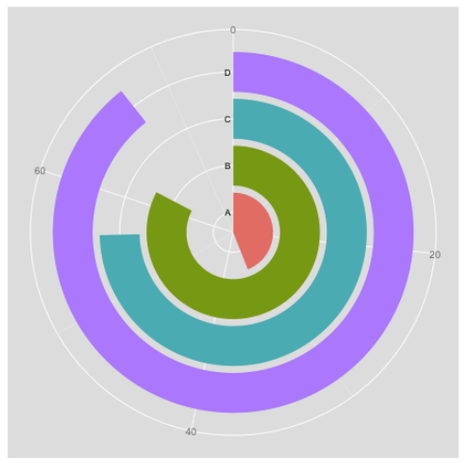


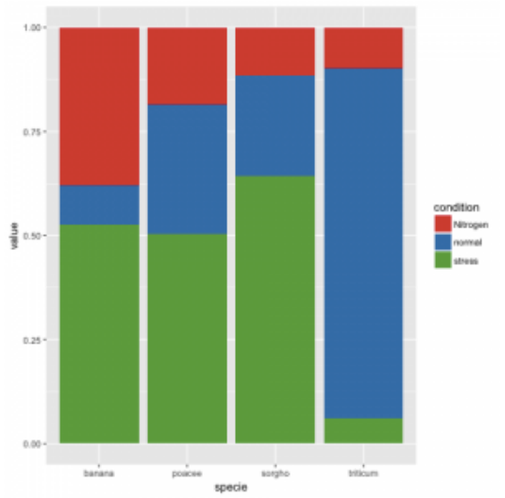


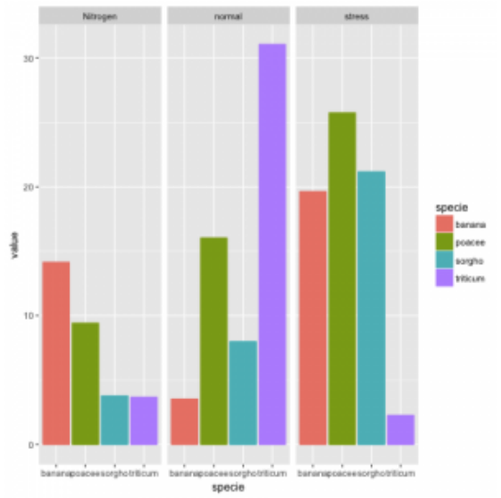
Why R?

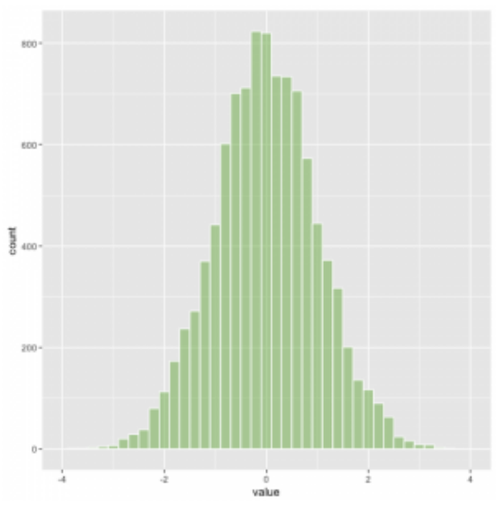
- **R** is **statistical**.
- Use R for **data visualisation**.





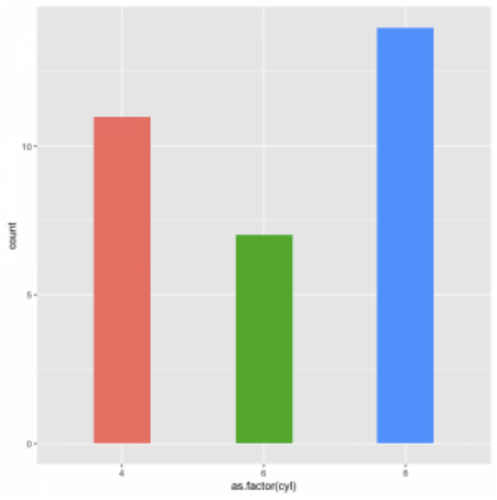


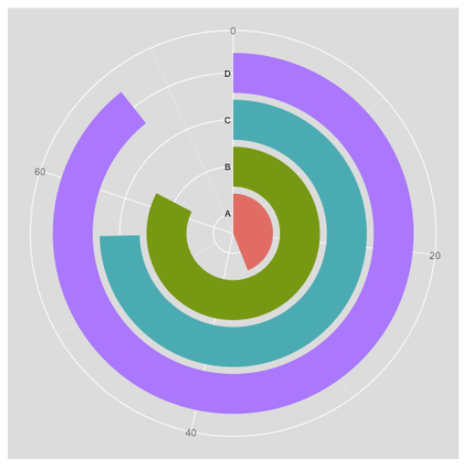


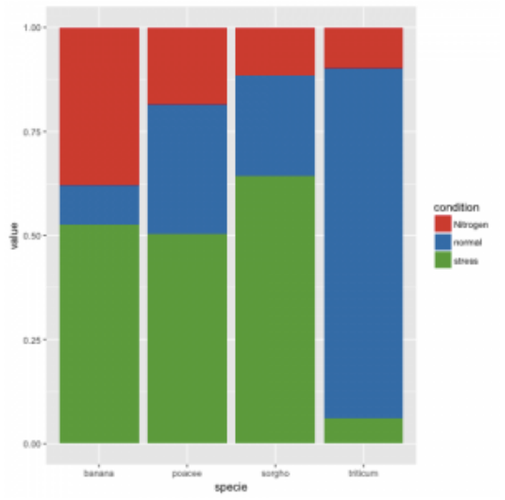


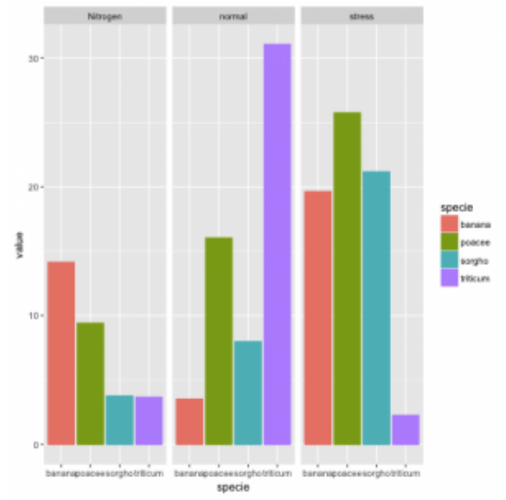
Why R?

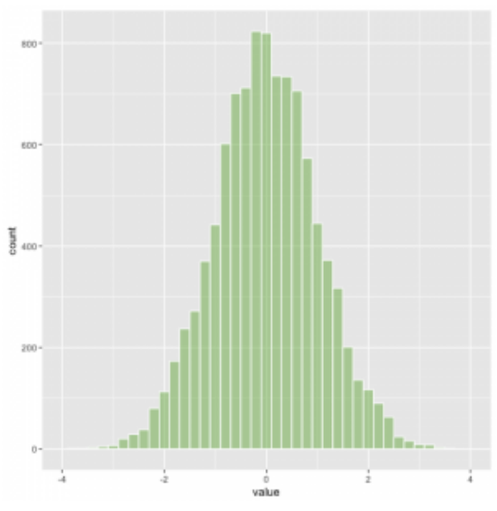
- **R** is **statistical**.
- Use R for **data visualisation**.

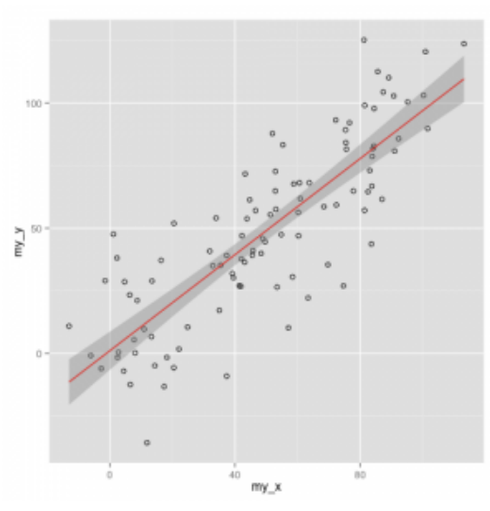






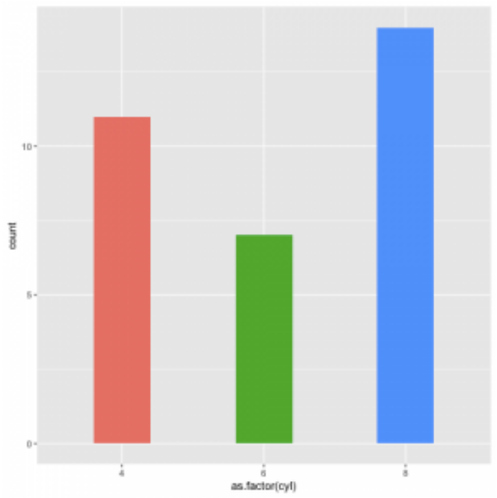


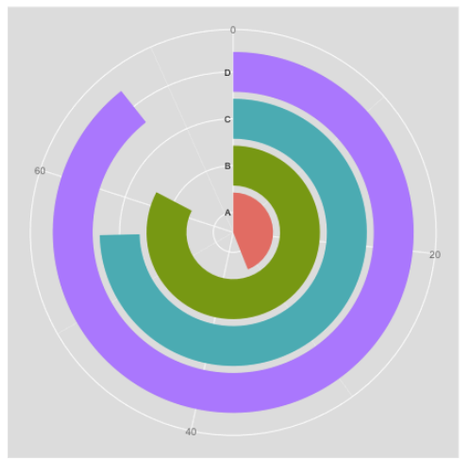


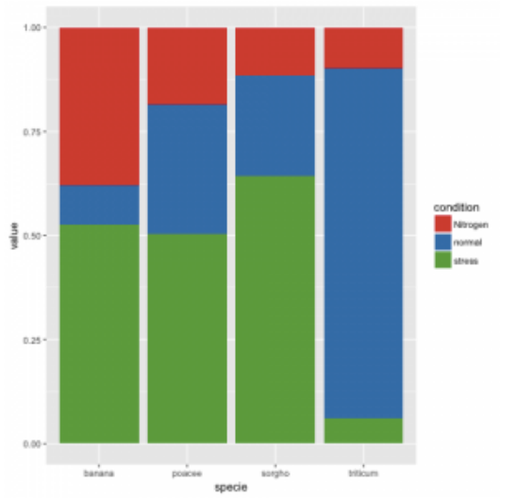


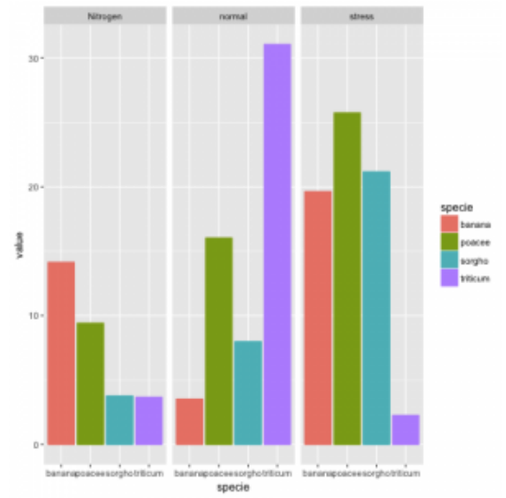
Why R?

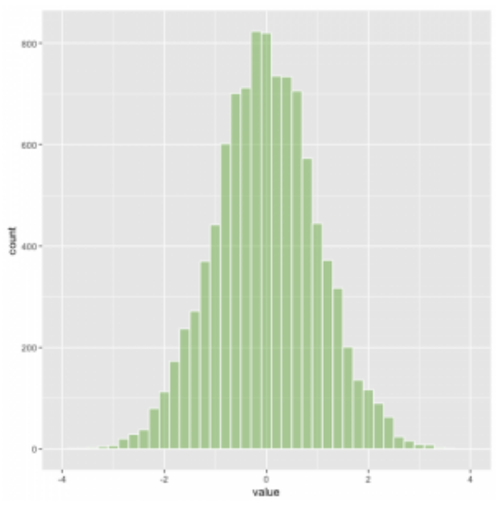
- **R** is **statistical**.
- Use R for **data visualisation**.

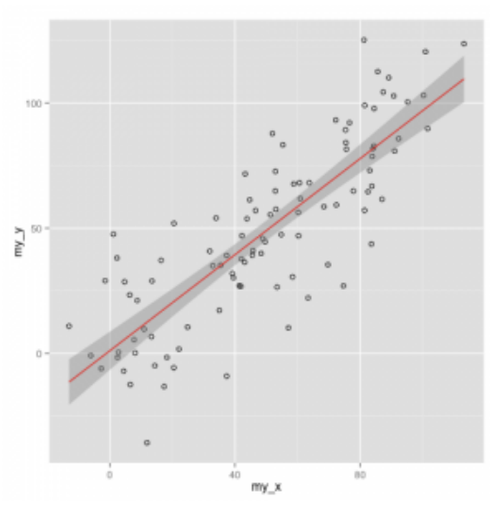


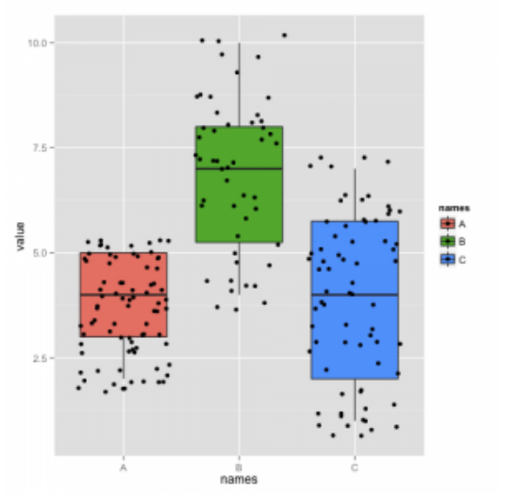






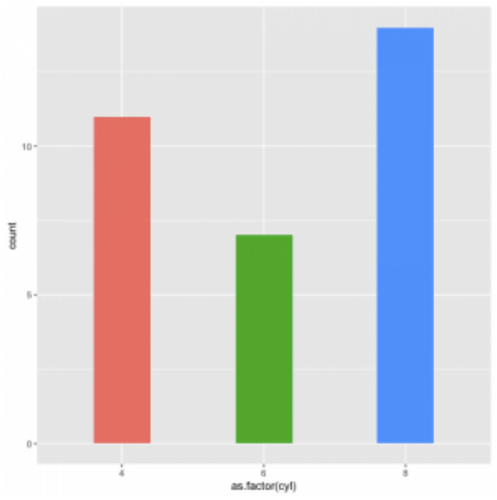


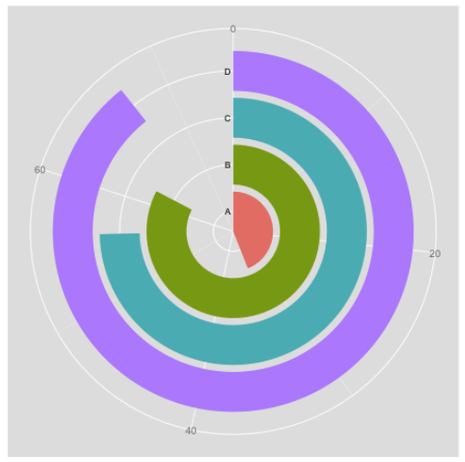


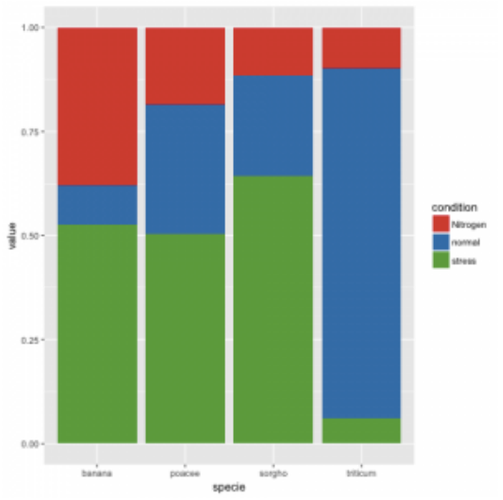


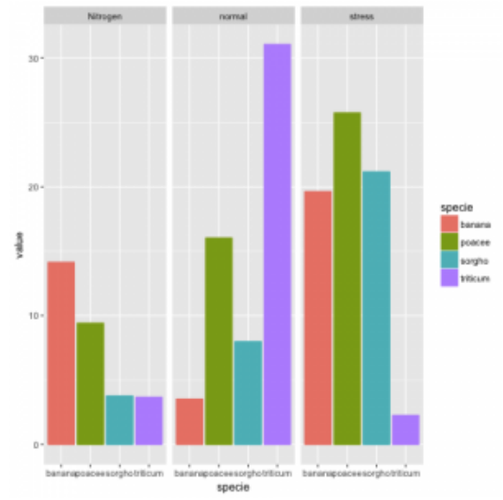
Why R?

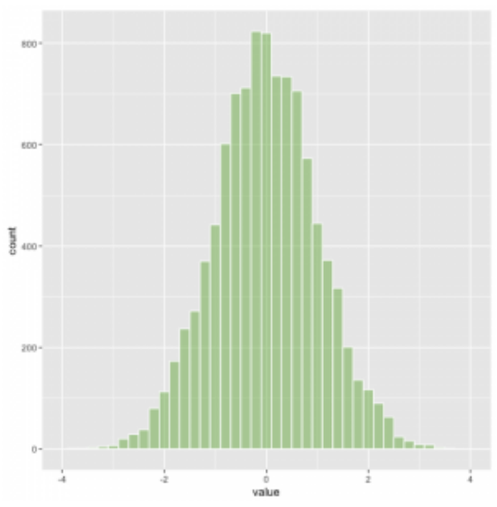
- **R** is **statistical**.
- Use R for **data visualisation**.

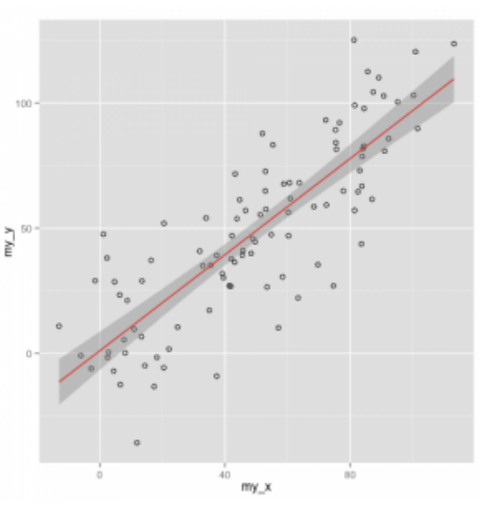


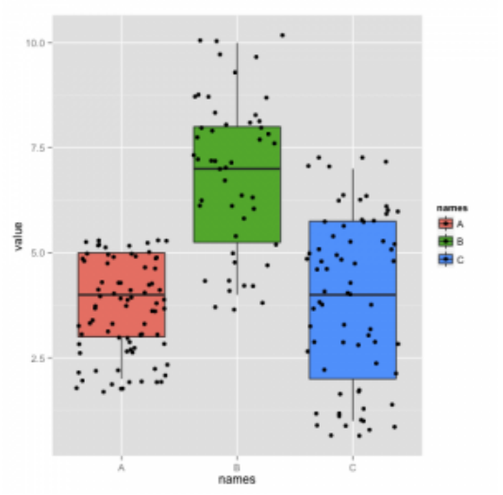


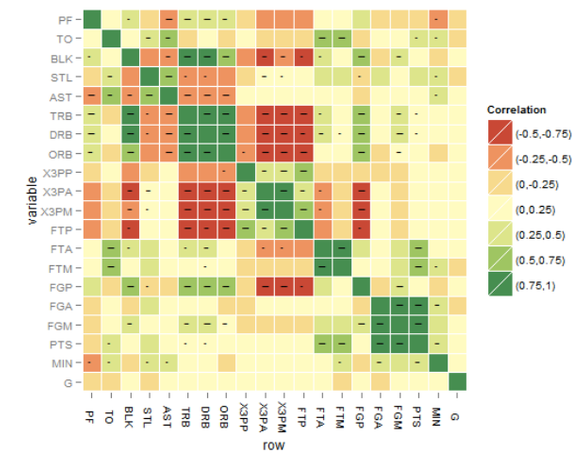






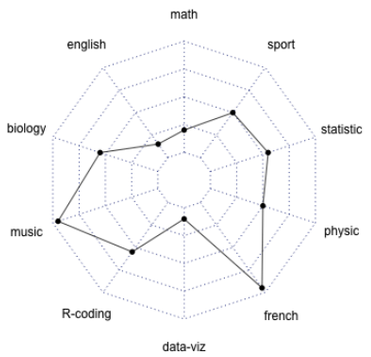


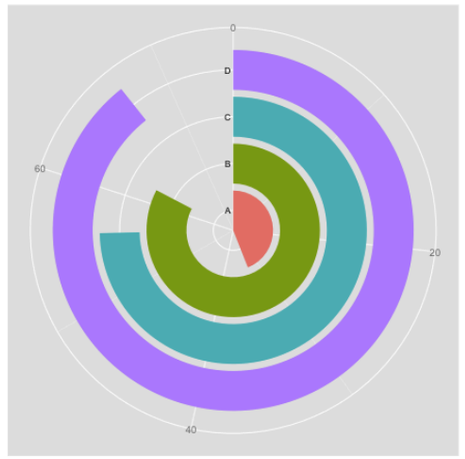


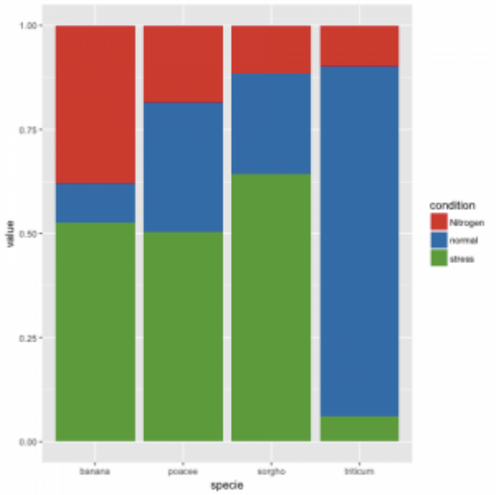


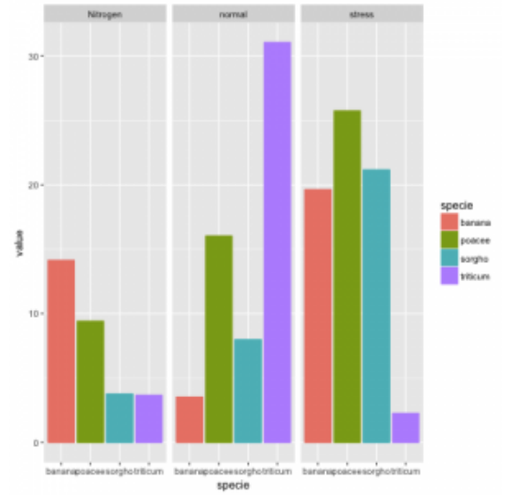
Why R ?

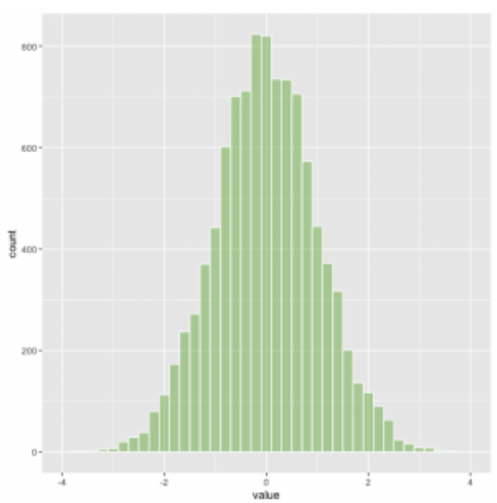
- **R** is **statistical**.
- Use R for **data visualisation**.

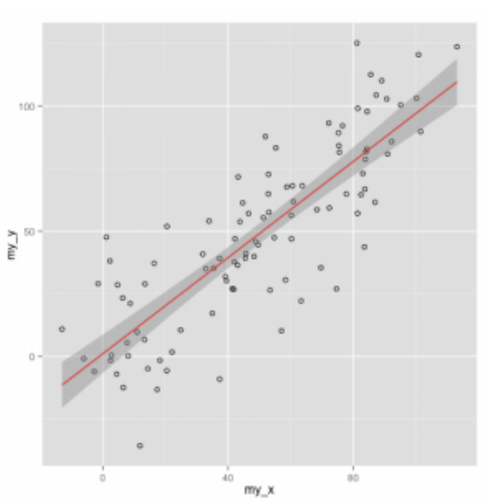


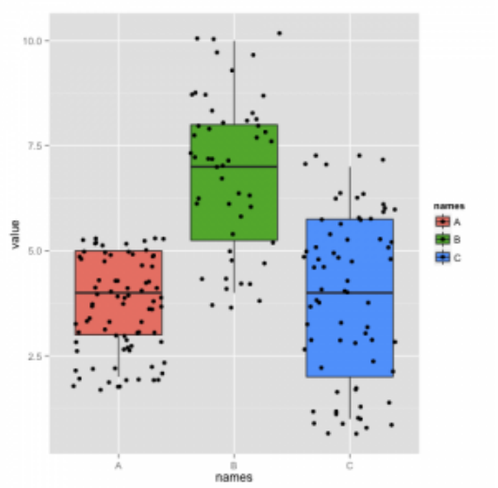


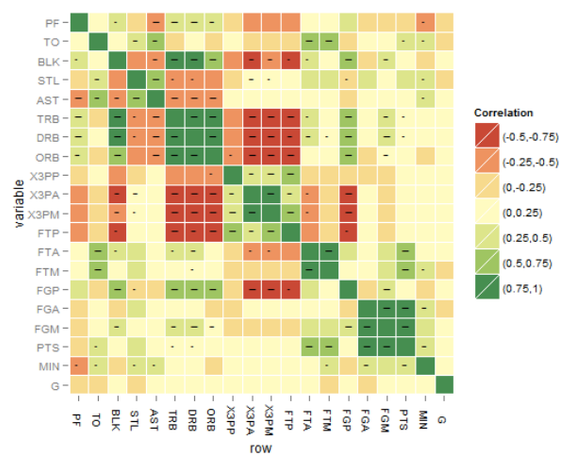






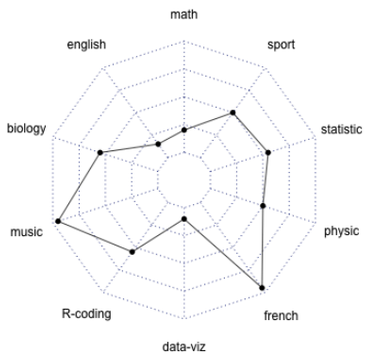


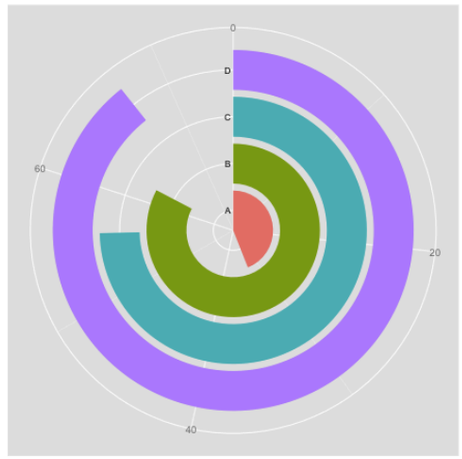


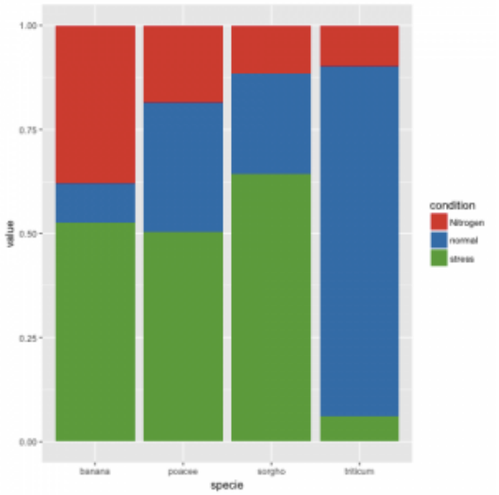


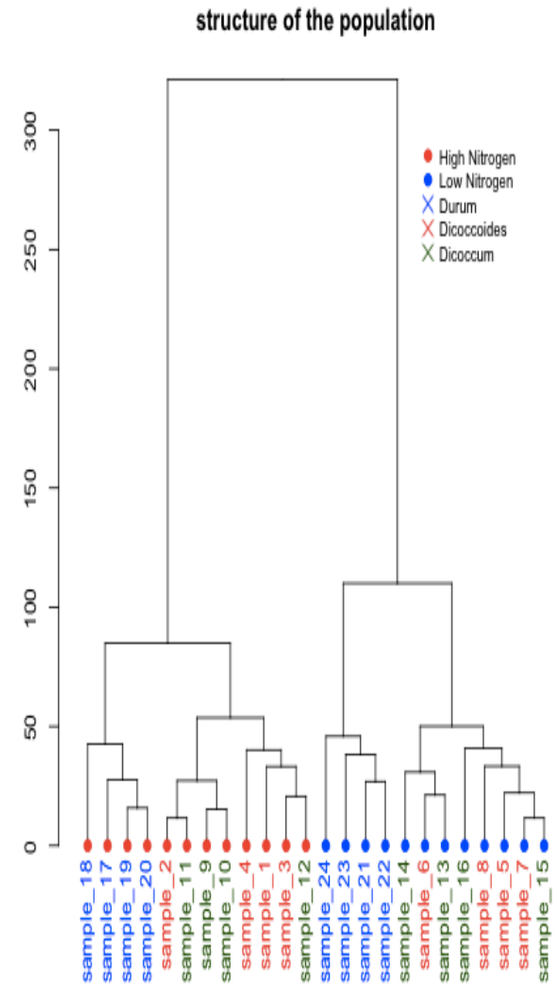
Why R ?

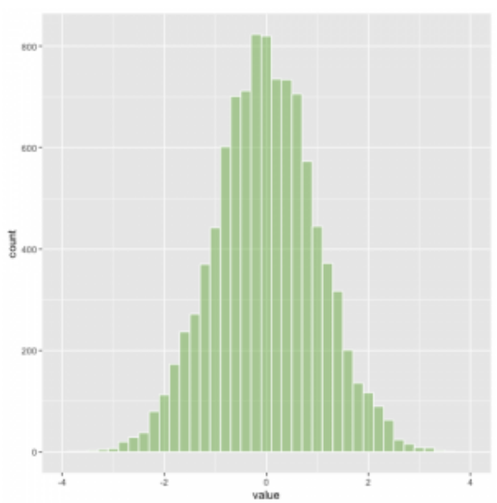
- **R** is **statistical**.
- Use R for **data visualisation**.

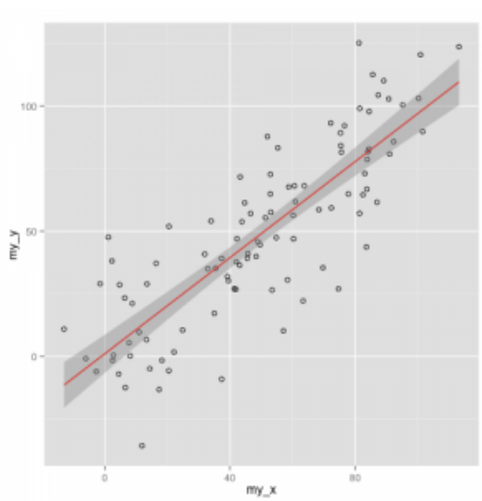


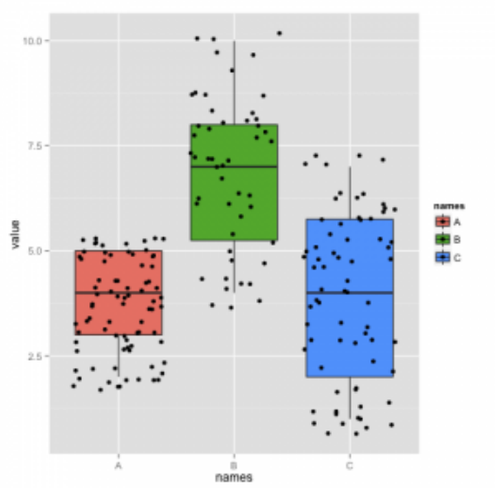






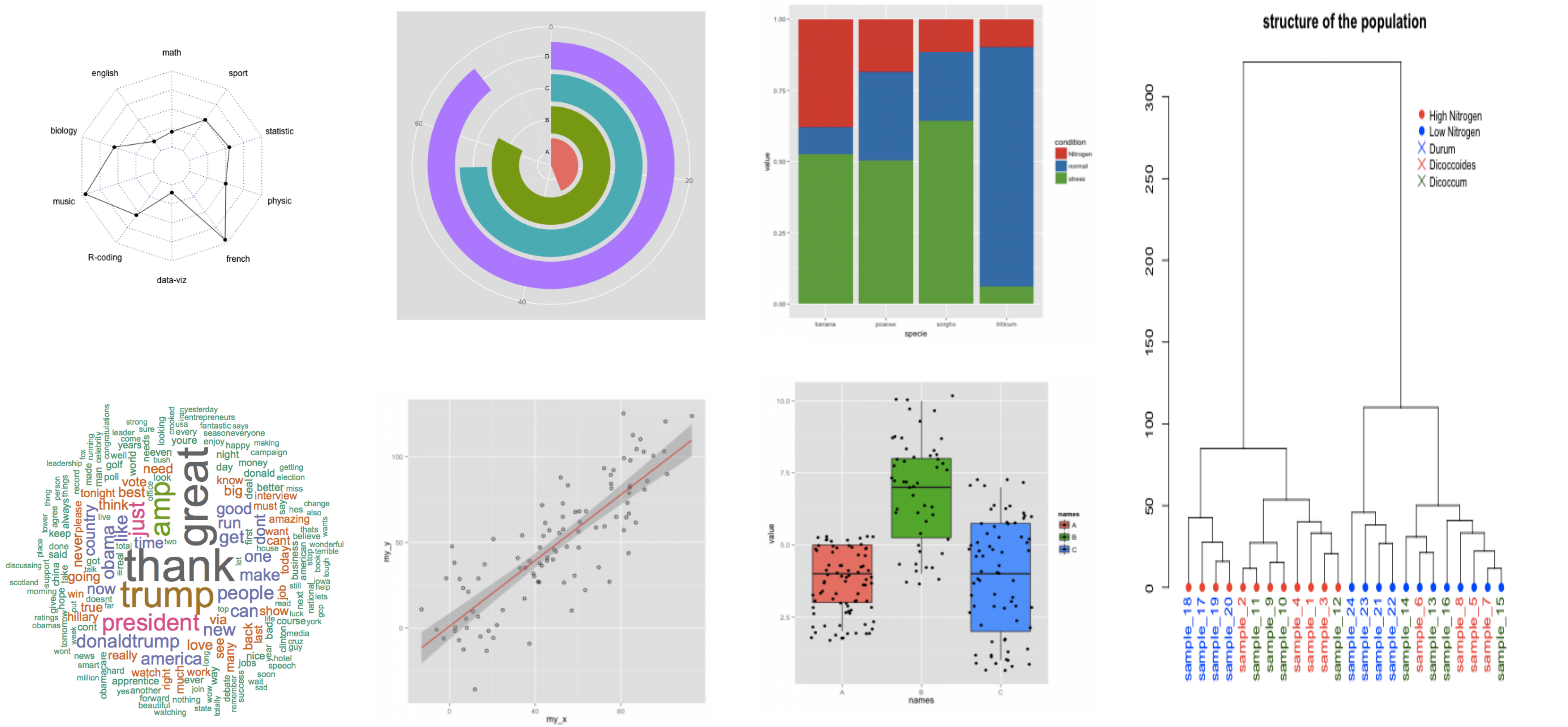






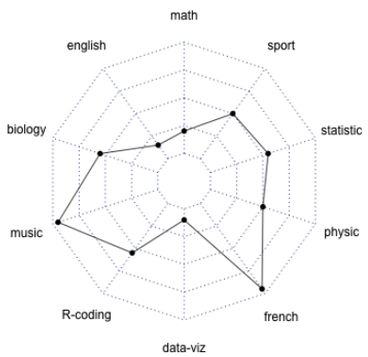
Why R ?

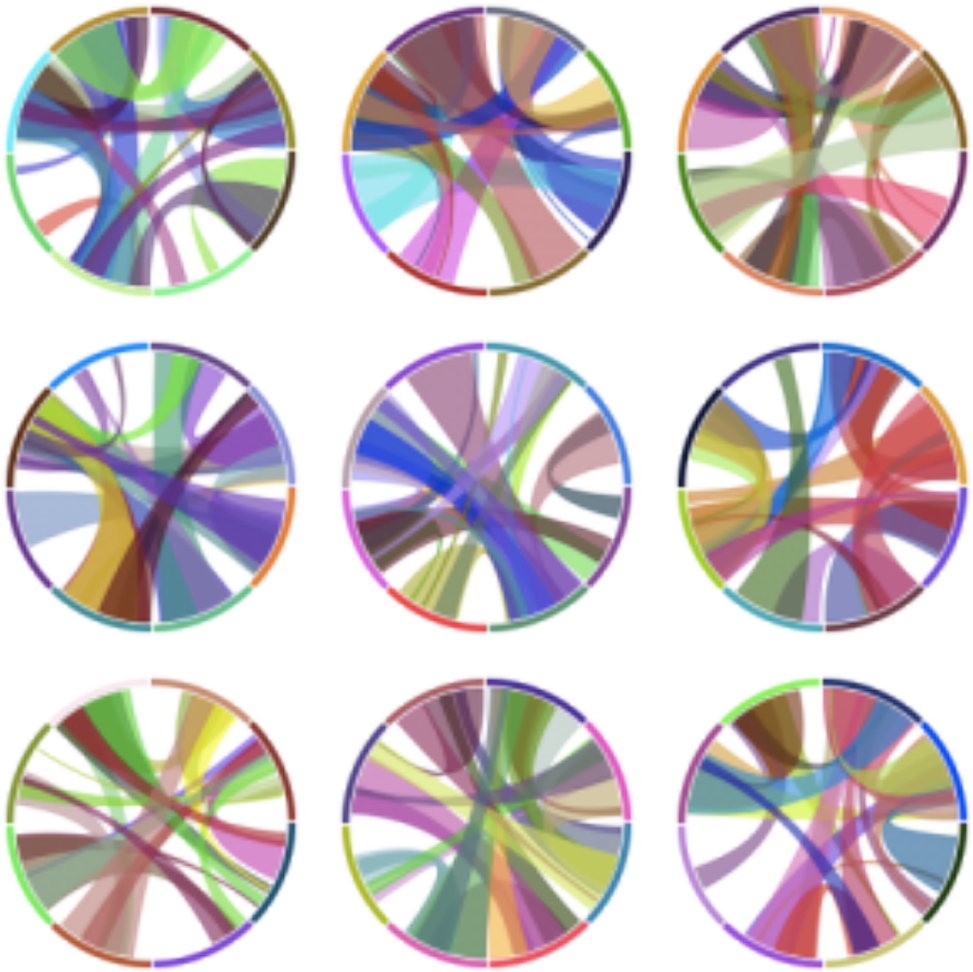
- **R** is **statistical**.
- Use R for **data visualisation**.

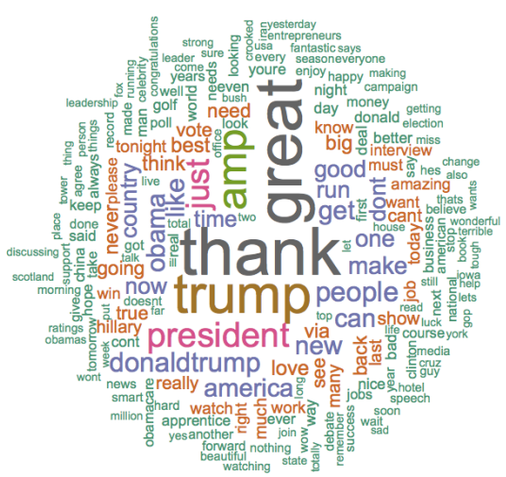


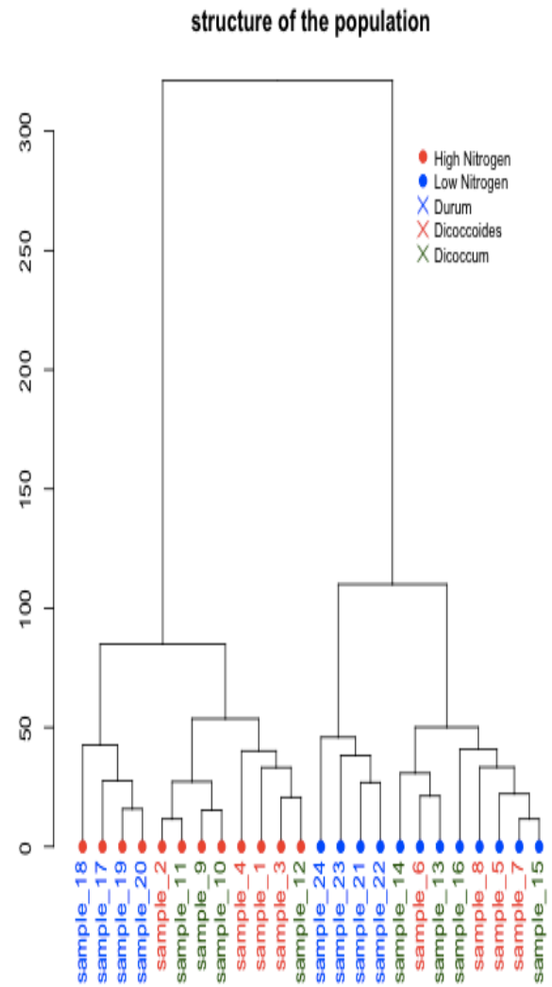
Why R?

- **R** is **statistical**.
- Use R for **data visualisation**.



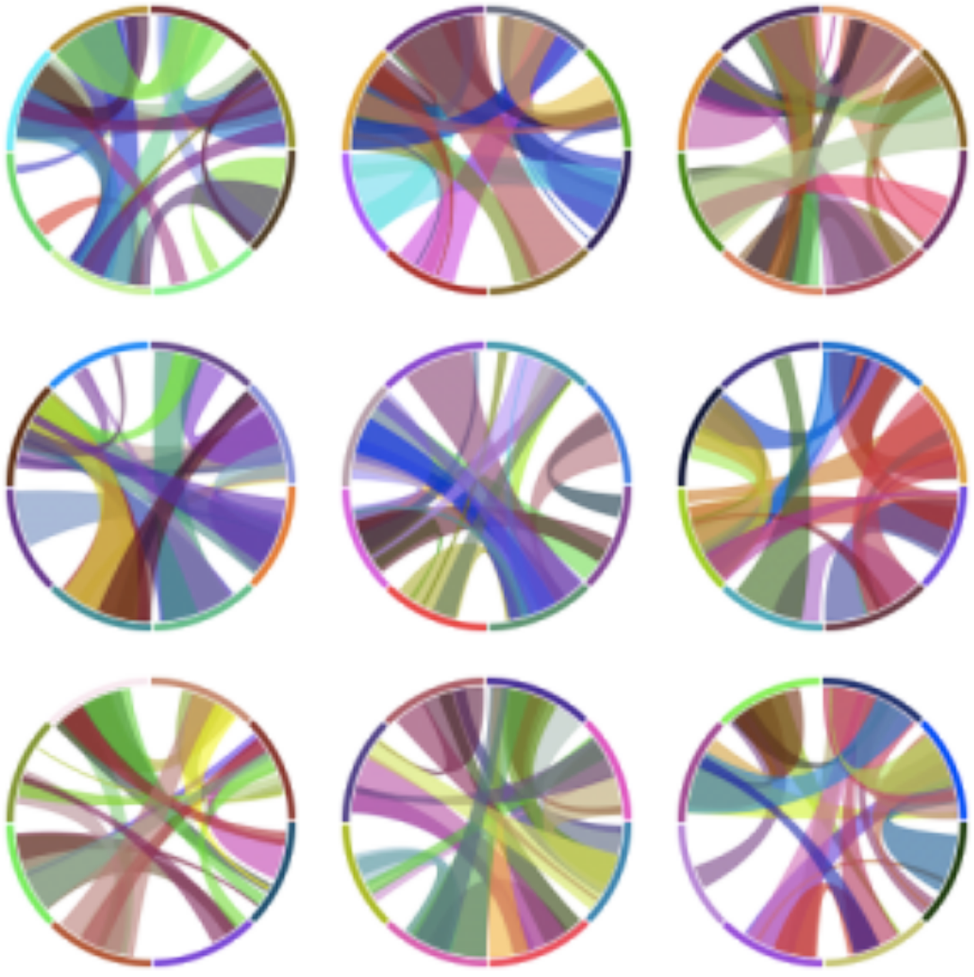


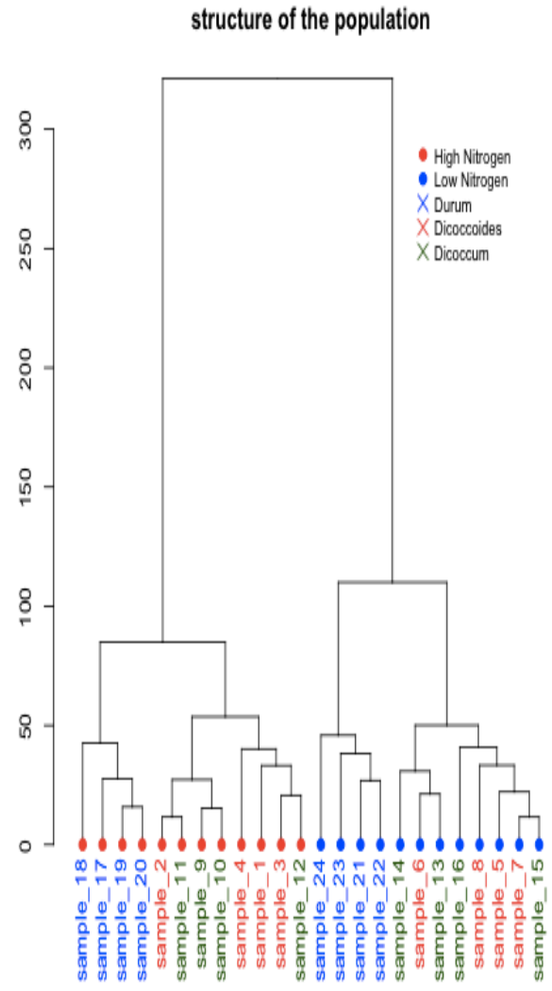




Why R?

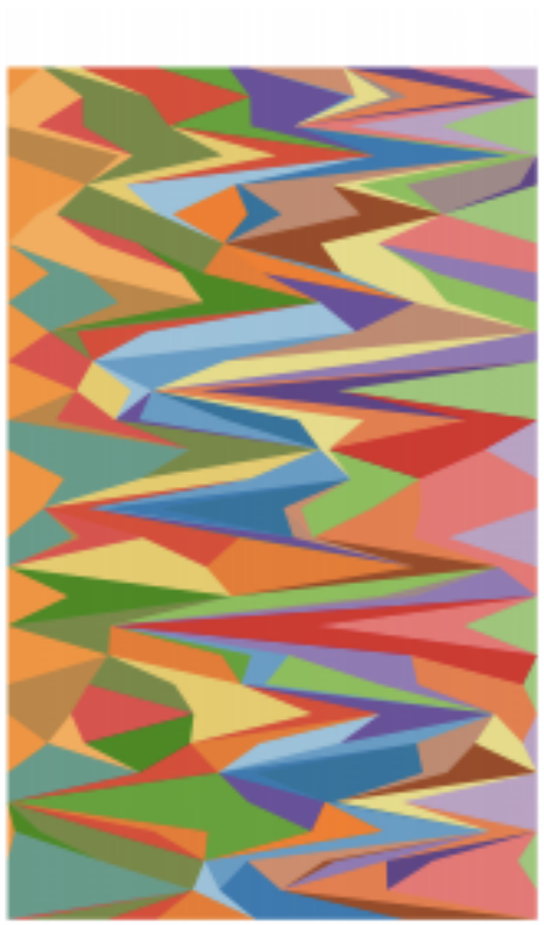
- **R** is **statistical**.
- Use R for **data visualisation / art**.

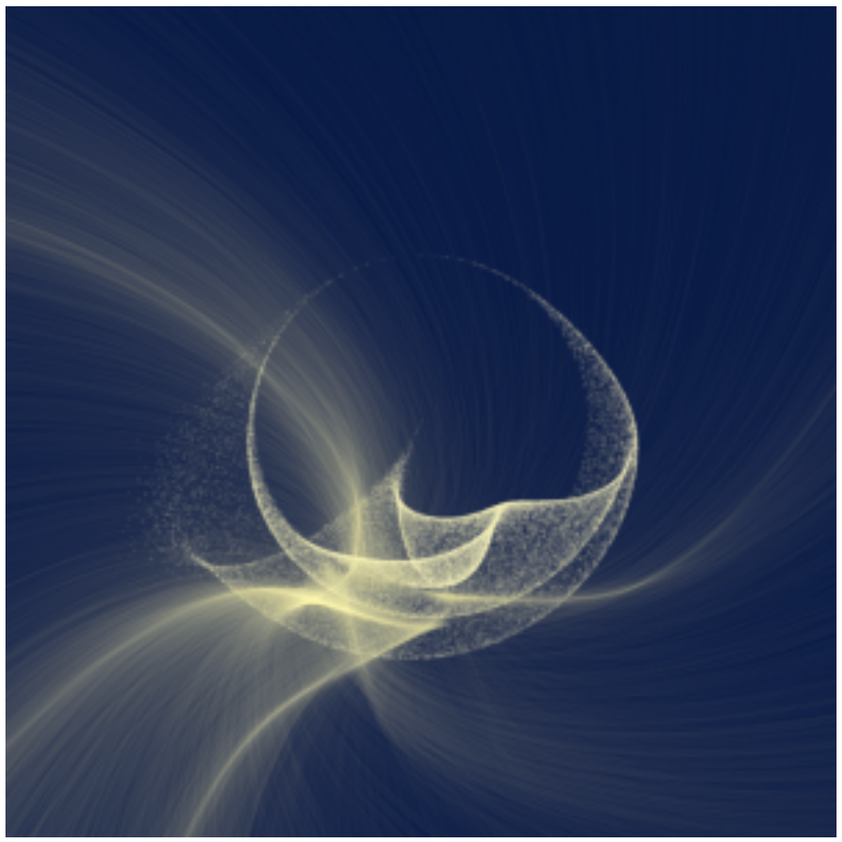


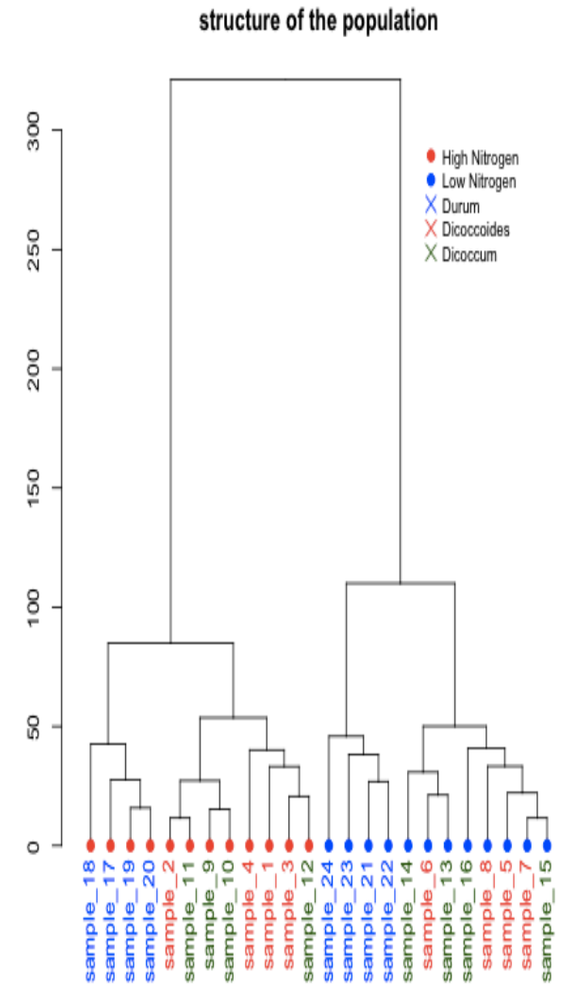


Why R?

- **R** is **statistical**.
- Use R for **data visualisation / art**.

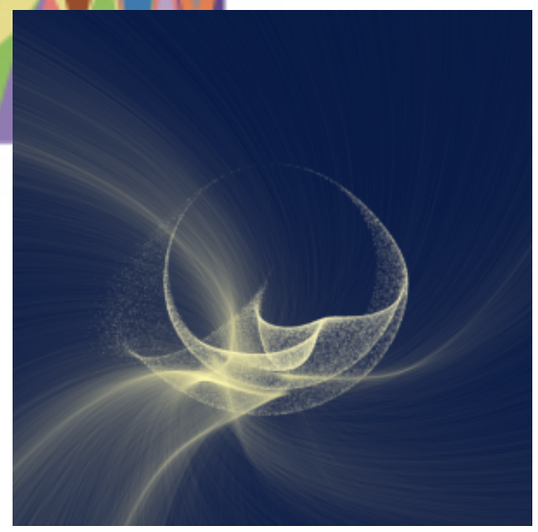




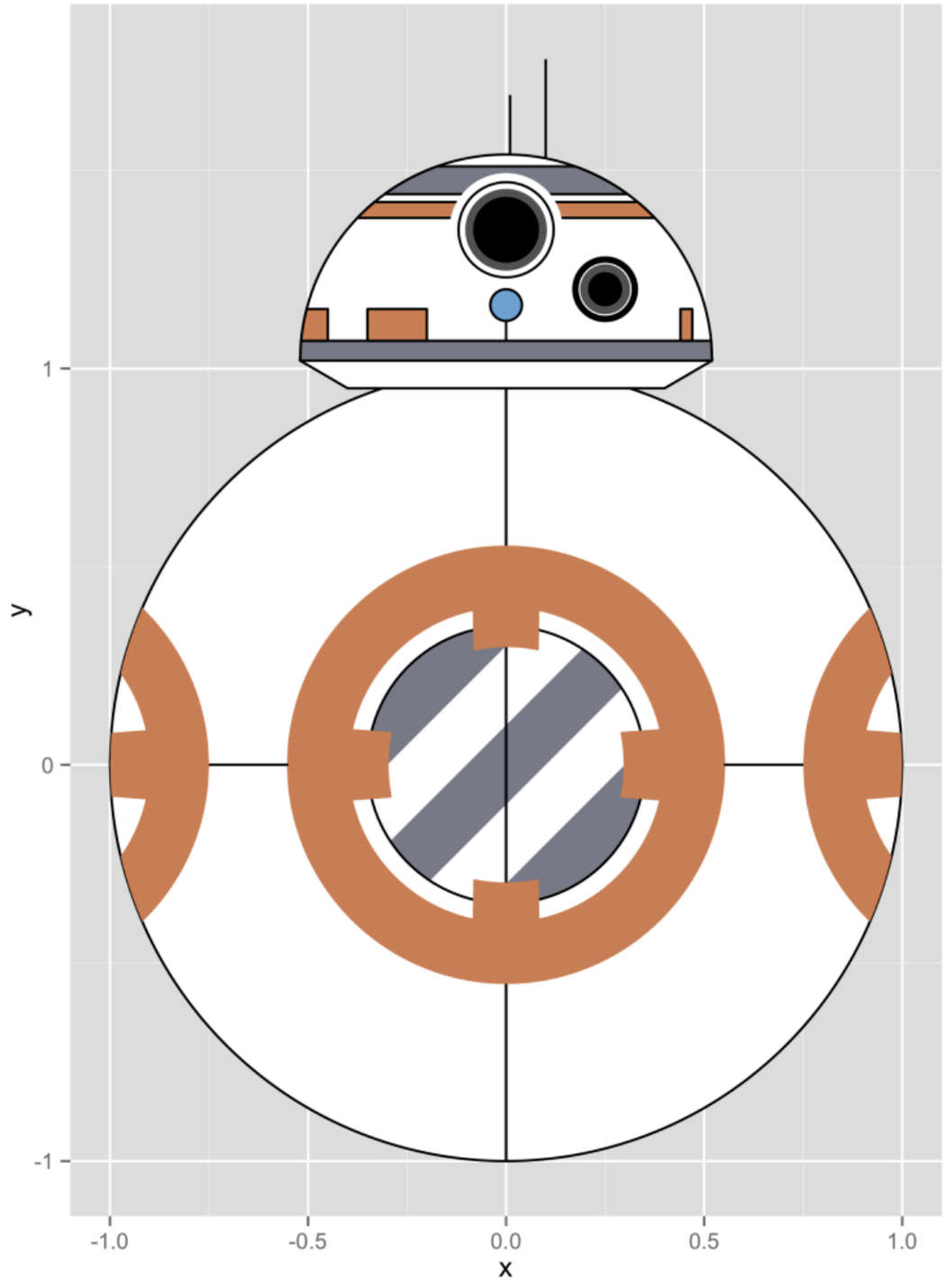


Why R?

- **R** is **statistical**.
- Use R for **data art**.

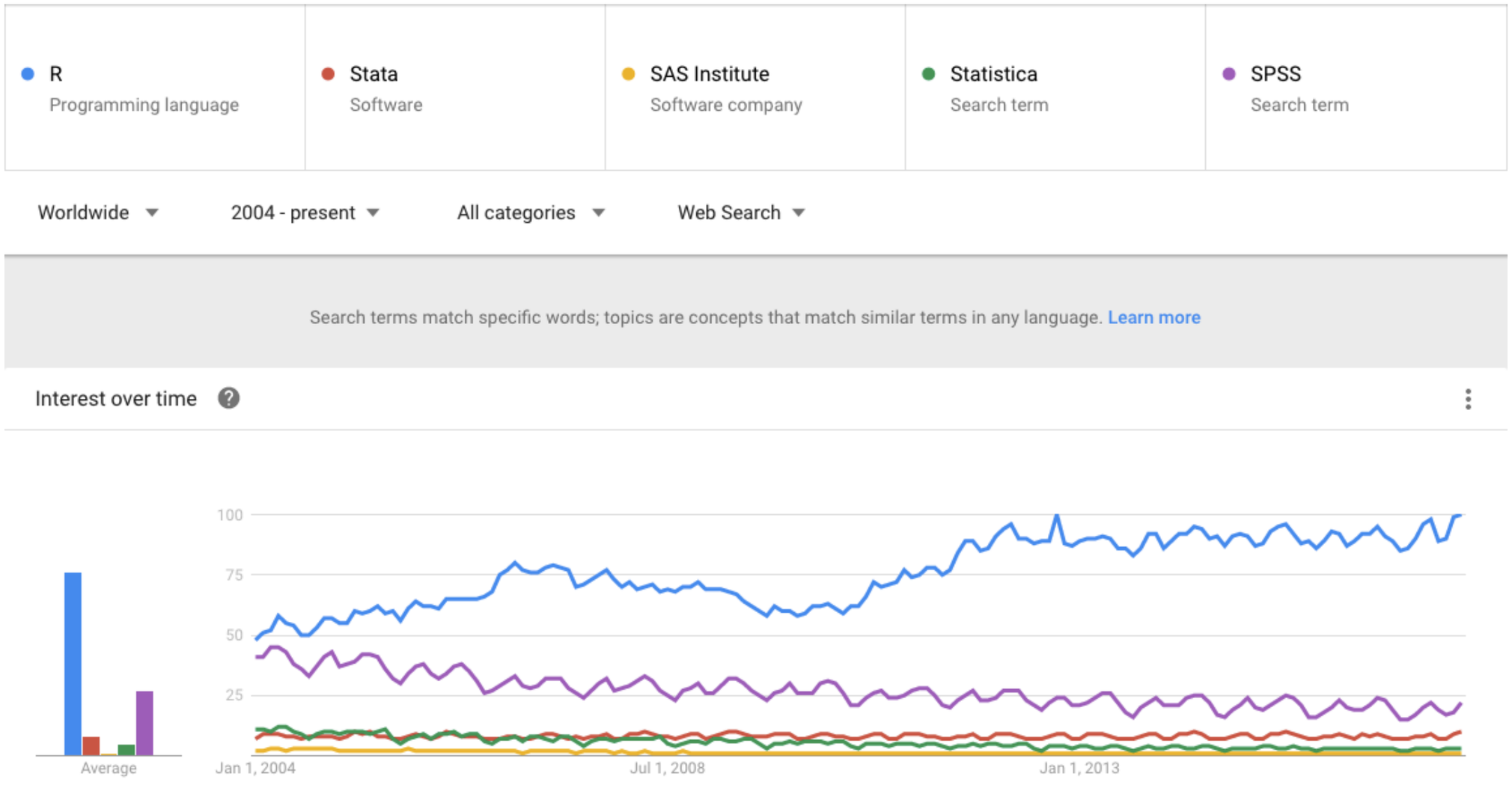


* “Droid”. https://goo.gl/kYXRRw



Why R?

- **R** is **popular**.
  - Google trends data (https://goo.gl/jyOViq)



Why  ?

- **R** is **popular**.
  - Google trends data (https://goo.gl/jyOViq)
  - With popularity comes a ***large* community**.

Why  ?

- **R** is **popular**.
  - Google trends data (https://goo.gl/jyOViq)
  - With popularity comes a ***large* community**.

    ☑Better support — easy to get help.
    - R Mailing lists: https://www.r-project.org/mail.html (*R-help*, *R-package-devel*, etc.)
    - http://stackoverflow.com/questions/tagged/r
    - https://www.r-bloggers.com/

Why R?

- **R** is **popular**.
  - Google trends data (https://goo.gl/jyOViq)
  - With popularity comes a ***large* community**.

    ☑Better support — easy to get help.
    - R Mailing lists: https://www.r-project.org/mail.html (*R-help*, *R-package-devel*, etc.)
    - http://stackoverflow.com/questions/tagged/r
    - https://www.r-bloggers.com/

☑More developers — many packages available.

- Ranging from *Rcpp* to *ggplot2* to *Bioconductor*!

WARNING: COMPLETELY FOR BEGINNERS!

IN TODAY'S GUIDE...

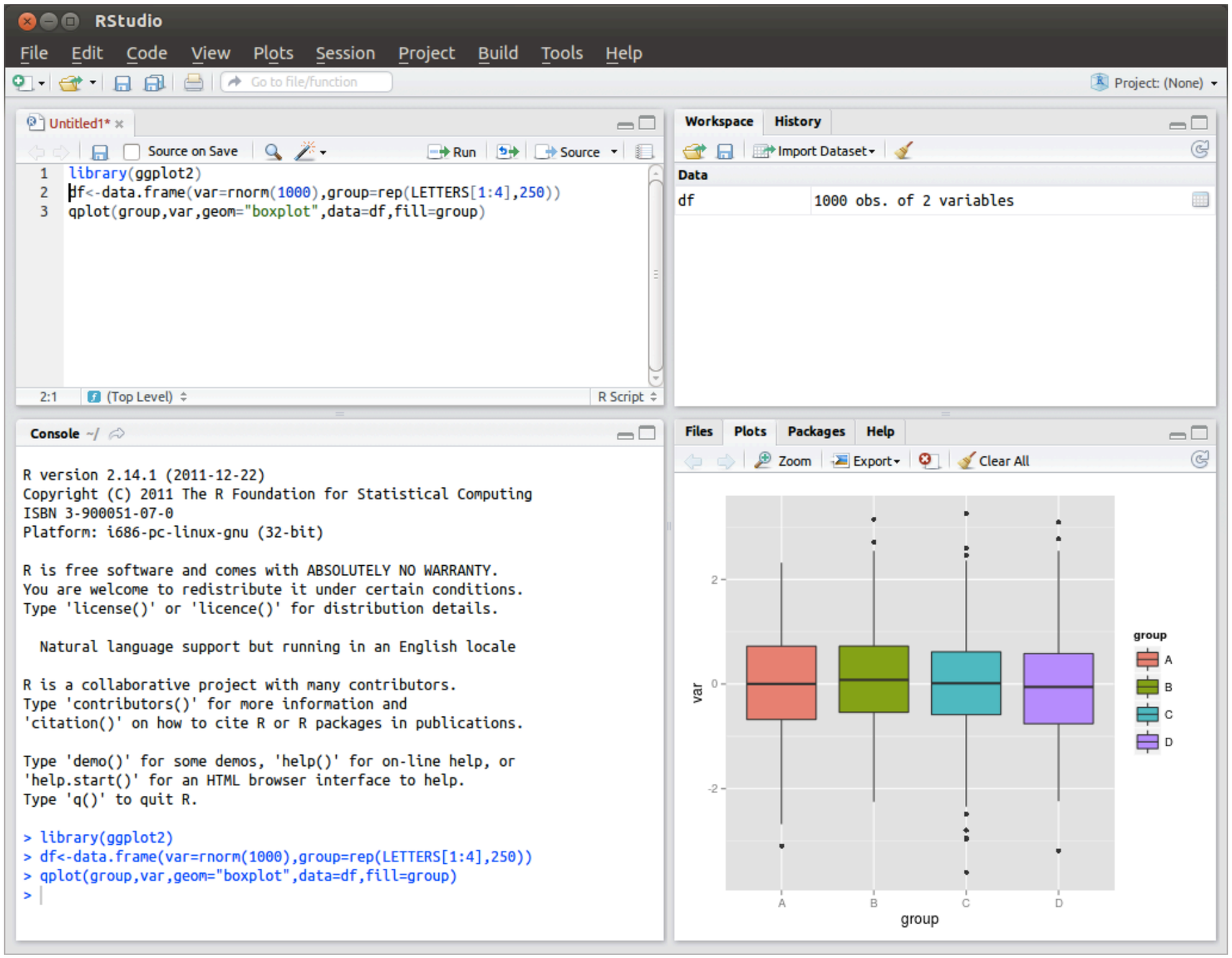
1. What is R? Why R?
2. Installation and "Hello World!" in R
3. R data types – vectors, matrices and data frames
4. R operators and managing a data frame
5. I/O and basic graphs in R
6. Pop quiz

Installing 

- The **Comprehensive R Archive Network** (CRAN) is your friend!
- **Linux**: I assume you could find your own way…
  - RedHat-based: `sudo yum install` (or `sudo dnf install`)
  - Debian-based: `sudo apt-get install`
  - Slackware-based: ~~You are on your own~~ https://slackbuilds.org/repository/13.37/academic/R/
- **Windows**: https://cran.r-project.org/bin/windows/base/
- **Mac OS X**: https://cran.r-project.org/bin/macosx/

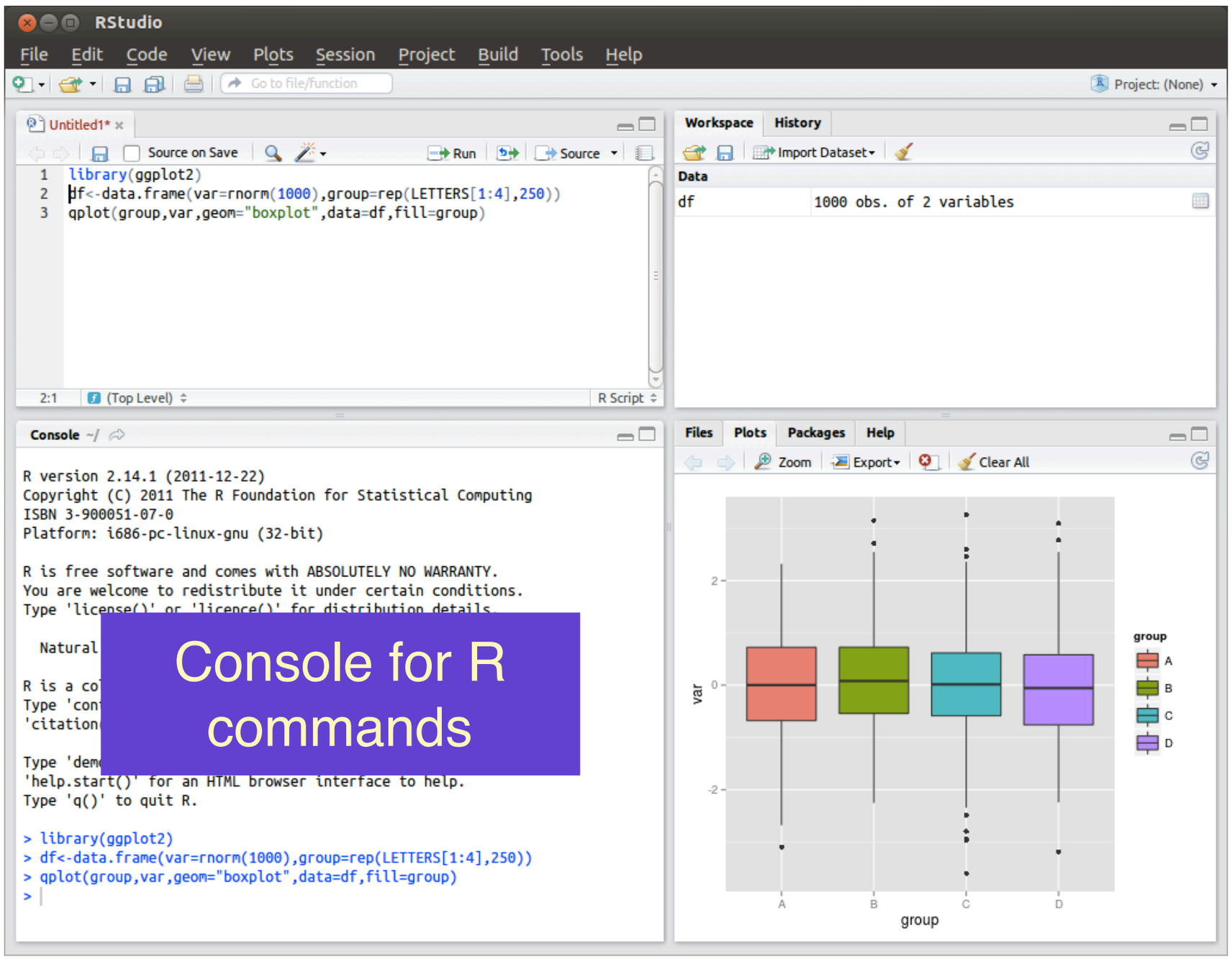
(Optional) Installing RStudio®

- An open-source integrated development environment for R, available via: https://www.rstudio.com/products/rstudio/download/



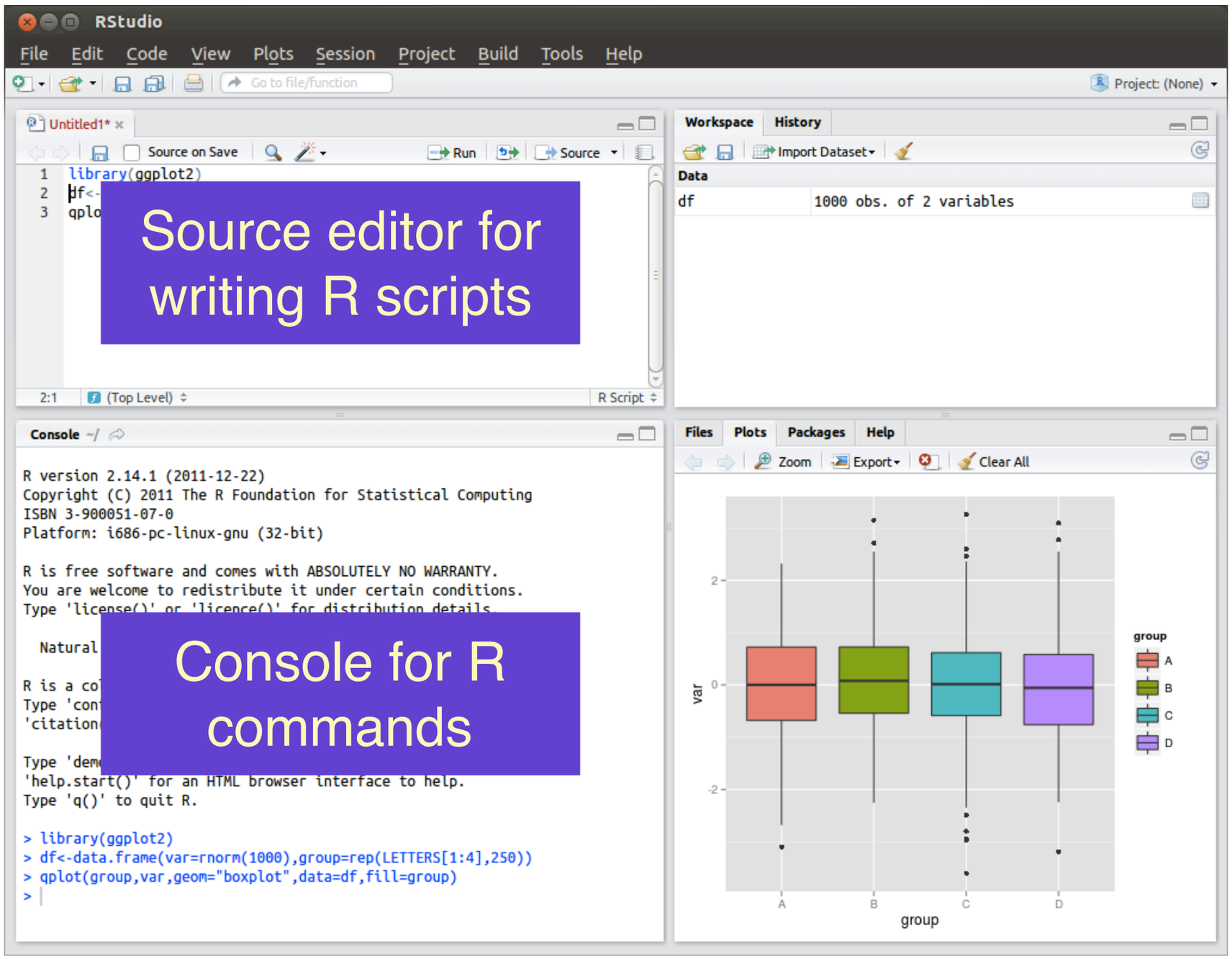
(Optional) Installing RStudio®

- An open-source integrated development environment for R, available via: https://www.rstudio.com/products/rstudio/download/



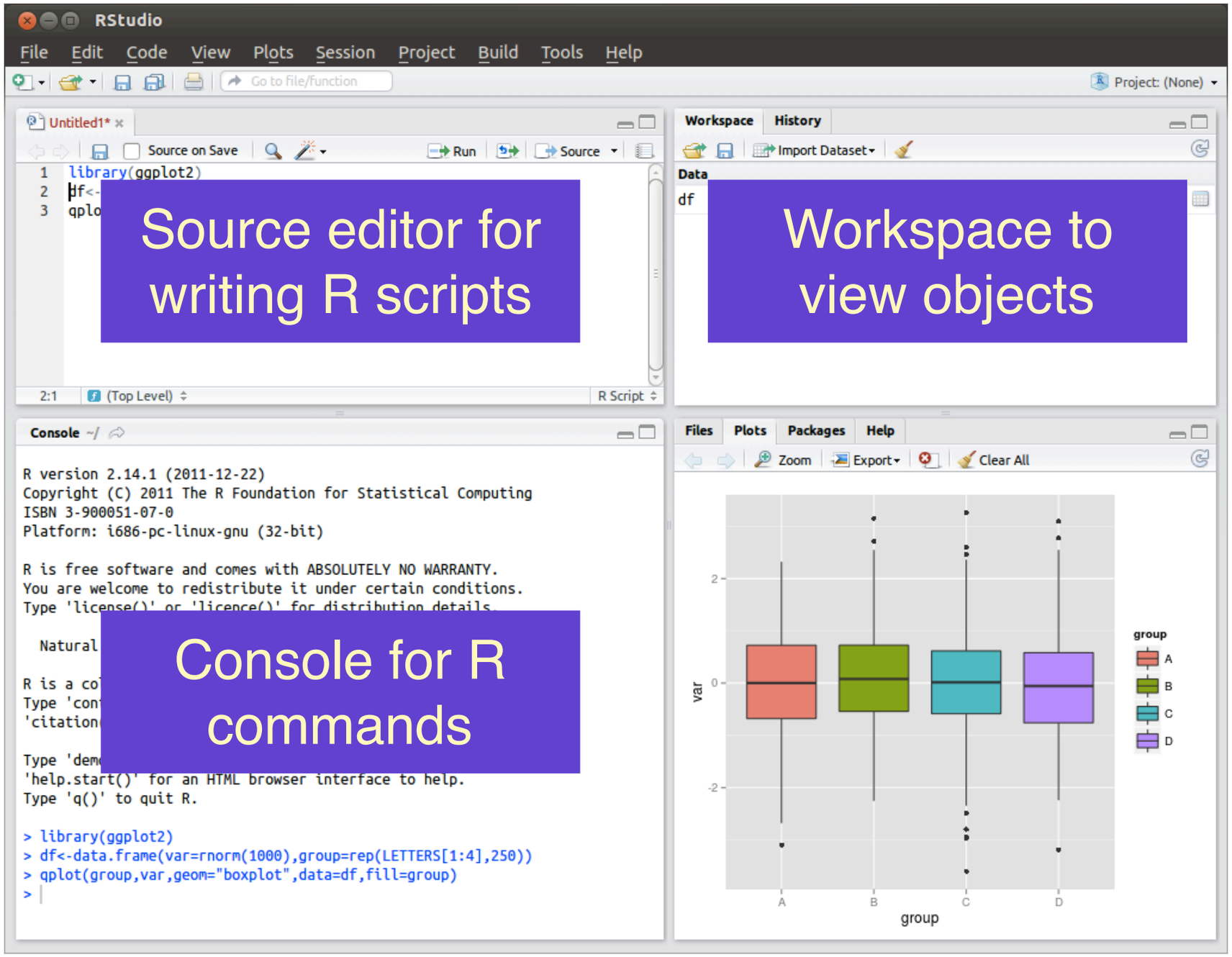
(Optional) Installing RStudio®

- An open-source integrated development environment for R, available via: https://www.rstudio.com/products/rstudio/download/



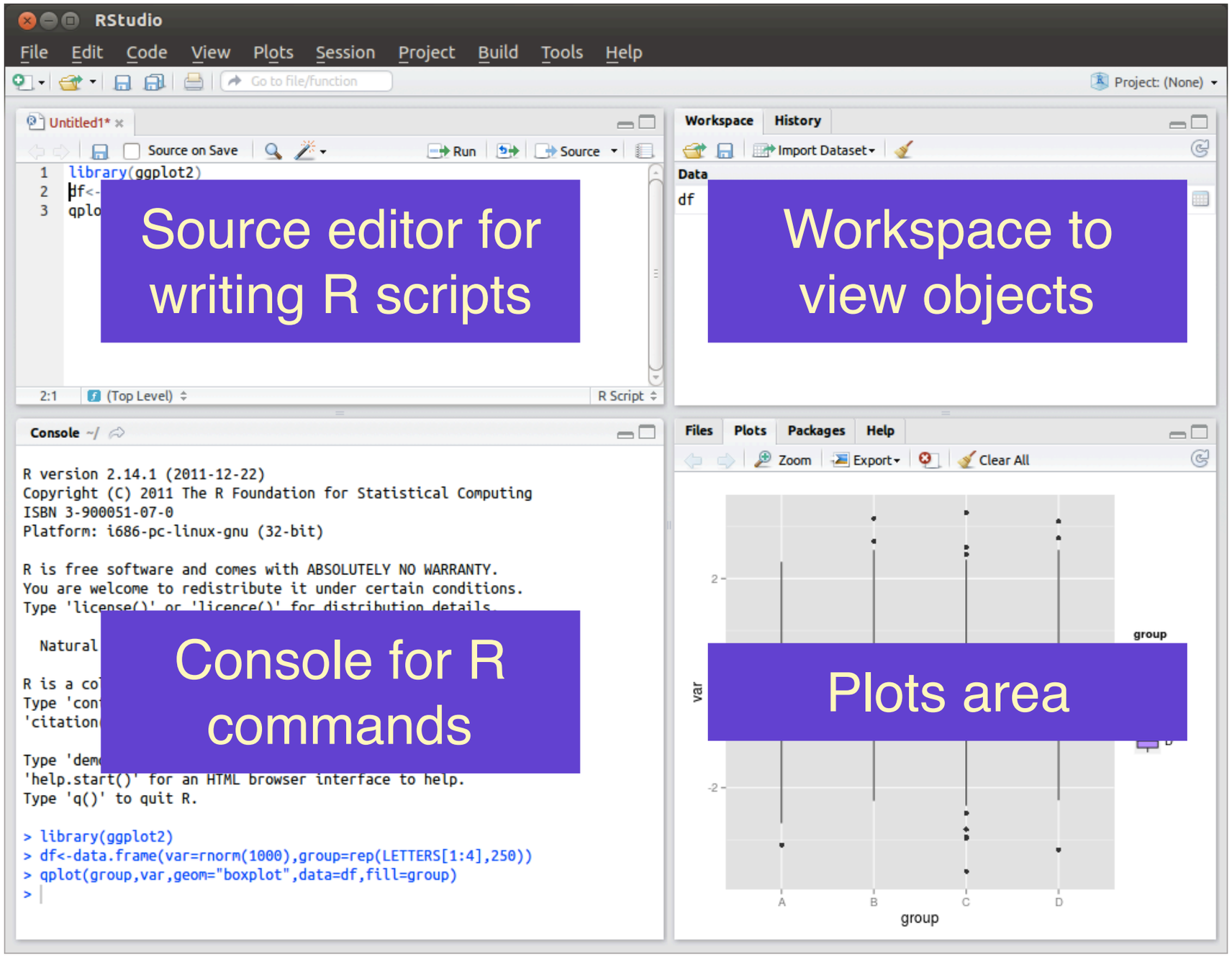
(Optional) Installing RStudio®

- An open-source integrated development environment for R, available via: https://www.rstudio.com/products/rstudio/download/



(Optional) Installing RStudio®

- An open-source integrated development environment for R, available via: https://www.rstudio.com/products/rstudio/download/



Running R

```
[y_li@aerodynamik ~]$ R

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

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

  Natural language support but running in an English locale

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

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

> |
```

Running R

```
[y_li@aerodynamik ~]$ R

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

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

  Natural language support but running in an English locale

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

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

>
```

The R prompt

Running R

```
[y_li@aerodynamik ~]$ R

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

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

  Natural language support but running in an English locale

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

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

> print("Hello World!")
```

Type a command

Running R

```
[y_li@aerodynamik ~]$ R

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

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

  Natural language support but running in an English locale

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

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

> print("Hello World!")
[1] "Hello World!"
>
```

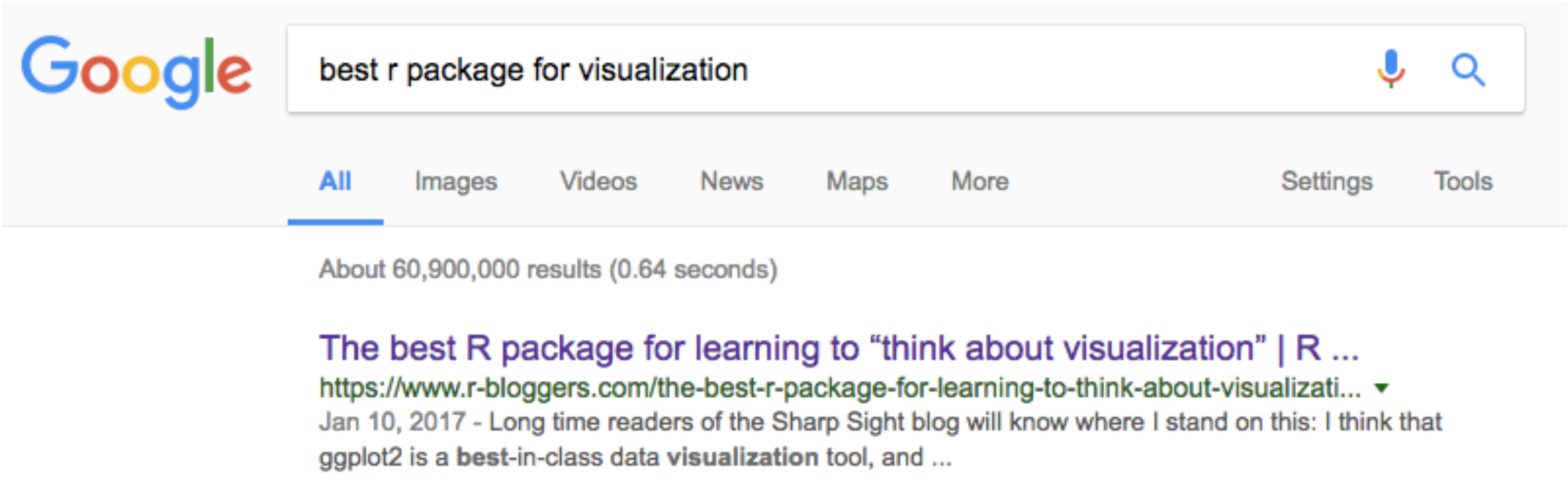
- Hit “Enter”
- R evaluates the expression and prints to screen the output

Installing R packages

1. Google for the R package you desire.

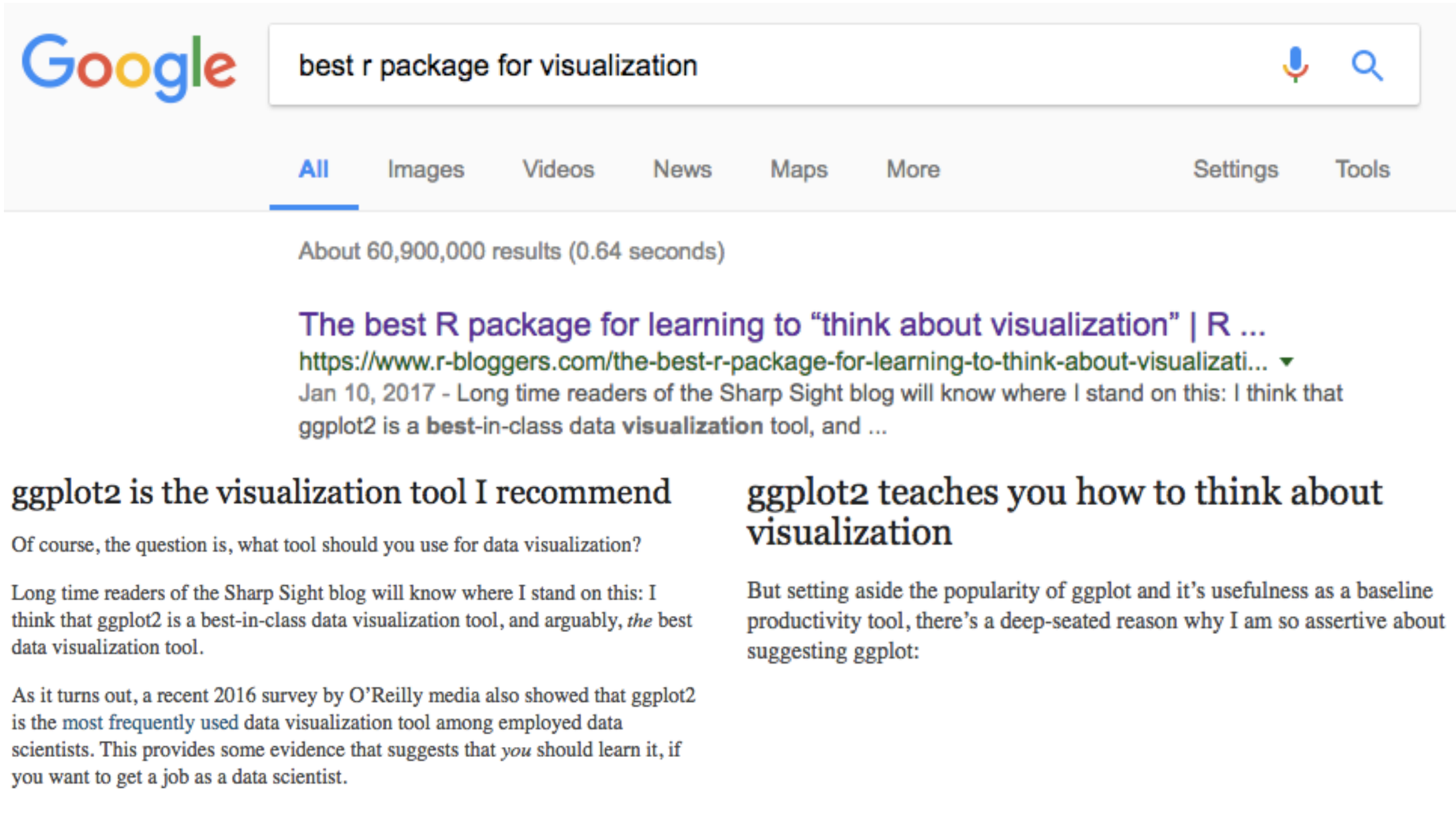
Installing R packages

1. Google for the R package you desire.



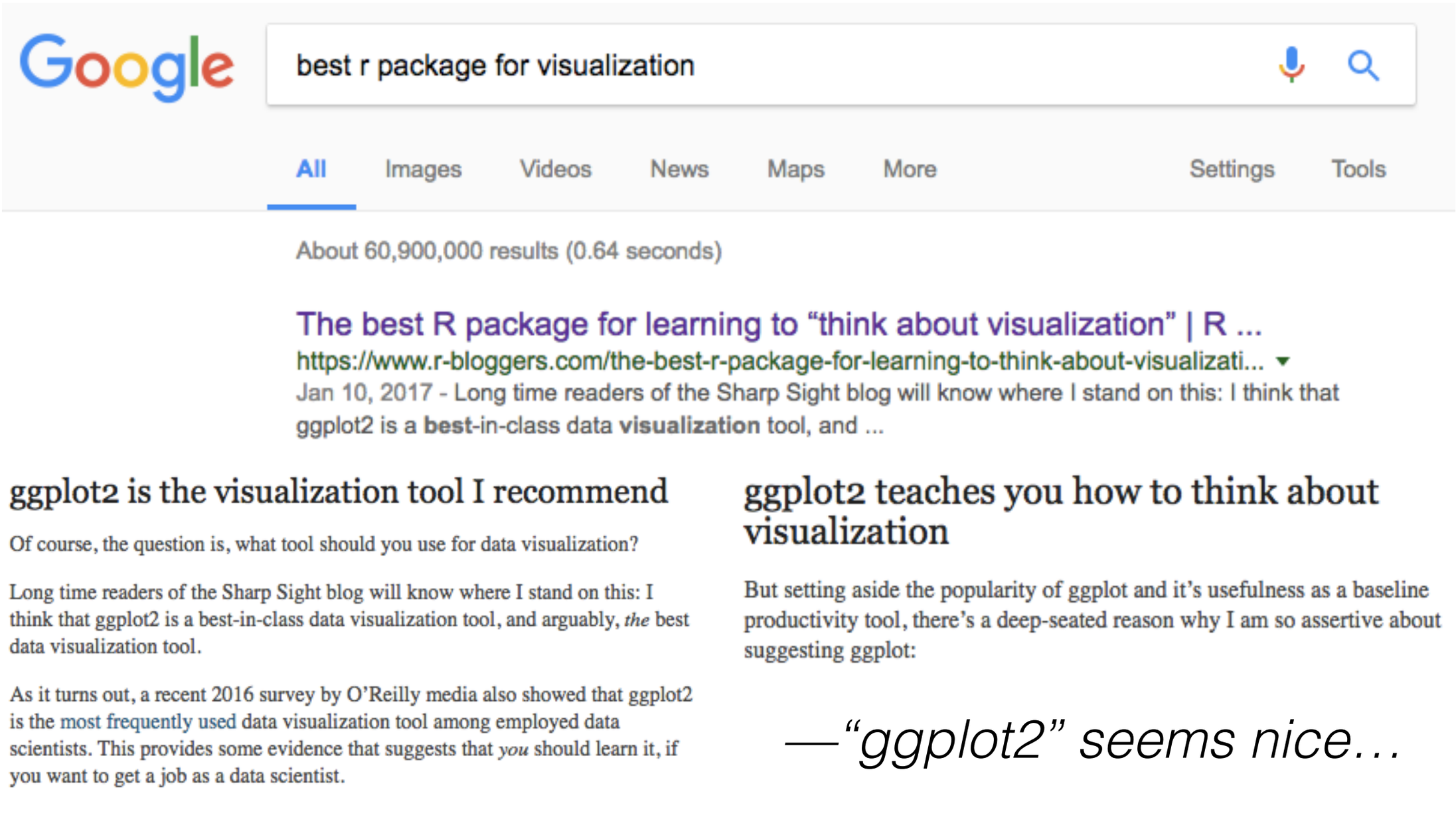
Installing R packages

1. Google for the R package you desire.



Installing R packages

1. Google for the R package you desire.



Installing R packages

1. Google for the R package you desire.
2. Open R and give the package installation command.
   - `> install.packages("ggplot2")`
   - You would be asked to choose a mirror. Just choose one close to you — if the mirror is broken, try another one.

Installing R packages

1. Google for the R package you desire.
2. Open R and give the package installation command.
   - `> install.packages("ggplot2")`
   - You would be asked to choose a mirror. Just choose one close to you — if the mirror is broken, try another one.
3. Have some tea and wait for the installation to finish.

Installing R packages

1. Google for the R package you desire.
2. Open R and give the package installation command.
   - `> install.packages("ggplot2")`
     - You would be asked to choose a mirror. Just choose one close to you — if the mirror is broken, try another one.
3. Have some tea and wait for the installation to finish.
4. After the installation has finished, load the library.
   - `> library("ggplot2")`

Installing R packages

1. Google for the R package you desire.
2. Open R and give the package installation command.
   - `> install.packages("ggplot2")`
   - You would be asked to choose a mirror. Just choose one close to you — if the mirror is broken, try another one.
3. Have some tea and wait for the installation to finish.
4. After the installation has finished, load the library.
   - `> library("ggplot2")`
5. **Read its manual and enjoy.**

WARNING: COMPLETELY FOR BEGINNERS!

IN TODAY'S GUIDE...

1. What is R? Why R?
2. Installation and "Hello World!" in R
3. R data types – vectors, matrices and data frames
4. R operators and managing a data frame
5. I/O and basic graphs in R
6. Pop quiz

R data types

- R has a wide variety of data types including —
  - **Scalars**
  - **Vectors** (numerical, character, logical)
  - **Matrices**
  - **Data frames**
  - **Lists**
- We could use `class(objectName)` to find out which type an R object is.

R data types

- R has a wide variety of data types including —
  - **Scalars**
  - **Vectors** (numerical, character, logical)
  - **Matrices**
  - **Data frames**
  - **Lists**
- We could use `class(objectName)` to find out which type an R object is.

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector
```

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector
```

Assignment operator ("=" is also okay)
Here we are assigning a value to the vector named "a".

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector
```

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector
```

`c()` is actually a function in R, which **concatenates**, or **combines**.

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

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector
```

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns ?

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`
  - `b[0]` returns ?

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`
  - `b[0]` returns `character(0)`

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`
  - `b[0]` returns `character(0)`
  - `a[10]` returns ?

Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`
  - `b[0]` returns `character(0)`
  - `a[10]` returns `NA`

Vectors

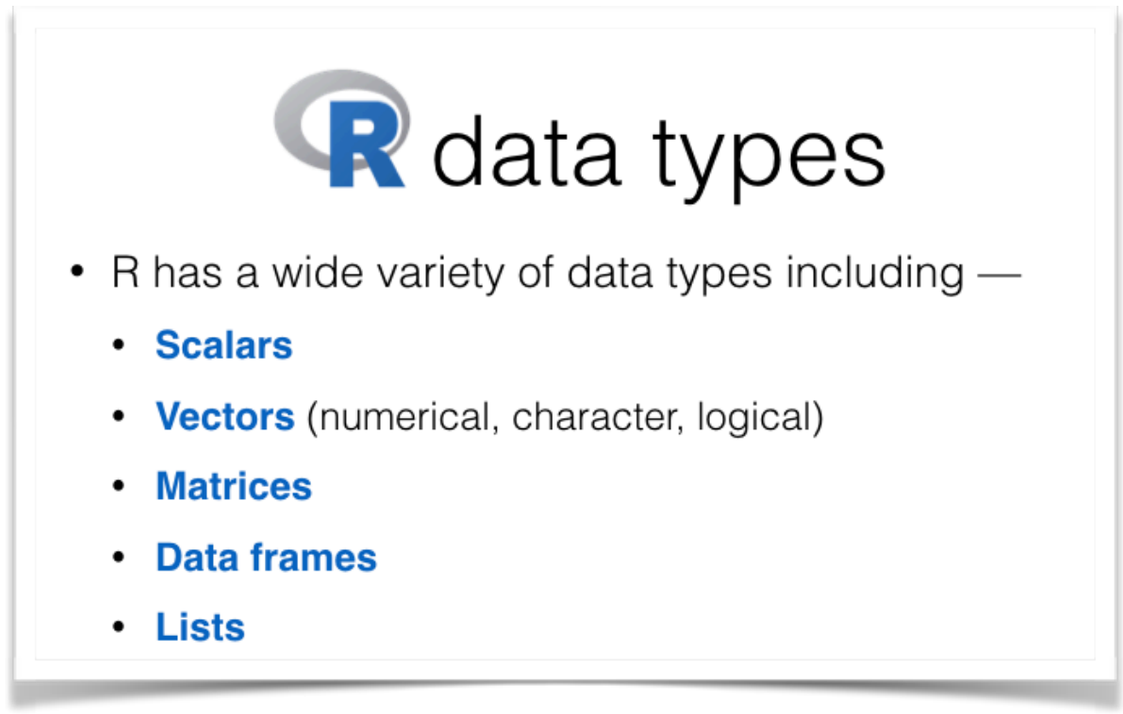
- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`
  - `b[0]` returns `character(0)`
  - `a[10]` returns `NA`



Vectors

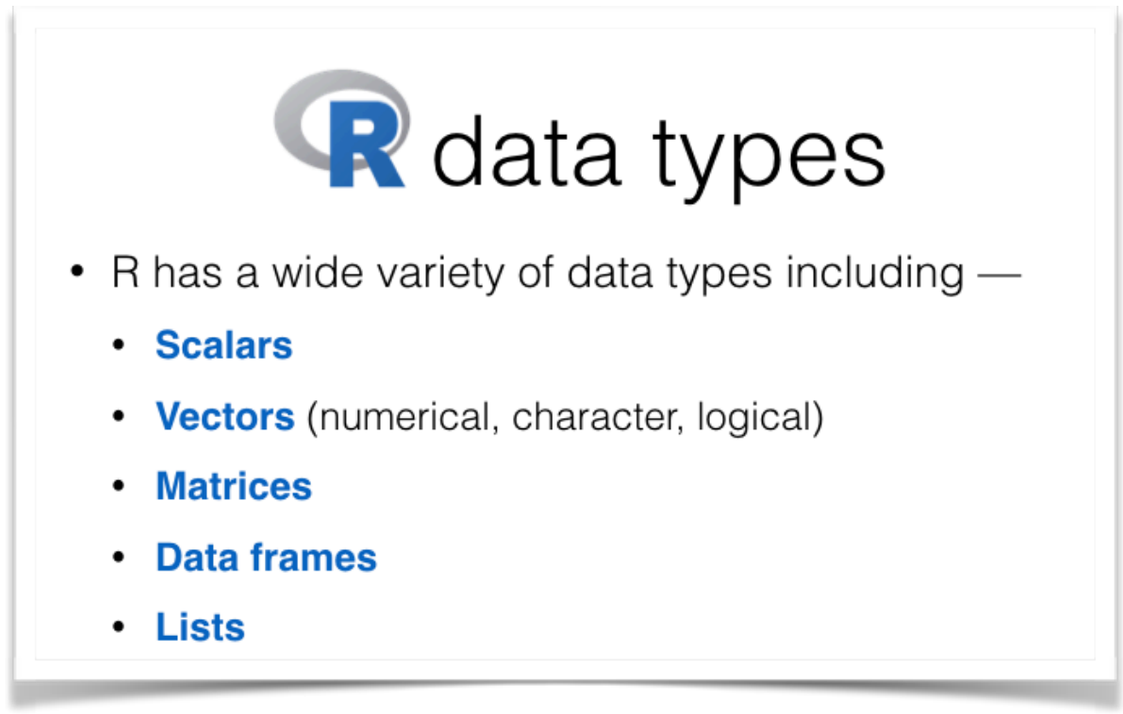
- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`
  - `b[0]` returns `character(0)`
  - `a[10]` returns `NA`
- A **scalar** is just a vector of length 1.



Vectors

- By "**vector**" we usually mean atomic vectors. An **atomic vector** is a linear vector of a **single** primitive type.
- Examples

```
a <- c(1,2,5,3,6,-2,4) # Numeric vector

b <- c("one","two","three") # Character vector

c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) # Logical vector
```

- All **indexing** in R is **base-one**.
  - `a[1]` returns `1`
  - `c[3]` returns `TRUE`
  - `b[0]` returns `character(0)`
  - `a[10]` returns `NA`
- A **scalar** is just a vector of length 1.

How about categorical variables?

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Unordered factor

```
> mons <- c(“March”,”April”,”January”,”November",”January",”September",
“October",“September”,”November”,"August","January","November",
“November",”February",“May”,“August",”July",”December","August",
"August","September","November","February", "April")

> mons2 <- factor(mons) # Convert to unordered factor
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Unordered factor

```
> mons <- c(“March”,”April”,”January”,”November",”January",”September",
“October",“September”,”November”,"August","January","November",
“November",”February",“May”,“August",”July",”December","August",
"August","September","November","February", "April")

> mons2 <- factor(mons) # Convert to unordered factor
```

The part after # is interpreted as comments

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Unordered factor

```
> mons <- c(“March”,”April”,”January”,”November",”January",”September",
“October",“September”,”November”,"August","January","November",
“November",”February",“May”,“August",”July",”December","August",
"August","September","November","February", "April")

> mons2 <- factor(mons) # Convert to unordered factor

> table(mons2) # Build contingency table

mons2
    April     August   December   February    January       July
        2          4          1          2          3          1
    March        May   November    October  September
        1          1          5          1          3
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor

```
> mons3 <- factor(mons,levels=c("January","February","March","April",
"May","June","July","August","September","October","November",
"December"),ordered=TRUE) # Convert to ordered factor
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor

```
> mons3 <- factor(mons,levels=c("January","February","March","April",
"May","June","July","August","September","October","November",
"December"),ordered=TRUE) # Convert to ordered factor

> mons3[1] < mons3[2] # Now we could do comparison
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor

```
> mons3 <- factor(mons,levels=c("January","February","March","April",
"May","June","July","August","September","October","November",
"December"),ordered=TRUE) # Convert to ordered factor

> mons3[1] < mons3[2] # Now we could do comparison

[1] TRUE
```

```
> mons <- c("March","April"
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor

```
> mons3 <- factor(mons,levels=c("January","February","March","April",
"May","June","July","August","September","October","November",
"December"),ordered=TRUE) # Convert to ordered factor
> mons3[1] < mons3[2] # Now we could do comparison
[1] TRUE
> table(mons3) # Build contingency table
mons
  January  February     March     April       May      June
        3         2         1         2         1         0
     July    August September   October  November  December
        1         4         3         1         5         1
```

```
> mons <- c("March","April"
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
> levels(fert)
[1] "10" "20" "50"
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
> levels(fert)
[1] "10" "20" "50"
> mean(as.numeric(levels(fert)[fert]))
# Calculate the mean of the original numeric values of the fert variable
[1] 23.33333
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
> levels(fert)
[1] "10" "20" "50"
> mean(as.numeric(levels(fert)[fert]))
# Calculate the mean of the original numeric values of the fert variable
[1] 23.33333
```

Factor levels of `fert`

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
> levels(fert)
[1] "10" "20" "50"
> mean(as.numeric(levels(fert)[fert]))
# Calculate the mean of the original numeric values of the fert variable
[1] 23.33333
```

When you use a factor as an index, R silently converts it to an integer vector

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
> levels(fert)
[1] "10" "20" "50"
> mean(as.numeric(levels(fert)[fert]))
# Calculate the mean of the original numeric values of the fert variable
[1] 23.33333
```

```
[1] "10" "20" "20" "50" "10" "20" "10" "50" "20"
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
> levels(fert)
[1] "10" "20" "50"
> mean(as.numeric(levels(fert)[fert]))
# Calculate the mean of the original numeric values of the fert variable
[1] 23.33333
```

```
[1] 10 20 20 50 10 20 10 50 20
```

Factors

- A **factor vector** is a special storage class used for **qualitative** data.
  - The values are internally **stored as integers**.
  - Each integer corresponds to a **level**, which is a **character string**.
- Ordered factor: Another example

```
> fert <- c(10,20,20,50,10,20,10,50,20)
> fert <- factor(fert,levels=c(10,20,50),ordered=TRUE)
> fert
[1] 10 20 20 50 10 20 10 50 20
Levels: 10 < 20 < 50
> levels(fert)
[1] "10" "20" "50"
> mean(as.numeric(levels(fert)[fert]))
# Calculate the mean of the original numeric values of the fert variable
[1] 23.33333
```

Take the average of —

```
[1] 10 20 20 50 10 20 10 50 20
```

R data types

- R has a wide variety of data types including —
  - **Scalars**
  - **Vectors** (numerical, character, logical)
  - **Matrices**
  - **Data frames**
  - **Lists**
- We could use `class(objectName)` to find out which type an R object is.

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

```
> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                   # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows

> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                  # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

`A[2,3]`

Element at position (2,3)

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows

> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                  # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

A["row2", "col3"]

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows

> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                  # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

`A["row2", "col3"]`

Refer by row name and column name

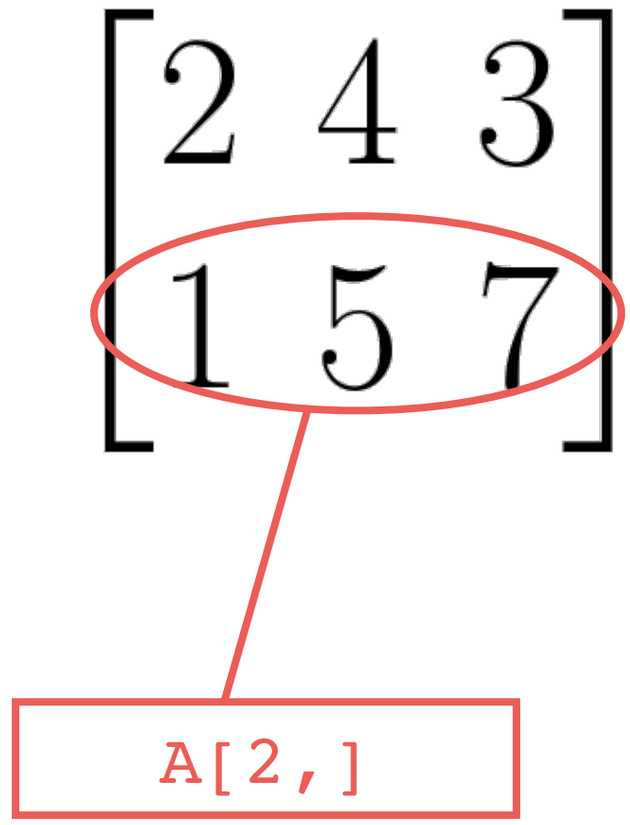
Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows

> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                  # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```



Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows

> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                  # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

`A[2,]`

Get the 2nd row

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows

> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                  # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

A[,3]

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows

> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                  # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

`A[,3]`

Get the 3rd column

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

```
> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                   # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

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

?

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

```
> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                    # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

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

$$\begin{bmatrix} 2 & 3 \\ 1 & 7 \end{bmatrix}$$

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

```
> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                    # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

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

$$\begin{bmatrix} 2 & 3 \\ 1 & 7 \end{bmatrix}$$

Get sub-matrix

Matrices

- A **matrix** is a collection of data elements arranged in a **two-dimensional rectangular** layout. The data elements must be of the **same basic type**.
- Example

```
> A <- matrix(
+   c(2, 4, 3, 1, 5, 7), # The data elements
+   nrow=2,              # Number of rows
+   ncol=3,              # Number of columns
+   byrow = TRUE)        # Fill matrix by rows
```

$$\begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

```
> dimnames(A) <- list(
+   c("row1", "row2"),          # Row names
+   c("col1", "col2", "col3")) # Column names

> A                    # Print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

`t(A)`
Transpose of A

$$\begin{bmatrix} 2 & 1 \\ 4 & 5 \\ 3 & 7 \end{bmatrix}$$

R data types

- R has a wide variety of data types including —
  - **Scalars**
  - **Vectors** (numerical, character, logical)
  - **Matrices**
  - **Data frames**
  - **Lists**
- We could use `class(objectName)` to find out which type an R object is.

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> d <- c(1,2,3,4)

> e <- c("red", "white", "red", NA)

> f <- c(TRUE,TRUE,TRUE,FALSE)

> mydata <- data.frame(d,e,f) # A data frame

> colnames(mydata) <- c("ID","Color","Passed") # Column names (header)

> mydata

  ID Color Passed

1  1   red   TRUE

2  2 white   TRUE

3  3   red   TRUE

4  4  <NA>  FALSE
```

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> mydata[1,2]
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> mydata[1,2]
[1] red
Levels: red white
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> mydata[1,2]
[1] red
Levels: red white
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

To avoid character vectors being converted to strings, add the option **`stringsAsFactors = FALSE`** when creating a data frame

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> mydata[1,2]
[1] red
Levels: red white
> nrow(mydata) # Number of rows
[1] 4
> ncol(mydata) # Number of columns
[1] 3
> dim(mydata) # Dimensions
[1] 4 3
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> str(mydata) # Get a summary of the data frame

'data.frame': 4 obs. of  3 variables:

$ ID    : num  1 2 3 4

$ Color : Factor w/ 2 levels "red","white": 1 2 1 NA

$ Passed: logi  TRUE TRUE TRUE FALSE
```

```
> mydata

  ID Color Passed

1  1   red   TRUE

2  2 white   TRUE

3  3   red   TRUE

4  4  <NA>  FALSE
```

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> str(mydata) # Get a summary of the data frame

'data.frame': 4 obs. of  3 variables:
 $ ID    : num  1 2 3 4
 $ Color : Factor w/ 2 levels "red","white": 1 2 1 NA
 $ Passed: logi  TRUE TRUE TRUE FALSE
> head(mydata) # Show first several rows
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Data frames

- A **data frame** is used for storing data tables. It is a list of **vectors of equal length**. Different columns can have **different classes** (numeric, character, factor, etc.).
- Example

```
> str(mydata) # Get a summary of the data frame

'data.frame': 4 obs. of  3 variables:
 $ ID    : num  1 2 3 4
 $ Color : Factor w/ 2 levels "red","white": 1 2 1 NA
 $ Passed: logi  TRUE TRUE TRUE FALSE
> head(mydata) # Show first several rows
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

- **`head()`** by default returns the first 6 rows (or all the rows if nrow <= 6)
- To show the first **i** rows, use **`head(mydata, n = i)`**

WARNING: COMPLETELY FOR BEGINNERS!

IN TODAY'S GUIDE...

1. What is R? Why R?
2. Installation and "Hello World!" in R
3. R data types – vectors, matrices and data frames
4. R operators and managing a data frame
5. I/O and basic graphs in R
6. Pop quiz

operators

- Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division

R operators

- Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or **	Exponentiation

R operators

- Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or **	Exponentiation
x %% y	x mod y (5 %% 2 is 1)

R operators

- Arithmetic operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or **	Exponentiation
x %% y	x mod y (5 %% 2 is 1)
x %/% y	Integer division (5 %/% 2 is 2)

Special values in 

Special values in 

- **`NA`**: Not available (missing); a logical constant

Special values in 

- **`NA`**: Not available (missing); a logical constant
  - Check via **`is.na(x)`**

Special values in 

- **`NA`**: Not available (missing); a logical constant
  - Check via **`is.na(x)`**
  - Different from the string **`"NA"`**!

Special values in 

- **`NA`**: Not available (missing); a logical constant
  - Check via **`is.na(x)`**
  - Different from the string **`"NA"`**!
- **`NaN`**: Not a number

```
> 0 / 0

[1] NaN
```

Special values in 

- **`NA`**: Not available (missing); a logical constant
  - Check via **`is.na(x)`**
  - Different from the string **`"NA"`**!
- **`NaN`**: Not a number

```
> 0 / 0
[1] NaN
```

- **`Inf`** (**`-Inf`**): Infinity

```
> 12 / 0
[1] Inf
```

Special values in 

- **`NA`**: Not available (missing); a logical constant
  - Check via **`is.na(x)`**
  - Different from the string **`"NA"`**!
- **`NaN`**: Not a number

```
> 0 / 0
[1] NaN
```

- **`Inf`** (**`-Inf`**): Infinity

```
> 12 / 0
[1] Inf
```

- **`NULL`**: The null object; undefined and of length 0

R operators

- Logical operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

R operators

- Logical operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to

R operators

- Logical operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to

R operators

- Logical operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x

R operators

- Logical operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x | y; x || y	x OR y (| is vectorized)

R operators

- Logical operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x | y; x || y	x OR y (| is vectorized)
x & y; x && y	x AND y (& is vectorized)

R operators

- Logical operators

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x | y; x || y	x OR y (| is vectorized)
x & y; x && y	x AND y (& is vectorized)
isTRUE(x)	Test if x is TRUE

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =
- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =
- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

Examples

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =
- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

Examples

```
> 4 + 20 / 17 %/% 3
[1]   ?
```

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =
- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

Examples

```
> 4 + 20 / 17 %/% 3
[1] 8
```

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =

Examples

```
> 4 + 20 / 17 %/% 3
[1] 8
```

```
> !FALSE | TRUE & FALSE
[1]   ?
```

- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =
- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

Examples

```
> 4 + 20 / 17 %/% 3
[1] 8
```

```
> !FALSE | TRUE & FALSE
[1] TRUE
```

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =

Examples

```
> 4 + 20 / 17 %/% 3
[1] 8
```

```
> !FALSE | TRUE & FALSE
[1] TRUE
```

```
> (!FALSE | TRUE) & FALSE
[1]    ?
```

- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

operator rules

- **Operator precedence**
  1. ^
  2. %% and %/%
  3. * and /
  4. + and -
  5. <, >, <=, >= and !=
  6. !
  7. & and &&
  8. | and ||
  9. <-
  10. =

Examples

```
> 4 + 20 / 17 %/% 3
[1] 8
```

```
> !FALSE | TRUE & FALSE
[1] TRUE
```

```
> (!FALSE | TRUE) & FALSE
[1] FALSE
```

- **Associativity**: Left to right, except for exponentiation and assignment
- **Parentheses** override order

Working with data frames:

Subsetting / Sampling

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

Working with data frames:

Subsetting / Sampling

```
> mydata[4,] # Select 4th row

  ID Color Passed

4  4  <NA>  FALSE
```

```
  > mydata

    ID Color Passed

  1  1   red   TRUE

  2  2 white   TRUE

  3  3   red   TRUE

  4  4  <NA>  FALSE
```

Working with data frames:

Subsetting / Sampling

```
> mydata[4,] # Select 4th row

  ID Color Passed

4  4  <NA>  FALSE
```

```
> mydata[,c(2:3)]

> # Select the 2nd and 3rd columns

  Color Passed

1   red   TRUE

2 white   TRUE

3   red   TRUE

4  <NA>  FALSE
```

```
> mydata

  ID Color Passed

1  1   red   TRUE

2  2 white   TRUE

3  3   red   TRUE

4  4  <NA>  FALSE
```

Working with data frames:

Subsetting / Sampling

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

```
> mydata[4,] # Select 4th row
  ID Color Passed
4  4  <NA>  FALSE
```

```
> mydata[,c(2:3)]
> # Select the 2nd and 3rd columns
  Color Passed
1   red   TRUE
2 white   TRUE
3   red   TRUE
4  <NA>  FALSE
```

```
> mydata$ID
> # Select the column named “ID”
[1] 1 2 3 4
```

Working with data frames:

Subsetting / Sampling

```
> mydata
  ID Color Passed
1  1   red    TRUE
2  2 white    TRUE
3  3   red    TRUE
4  4  <NA>   FALSE
```

```
> mydata[4,] # Select 4th row
  ID Color Passed
4  4  <NA>   FALSE
```

```
> mydata[,c(2:3)]
> # Select the 2nd and 3rd columns
  Color Passed
1   red    TRUE
2 white    TRUE
3   red    TRUE
4  <NA>   FALSE
```

```
> mydata$ID
> # Select the column named “ID”
[1] 1 2 3 4
```

```
> mydata[which(mydata$Passed & mydata$ID > 2), ]
> # Select observation(s) by value
  ID Color Passed
3  3   red    TRUE
```

Working with data frames:

Subsetting / Sampling

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

```
> mydata[4,] # Select 4th row
  ID Color Passed
4  4  <NA>  FALSE
```

```
> mydata[,c(2:3)]
> # Select the 2nd and 3rd columns
  Color Passed
1   red   TRUE
2 white   TRUE
3   red   TRUE
4  <NA>  FALSE
```

```
> mydata$ID
> # Select the column named "ID"
[1] 1 2 3 4
```

```
> set.seed(42) # Set random seed
> mydata[sample(1:nrow(mydata),2,replace=FALSE),]
> # Randomly sample 2 rows
  ID Color Passed
4  4  <NA>  FALSE
3  3   red   TRUE
```

```
> mydata[which(mydata$Passed & mydata$ID > 2), ]
> # Select observation(s) by value
  ID Color Passed
3  3   red   TRUE
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed
1  1   red   TRUE
2  2 white   TRUE
3  3   red   TRUE
4  4  <NA>  FALSE
```

```
> # Adding a new variable called weight
> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed weight
1  1   red   TRUE     65
2  2 white   TRUE     70
3  3   red   TRUE     75
4  4  <NA>  FALSE     80
```

```
> # Adding a new variable called weight
> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed weight
1  1   red   TRUE     65
2  2 white   TRUE     70
3  3   red   TRUE     75
4  4  <NA>  FALSE     80
```

```
> # Adding a new variable called weight

> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

```
> # Adding a new variable called height

> mydata$height <- rep(170, 4)
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed weight height
1  1   red   TRUE     65    170
2  2 white   TRUE     70    170
3  3   red   TRUE     75    170
4  4  <NA>  FALSE     80    170
```

```
> # Adding a new variable called weight
> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

```
> # Adding a new variable called height
> mydata$height <- rep(170, 4)
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed weight height
1  1   red   TRUE     65    170
2  2 white   TRUE     70    170
3  3   red   TRUE     75    170
4  4  <NA>  FALSE     80    170
```

```
> # Adding a new variable called weight
> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

```
> # Adding a new variable called height
> mydata$height <- rep(170, 4)
```

```
> # Adding a new variable calculated based on weight and height
> mydata$bmi <- mydata$weight / (mydata$height/100)^2
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed weight height      bmi
1  1   red   TRUE     65    170 22.49135
2  2 white   TRUE     70    170 24.22145
3  3   red   TRUE     75    170 25.95156
4  4  <NA>  FALSE     80    170 27.68166
```

```
> # Adding a new variable called weight
> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

```
> # Adding a new variable called height
> mydata$height <- rep(170, 4)
```

```
> # Adding a new variable calculated based on weight and height
> mydata$bmi <- mydata$weight / (mydata$height/100)^2
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed weight height      bmi
1  1   red   TRUE     65    170 22.49135
2  2 white   TRUE     70    170 24.22145
3  3   red   TRUE     75    170 25.95156
4  4  <NA>  FALSE     80    170 27.68166
```

```
> # Adding a new variable called weight
> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

```
> # Adding a new variable called height
> mydata$height <- rep(170, 4)
```

```
> # Adding a new variable calculated based on weight and height
> mydata$bmi <- mydata$weight / (mydata$height/100)^2
```

```
> # Adding a new logical variable based on bmi
> mydata$overwt <- mydata$bmi >= 25
```

Working with data frames:

Adding variables

```
> mydata
  ID Color Passed weight height      bmi overwt
1  1   red   TRUE     65    170 22.49135  FALSE
2  2 white   TRUE     70    170 24.22145  FALSE
3  3   red   TRUE     75    170 25.95156   TRUE
4  4  <NA>  FALSE     80    170 27.68166   TRUE
```

```
> # Adding a new variable called weight
> mydata$weight <- seq(from = 65, to = 80, by = 5)
```

```
> # Adding a new variable called height
> mydata$height <- rep(170, 4)
```

```
> # Adding a new variable calculated based on weight and height
> mydata$bmi <- mydata$weight / (mydata$height/100)^2
```

```
> # Adding a new logical variable based on bmi
> mydata$overwt <- mydata$bmi >= 25
```

Working with data frames:

Dropping variables

```
> mydata

  ID Color Passed weight height      bmi overwt
1  1   red   TRUE     65    170 22.49135  FALSE
2  2 white   TRUE     70    170 24.22145  FALSE
3  3   red   TRUE     75    170 25.95156   TRUE
4  4  <NA>  FALSE     80    170 27.68166   TRUE
```

Working with data frames:

Dropping variables

```
> mydata

  ID Color Passed weight height      bmi overwt
1  1   red   TRUE     65    170 22.49135  FALSE
2  2 white   TRUE     70    170 24.22145  FALSE
3  3   red   TRUE     75    170 25.95156   TRUE
4  4  <NA>  FALSE     80    170 27.68166   TRUE
```

```
> # Exclude variables ID, Color

> myvars <- colnames(mydata) %in% c("ID", "Color")

> newdata <- mydata[!myvars]

> newdata

  Passed weight height      bmi overwt
1   TRUE     65    170 22.49135  FALSE
2   TRUE     70    170 24.22145  FALSE
3   TRUE     75    170 25.95156   TRUE
4  FALSE     80    170 27.68166   TRUE
```

Working with data frames:

Dropping variables

```
> mydata
  ID Color Passed weight height      bmi overwt
1  1   red   TRUE     65    170 22.49135  FALSE
2  2 white   TRUE     70    170 24.22145  FALSE
3  3   red   TRUE     75    170 25.95156   TRUE
4  4  <NA>  FALSE     80    170 27.68166   TRUE
```

```
> # Exclude variables ID, Color

> myvars <- colnames(mydata) %in% c("ID", "Color")

> newdata <- mydata[!myvars]

> newdata
  Passed weight height      bmi overwt
1   TRUE     65    170 22.49135  FALSE
2   TRUE     70    170 24.22145  FALSE
3   TRUE     75    170 25.95156   TRUE
4  FALSE     80    170 27.68166   TRUE
```

```
> # Exclude 1st and 3rd variables

> newdata2 <- mydata[c(-1,-3)]

> newdata2
  Color weight height      bmi overwt
1   red     65    170 22.49135  FALSE
2 white     70    170 24.22145  FALSE
3   red     75    170 25.95156   TRUE
4  <NA>     80    170 27.68166   TRUE
```

Working with data frames:

Dropping variables

```
> mydata
  ID Color Passed weight height      bmi overwt
1  1   red   TRUE     65    170 22.49135  FALSE
2  2 white   TRUE     70    170 24.22145  FALSE
3  3   red   TRUE     75    170 25.95156   TRUE
4  4  <NA>  FALSE     80    170 27.68166   TRUE
```

```
> # Exclude variables ID, Color

> myvars <- colnames(mydata) %in% c("ID", "Color")

> newdata <- mydata[!myvars]

> newdata
  Passed weight height      bmi overwt
1   TRUE     65    170 22.49135  FALSE
2   TRUE     70    170 24.22145  FALSE
3   TRUE     75    170 25.95156   TRUE
4  FALSE     80    170 27.68166   TRUE
```

```
> # Exclude 1st and 3rd variables

> newdata2 <- mydata[c(-1,-3)]

> newdata2
  Color weight height      bmi overwt
1   red     65    170 22.49135  FALSE
2 white     70    170 24.22145  FALSE
3   red     75    170 25.95156   TRUE
4  <NA>     80    170 27.68166   TRUE
```

```
> # Delete variable Color

> mydata$Color <- NULL
```

Working with data frames:

Dropping variables

```
> mydata
  ID Passed weight height      bmi overwt
1  1   TRUE     65    170 22.49135  FALSE
2  2   TRUE     70    170 24.22145  FALSE
3  3   TRUE     75    170 25.95156   TRUE
4  4  FALSE     80    170 27.68166   TRUE
```

```
> # Exclude variables ID, Color
> myvars <- colnames(mydata) %in% c("ID", "Color")
> newdata <- mydata[!myvars]
> newdata
  Passed weight height      bmi overwt
1   TRUE     65    170 22.49135  FALSE
2   TRUE     70    170 24.22145  FALSE
3   TRUE     75    170 25.95156   TRUE
4  FALSE     80    170 27.68166   TRUE
```

```
> # Delete variable Color
> mydata$Color <- NULL
```

WARNING
This would directly delete from the data frame mydata!

```
> # Exclude 1st and 3rd variables
> newdata2 <- mydata[c(-1,-3)]
> newdata2
  Color weight height      bmi overwt
1   red     65    170 22.49135  FALSE
2 white     70    170 24.22145  FALSE
3   red     75    170 25.95156   TRUE
<NA>      80    170 27.68166   TRUE
```

Working with data frames:

Sorting by variables

```
> mydata
  ID Passed weight height      bmi overwt
1  1   TRUE     65    170 22.49135  FALSE
2  2   TRUE     70    170 24.22145  FALSE
3  3   TRUE     75    170 25.95156   TRUE
4  4  FALSE     80    170 27.68166   TRUE
```

- To sort a data frame in R, use the **`order( )`** function.
  - By default, sorting is ascending.
  - Prepend the sorting variable by a minus sign to indicate descending order.

Working with data frames:

Sorting by variables

```
> mydata
  ID Passed weight height      bmi overwt
1  1   TRUE     65    170 22.49135  FALSE
2  2   TRUE     70    170 24.22145  FALSE
3  3   TRUE     75    170 25.95156   TRUE
4  4  FALSE     80    170 27.68166   TRUE
```

```
> # Sort by descending weight and ascending height
> sortedData <- mydata[order(-mydata$weight, mydata$height),]
```

Working with data frames:

Sorting by variables

```
> mydata
  ID Passed weight height      bmi overwt
1  1   TRUE     65    170 22.49135  FALSE
2  2   TRUE     70    170 24.22145  FALSE
3  3   TRUE     75    170 25.95156   TRUE
4  4  FALSE     80    170 27.68166   TRUE
```

```
> # Sort by descending weight and ascending height
> sortedData <- mydata[order(-mydata$weight, mydata$height),]
```

```
> sortedData
  ID Passed weight height      bmi overwt
4  4  FALSE     80    170 27.68166   TRUE
3  3   TRUE     75    170 25.95156   TRUE
2  2   TRUE     70    170 24.22145  FALSE
1  1   TRUE     65    170 22.49135  FALSE
```

WARNING: COMPLETELY FOR BEGINNERS!

IN TODAY'S GUIDE...

1. What is R? Why R?
2. Installation and "Hello World!" in R
3. R data types – vectors, matrices and data frames
4. R operators and managing a data frame
5. I/O and basic graphs in R
6. Pop quiz

Exporting data from R

- **write.table()**: print data frame to text file

```
# First row contains variable names; do not print row names

# Delimiter is tab (“\t”)

# Do not double quote character / factor variables

write.table(mydata, file = “datFile.txt”, sep = “\t”, quote =
FALSE, row.names = FALSE, col.names = TRUE)
```

Exporting data from R

- **`write.table()`**: print data frame to text file

```
# First row contains variable names; do not print row names

# Delimiter is tab (“\t”)

# Do not double quote character / factor variables

write.table(mydata, file = “datFile.txt”, sep = “\t”, quote =
FALSE, row.names = FALSE, col.names = TRUE)
```

- **`save()`**: Write R objects to an external file

```
save(file = “savedData.RData”, list = ls())
```

Exporting data from R

- **write.table()**: print data frame to text file

```
# First row contains variable names; do not print row names

# Delimiter is tab (“\t”)

# Do not double quote character / factor variables

write.table(mydata, file = “datFile.txt”, sep = “\t”, quote =
FALSE, row.names = FALSE, col.names = TRUE)
```

- **save()**: Write R objects to an external file

```
save(file = “savedData.RData”, list = ls())
```

- Other functions for exporting data

Exporting data from 

- **write.table()**: print data frame to text file

```
# First row contains variable names; do not print row names

# Delimiter is tab (“\t”)

# Do not double quote character / factor variables

write.table(mydata, file = “datFile.txt”, sep = “\t”, quote =
FALSE, row.names = FALSE, col.names = TRUE)
```

- **save()**: Write R objects to an external file

```
save(file = “savedData.RData”, list = ls())
```

- Other functions for exporting data
  - **write.csv()**

Exporting data from R

- **`write.table()`**: print data frame to text file

```
# First row contains variable names; do not print row names

# Delimiter is tab (“\t”)

# Do not double quote character / factor variables

write.table(mydata, file = “datFile.txt”, sep = “\t”, quote =
FALSE, row.names = FALSE, col.names = TRUE)
```

- **`save()`**: Write R objects to an external file

```
save(file = “savedData.RData”, list = ls())
```

- Other functions for exporting data
  - **`write.csv()`**
  - **`write.xlsx()`** in the **`xlsx`** package

Exporting data from R

- **`write.table()`**: print data frame to text file

```
# First row contains variable names; do not print row names

# Delimiter is tab (“\t”)

# Do not double quote character / factor variables

write.table(mydata, file = “datFile.txt”, sep = “\t”, quote =
FALSE, row.names = FALSE, col.names = TRUE)
```

- **`save()`**: Write R objects to an external file

```
save(file = “savedData.RData”, list = ls())
```

- Other functions for exporting data
  - **`write.csv()`**
  - **`write.xlsx()`** in the **`xlsx`** package
  - ***`?<function_name>`*** and read their manual

Importing data into R

- **`read.table()`**: read a text file in table format and create a data frame from it

```
# First row contains variable names

# Delimiter is tab ("\t")

read.table(file = "datFile.txt", sep = "\t", header = TRUE)
```

- **`load()`**: Reload datasets written with the function "save"

```
load("savedData.RData")
```

- Other functions for importing data
  - **`read.csv()`**
  - **`read.xlsx()`** in the **`xlsx`** package
  - **`?<function_name>`** and read their manual

Practical: Simple visualisation in R

- There are actually a lot of built-in data sets in R.
  - Type `library(help = “datasets")` to see what are they…

Practical: Simple visualisation in R

- There are actually a lot of built-in data sets in R.
  - Type `library(help = "datasets")` to see what are they…

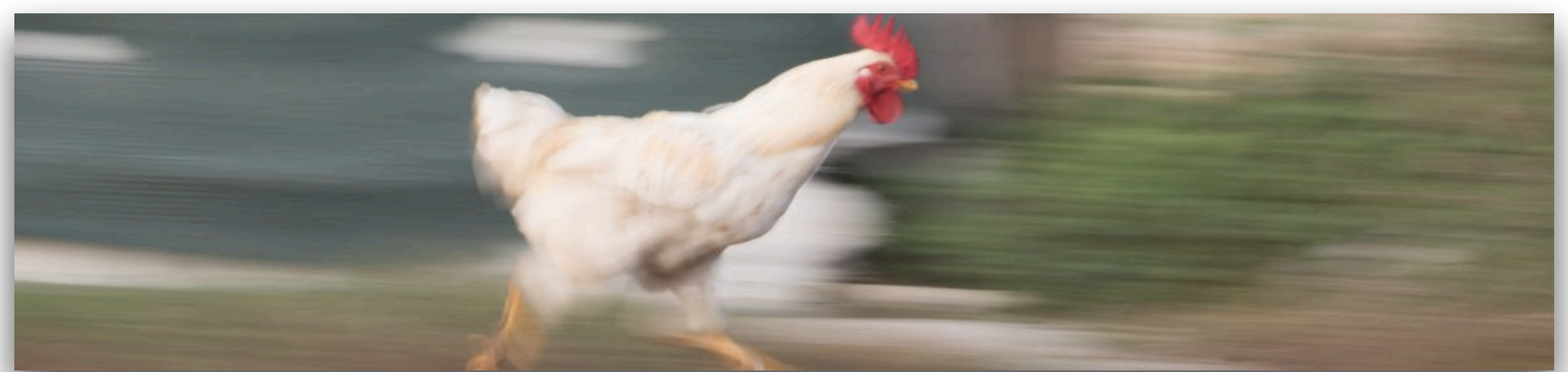
```
AirPassengers            Monthly Airline Passenger Numbers 1949-1960
BJsales                  Sales Data with Leading Indicator
BOD                      Biochemical Oxygen Demand
CO2                      Carbon Dioxide Uptake in Grass Plants
ChickWeight              Weight versus age of chicks on different diets
DNase                    Elisa assay of DNase
EuStockMarkets           Daily Closing Prices of Major European Stock
                         Indices, 1991-1998
Formaldehyde             Determination of Formaldehyde
```

Practical: Simple visualisation in R

- There are actually a lot of built-in data sets in R.
  - Type `library(help = "datasets")` to see what are they…

```
AirPassengers              Monthly Airline Passenger Numbers 1949-1960
BJsales                    Sales Data with Leading Indicator
BOD                        Biochemical Oxygen Demand
CO2                        Carbon Dioxide Uptake in Grass Plants
ChickWeight                Weight versus age of chicks on different diets
DNase                      Elisa assay of DNase
EuStockMarkets             Daily Closing Prices of Major European Stock
                           Indices, 1991-1998
Formaldehyde               Determination of Formaldehyde
```

- Since by Chinese zodiac this year is year of the rooster, we would try to deal with the **`ChickWeight`** data set.



Practical: Like Regular Chickens

- The data set is already available for use when we start R.
- First few lines of **`str(ChickWeight)`** —

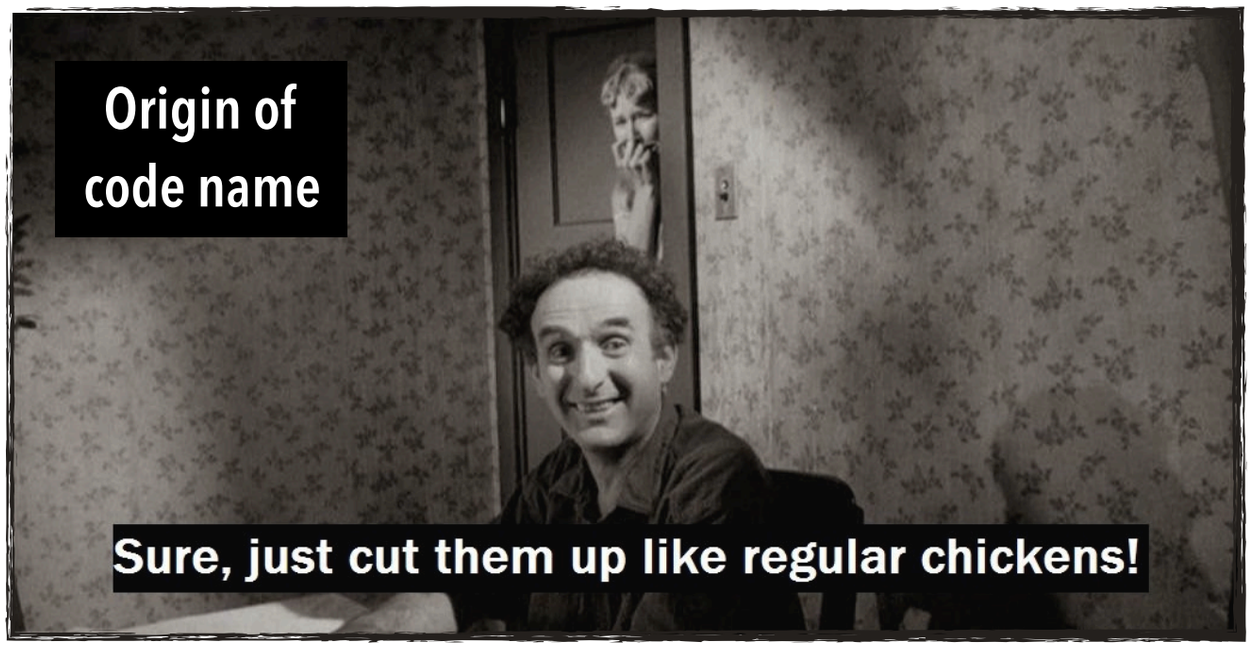
```
Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and
'data.frame':  578 obs. of  4 variables:

 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...

 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...

 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15
15 15 15 15 15 ...

 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
```



Practical: Like Regular Chickens

- The data set is already available for use when we start R.
- First few lines of **`str(ChickWeight)`** —

```
Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and
'data.frame':  578 obs. of  4 variables:
 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15
15 15 15 15 15 ...
 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 …
```

- Questions we could ask —

Practical: Like Regular Chickens

- The data set is already available for use when we start R.
- First few lines of **`str(ChickWeight)`** —

```
Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and
'data.frame':  578 obs. of  4 variables:

 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...

 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...

 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15
15 15 15 15 15 ...

 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 …
```

- Questions we could ask —
  - How are the chicken weights at time 0 **distributed**?

Practical: Like Regular Chickens

- The data set is already available for use when we start R.
- First few lines of **`str(ChickWeight)`** —

```
Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and
'data.frame':  578 obs. of  4 variables:
 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15
15 15 15 15 15 ...
 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 …
```

- Questions we could ask —
  - How are the chicken weights at time 0 **distributed**?
  - How do the chicken weights generally **change over time**?

Practical: Like Regular Chickens

- The data set is already available for use when we start R.
- First few lines of **`str(ChickWeight)`** —

```
Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and
'data.frame':  578 obs. of  4 variables:

 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...

 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...

 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15
15 15 15 15 15 ...

 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 …
```

- Questions we could ask —
  - How are the chicken weights at time 0 **distributed**?
  - How do the chicken weights generally **change over time**?
  - Is there a **difference** in the average chicken weights when they have different diets?

Practical: Like Regular Chickens

- The data set is already available for use when we start R.
- First few lines of **`str(ChickWeight)`** —

```
Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and
'data.frame':  578 obs. of  4 variables:
 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15
15 15 15 15 15 ...
 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 …
```

- Questions we could ask —
  - How are the chicken weights at time 0 **distributed**?
  - How do the chicken weights generally **change over time**?
  - Is there a **difference** in the average chicken weights when they have different diets?

- ***Explore by data visualisation!***

Practical: Like Regular Chickens

- Questions we could ask —
  - How are the chicken weights at time 0 **distributed**?

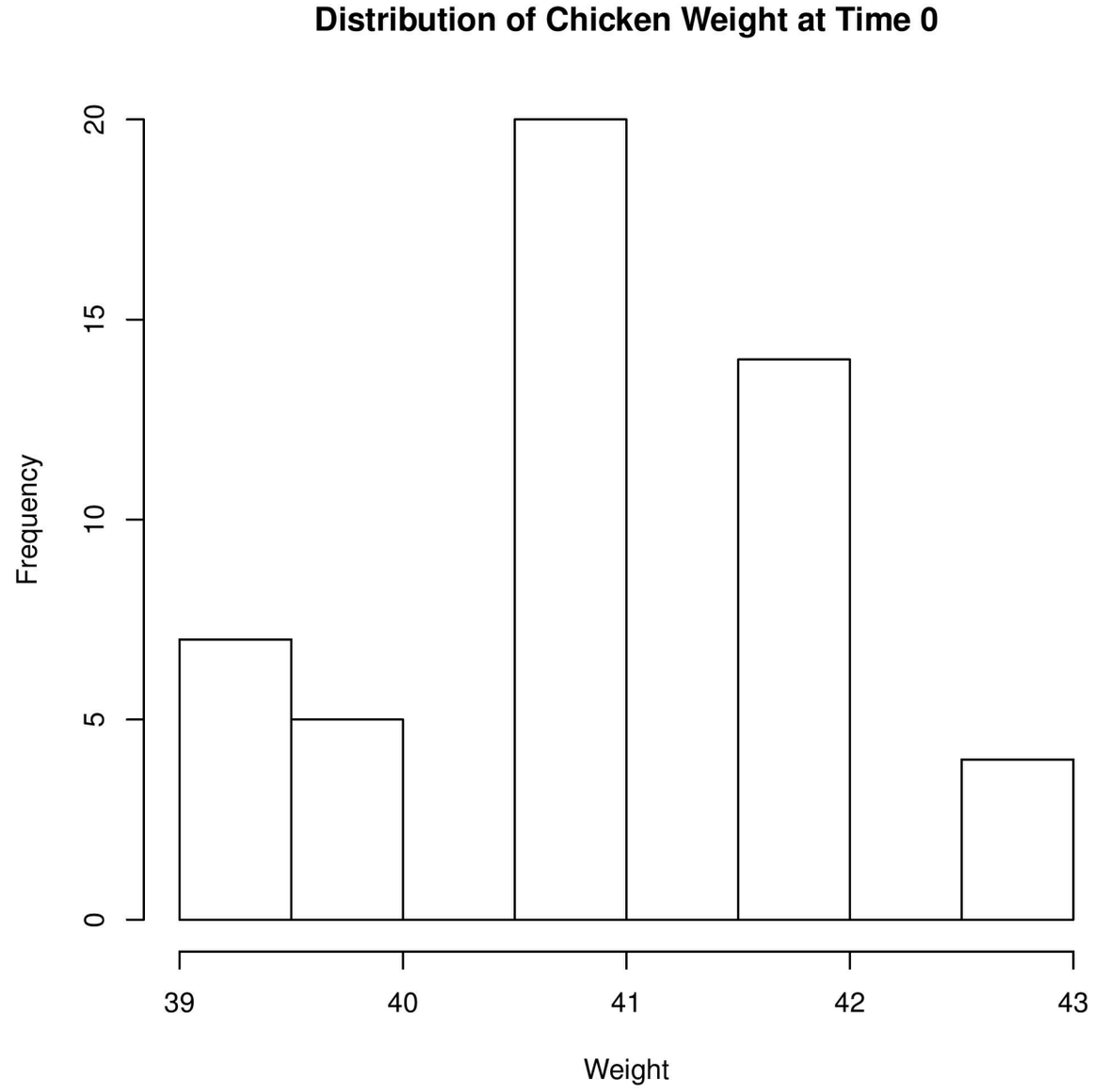
Practical: Like Regular Chickens

- Questions we could ask —
  - How are the chicken weights at time 0 **distributed**?
- Draw a histogram!

Practical: Like Regular Chickens

- Questions we could ask —
  - How are the chicken weights at time 0 **distributed**?
- Draw a histogram!

```
hist(ChickWeight$weight[
ChickWeight$Time == 0],
main = "Distribution of
Chicken Weight at Time
0", xlab = "Weight")
```



Practical: Like Regular Chickens

- Questions we could ask —
  - How do the chicken weights generally **change over time**?

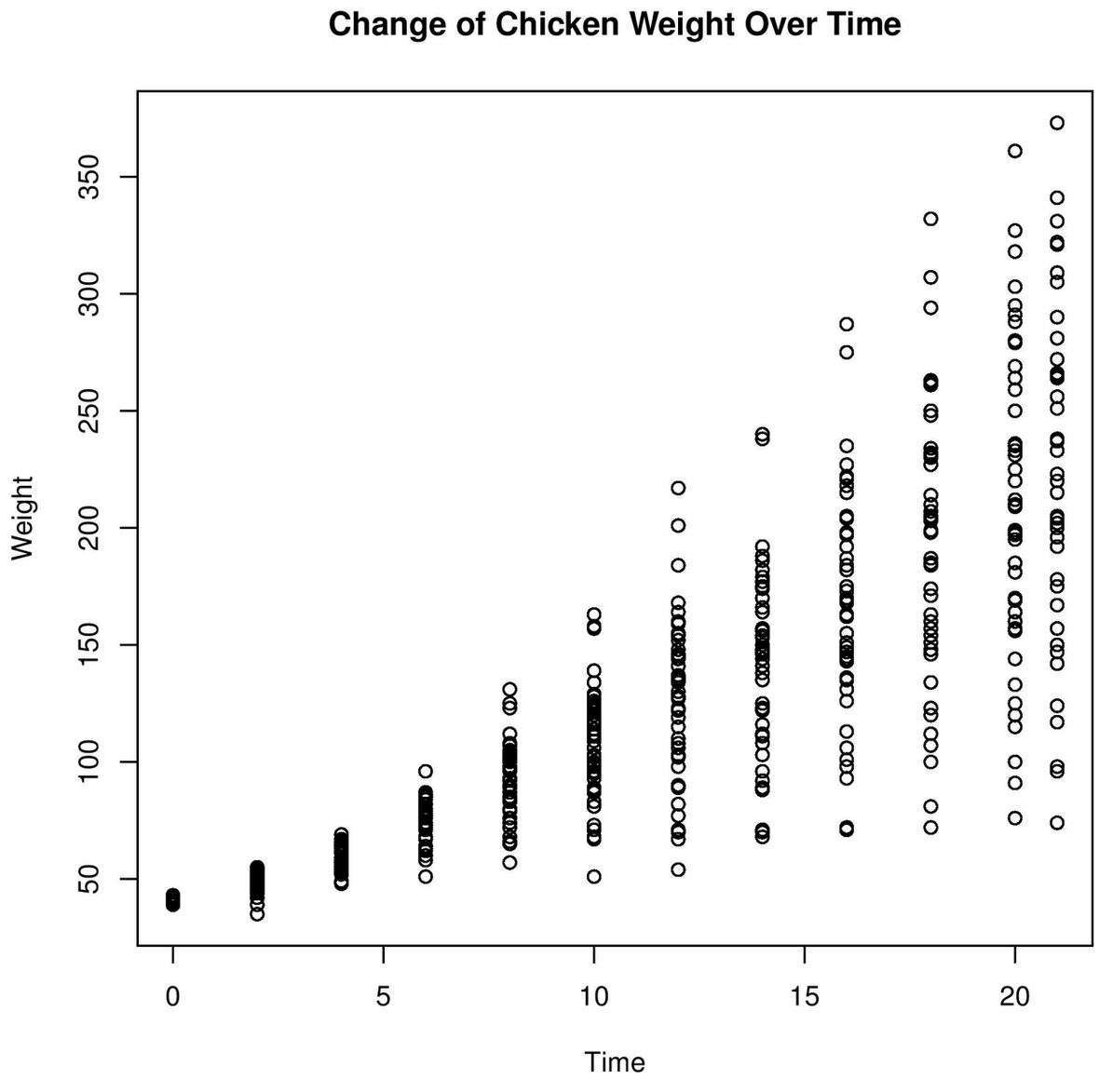
Practical: Like Regular Chickens

- Questions we could ask —
  - How do the chicken weights generally **change over time**?
- Draw a scatterplot!

Practical: Like Regular Chickens

- Questions we could ask —
  - How do the chicken weights generally **change over time**?
- Draw a scatterplot!

```
plot(ChickWeight$Time,
ChickWeight$weight, main
= "Change of Chicken
Weight Over Time", xlab =
"Time", ylab = "Weight")
```



Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?

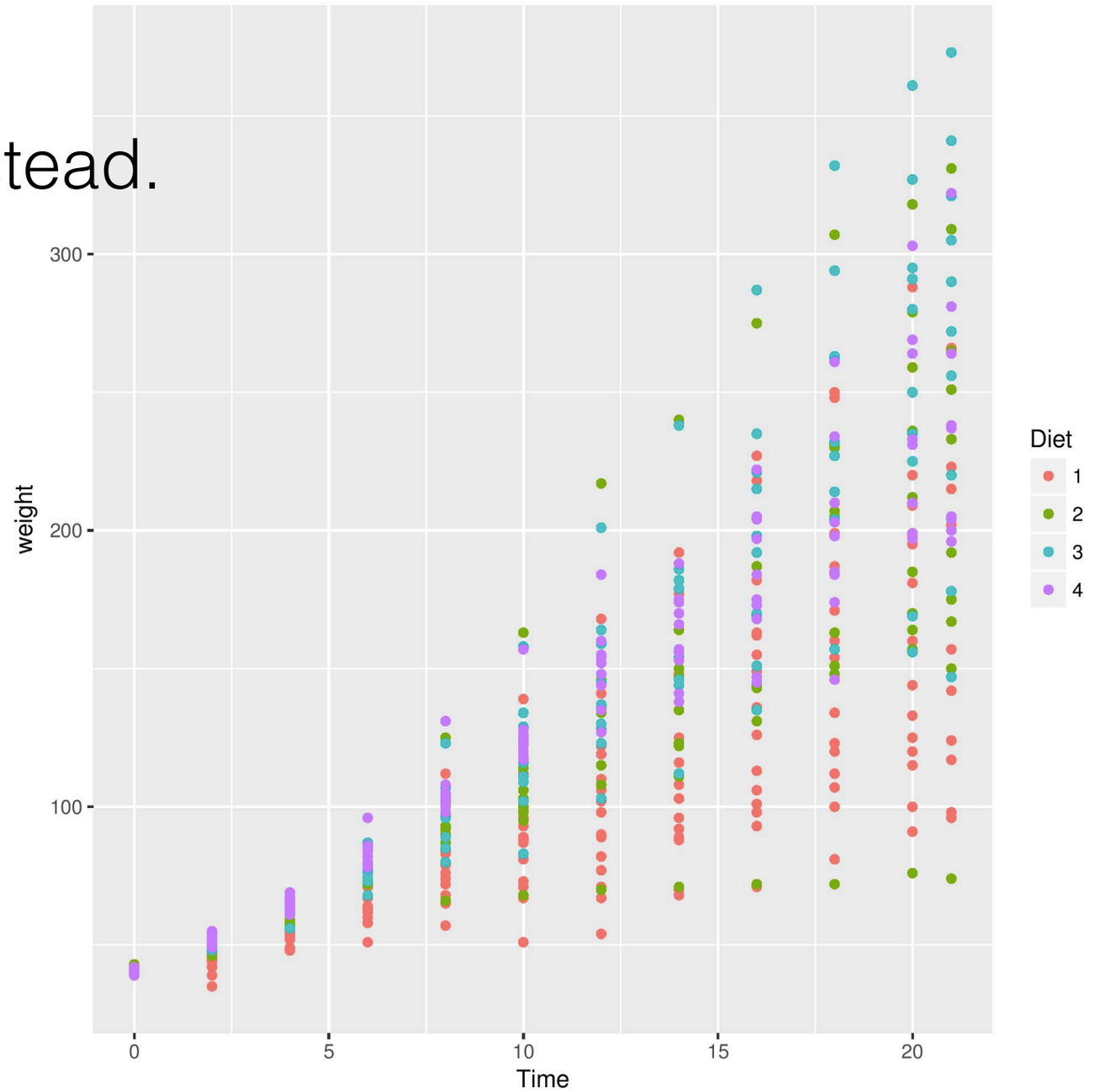
Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?
- This time we use `ggplot2` instead.

Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?
- This time we use `ggplot2` instead.

```
> library("ggplot2")
> qplot(Time, weight,
data = ChickWeight,
colour = Diet)
```

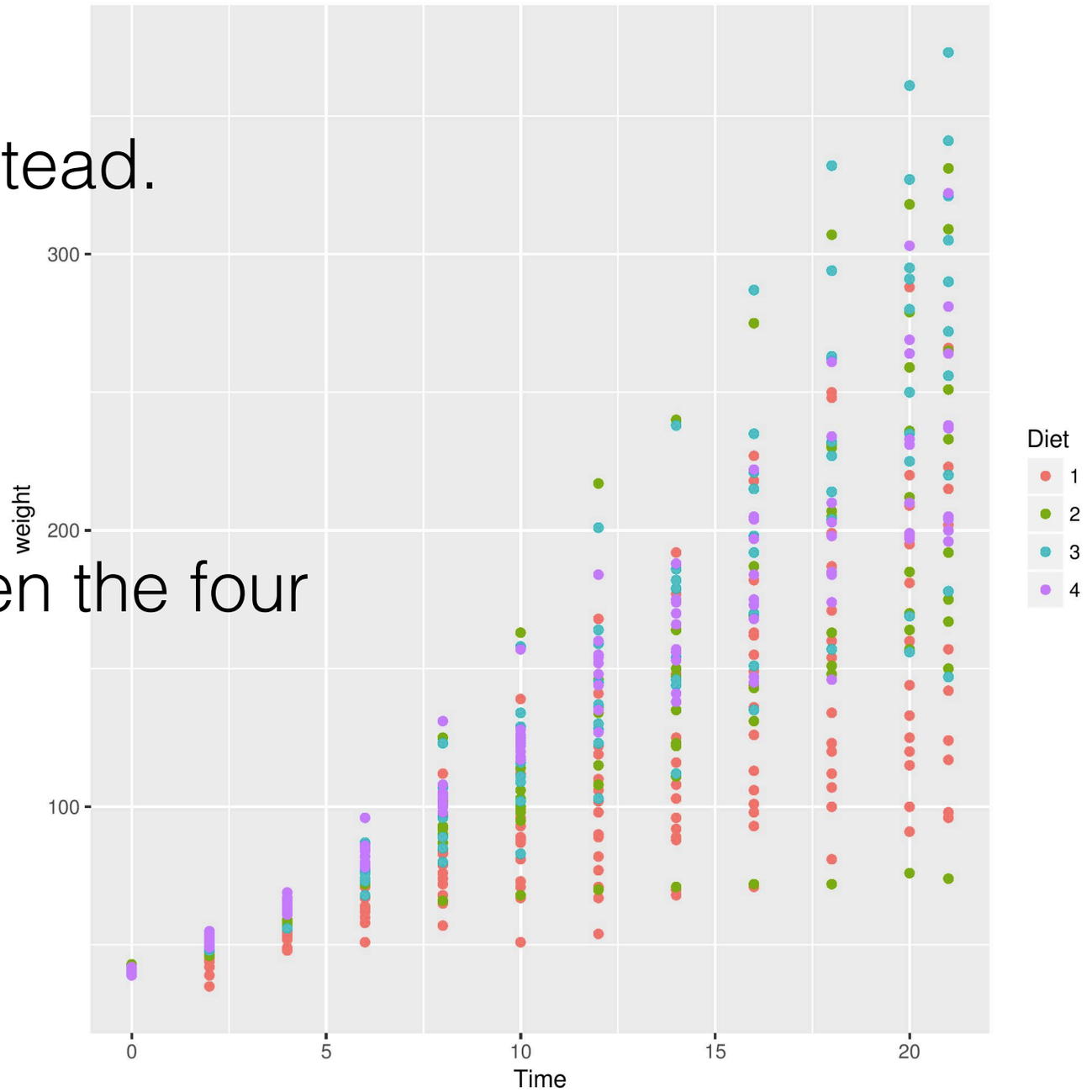


Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?
- This time we use `ggplot2` instead.

```
> library("ggplot2")
> qplot(Time, weight,
data = ChickWeight,
colour = Diet)
```

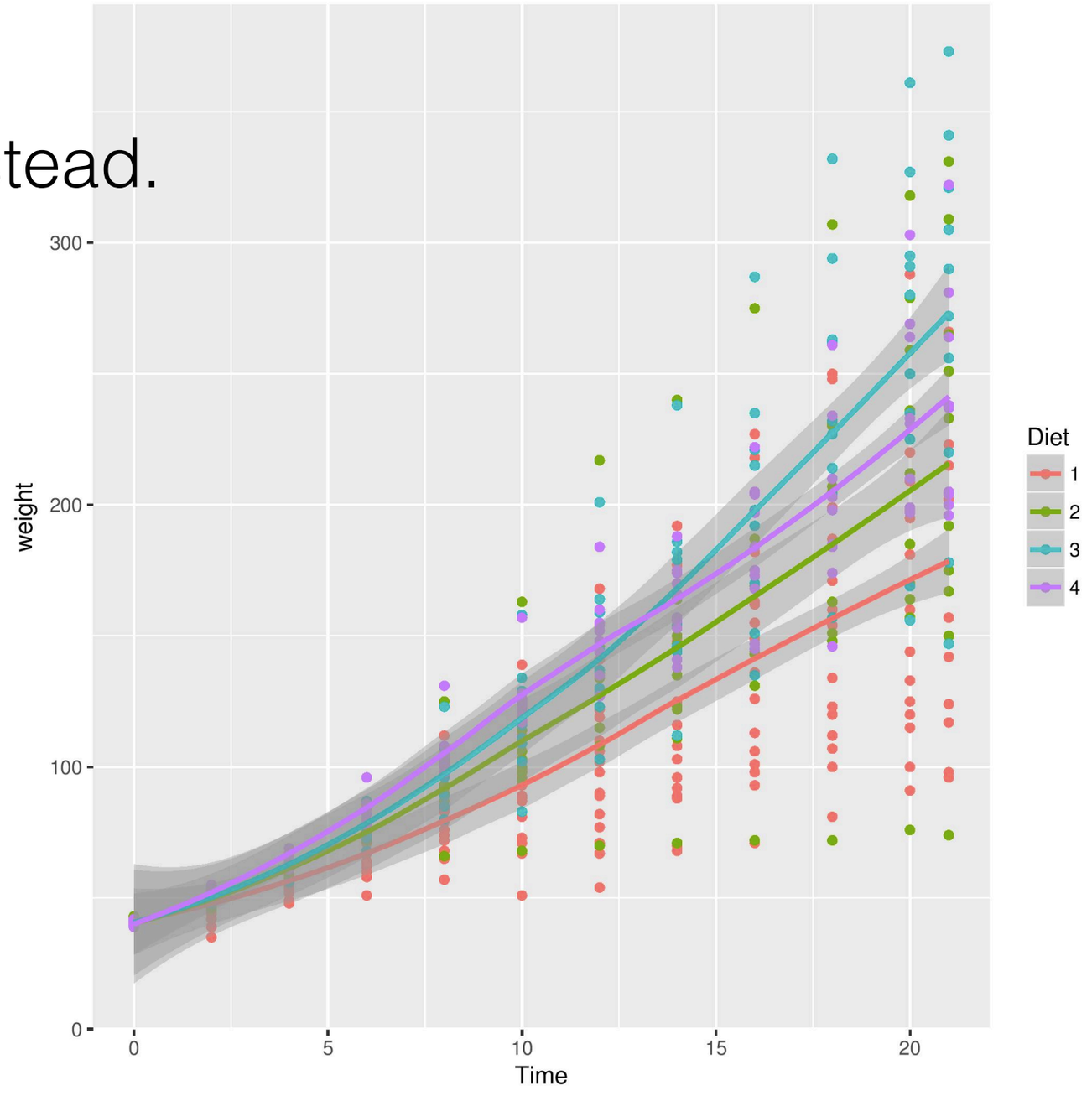
- It is hard to distinguish between the four diet groups.



Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?
- This time we use `ggplot2` instead.

```
> library("ggplot2")
> qplot(Time, weight,
data = ChickWeight,
colour = Diet, geom =
c("point", "smooth"))
```

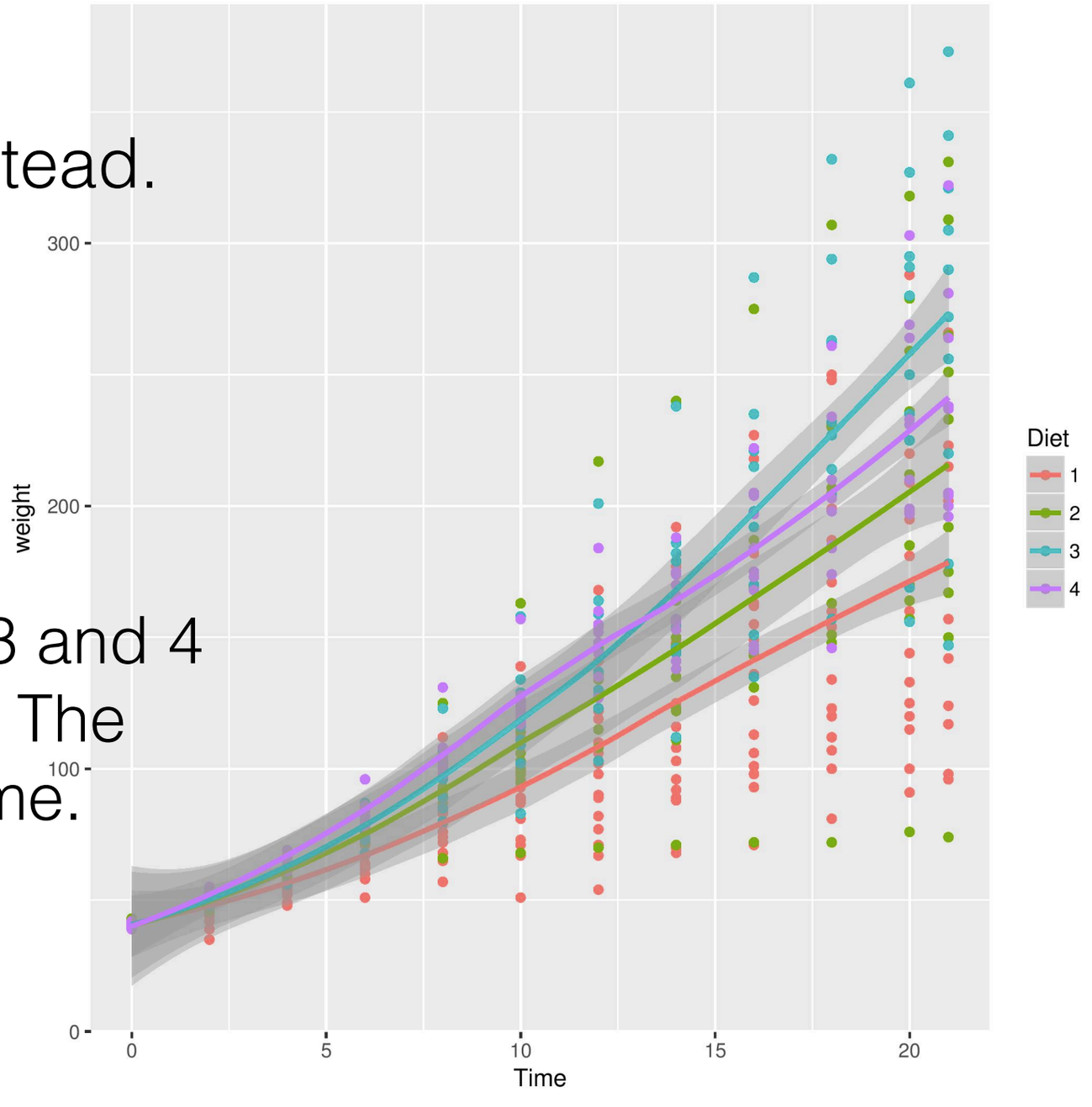


Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?
- This time we use `ggplot2` instead.

```
> library("ggplot2")
> qplot(Time, weight,
data = ChickWeight,
colour = Diet, geom =
c("point", "smooth"))
```

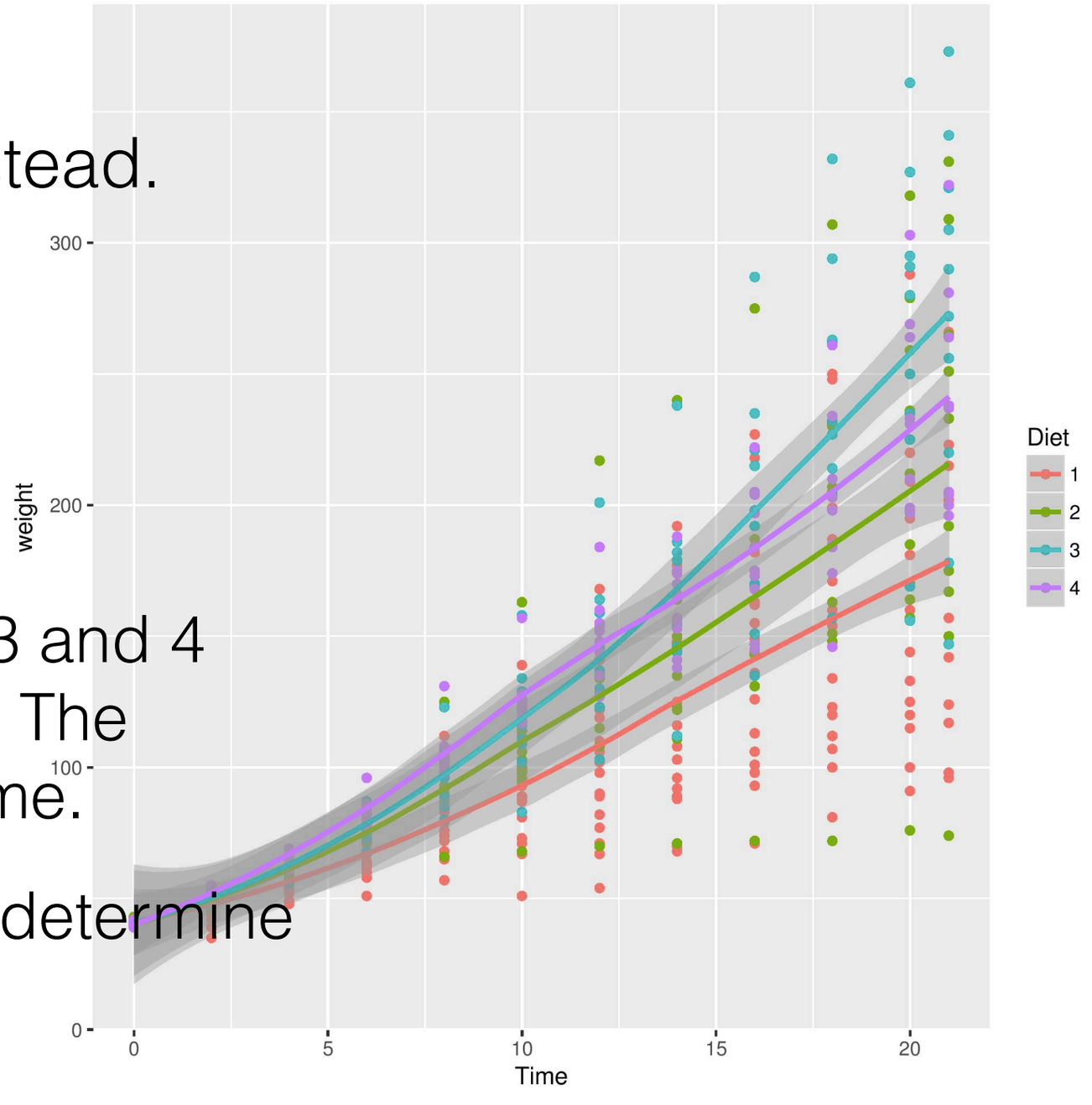
- It seems that on average, diets 3 and 4 result in heavier chicken weight. The difference grows greater over time.



Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?
- This time we use `ggplot2` instead.

```
> library("ggplot2")
> qplot(Time, weight,
data = ChickWeight,
colour = Diet, geom =
c("point", "smooth"))
```



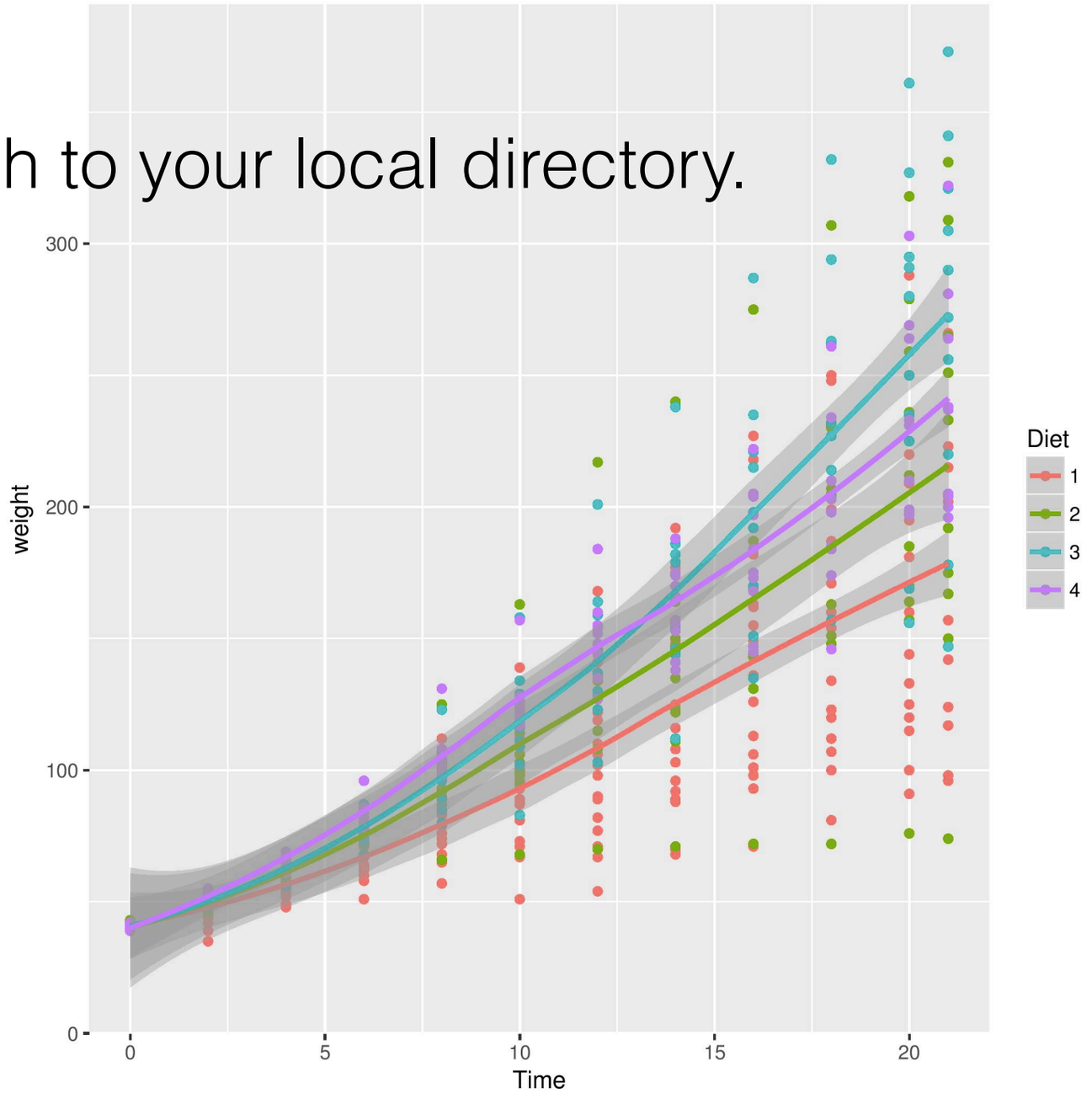
- It seems that on average, diets 3 and 4 result in heavier chicken weight. The difference grows greater over time.
- Statistical analysis is needed to determine whether this is truly significant.

Practical: Like Regular Chickens

- Questions we could ask —
  - Is there a **difference** in the average chicken weights when they have different diets?
- You could also save your graph to your local directory.

```
> library(“ggplot2”)
> pdf(“LRCvis.pdf”)
> qplot(Time, weight, data
= ChickWeight, colour =
Diet, geom = c("point",
“smooth"))
> dev.off()
```

- Your plot would then be saved as ./LRCvis.pdf.



WARNING: COMPLETELY FOR BEGINNERS!

IN TODAY'S GUIDE...

1. What is R? Why R?
2. Installation and "Hello World!" in R
3. R data types – vectors, matrices and data frames
4. R operators and managing a data frame
5. I/O and basic graphs in R
6. Pop quiz

References

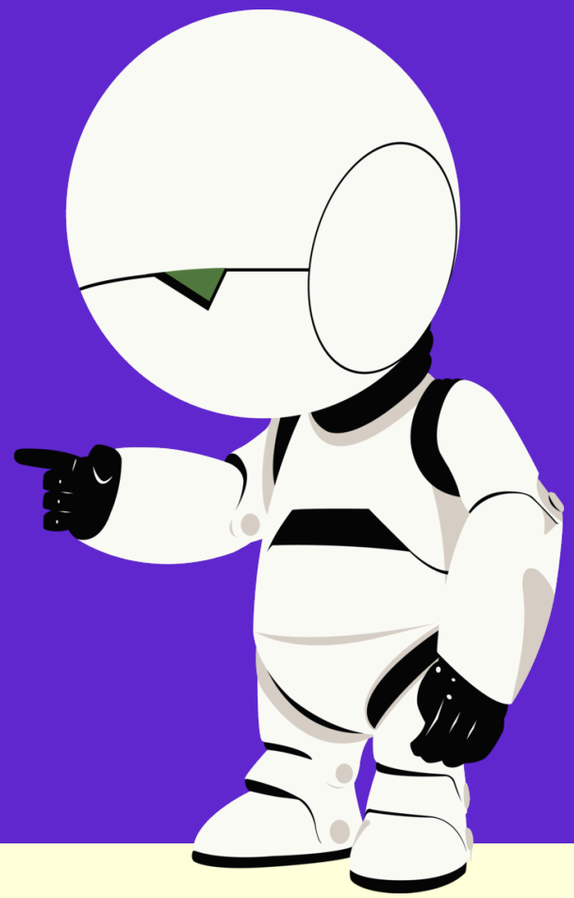
- Many ideas were generated when visiting the following websites / materials.
- Also some of the used code snippets were modified based on the demo codes there.
  - The R manual.
  - UC Berkeley STAT133 lecture notes.
  - http://stackoverflow.com/
  - http://www.statmethods.net/
  - http://arrgh.tim-smith.us/
  - http://www.r-tutor.com/r-introduction/matrix

Image sources

- R logo. https://www.r-project.org/logo/Rlogo.png
- Hitchhiker's thumb. http://i1.kym-cdn.com/entries/icons/facebook/000/018/991/HitchHikersGuideBlackSS.jpg
- Don't panic. http://geekifyinc.com/wp-content/uploads/2014/04/IMG_0333-1280.jpg
- Ross Ihaka. http://www.stats.org.nz/Newsletter69/images/Ross_Pickering_Medal.jpg
- Robert Gentleman. https://www.fredhutch.org/en/news/center-news/2009/05/Gentlemen-presents-lecture/_jcr_content/articletext/textimage/image.img.jpg/1322528033362.jpg
- Richard Stallman (left). https://upload.wikimedia.org/wikipedia/commons/f/f3/Richard_Stallman_by_Anders_Brenna_01.jpg
- Richard Stallman (right). http://i1-news.softpedia-static.com/images/news2/Richard-Stallman-Says-He-Created-GNU-Which-Is-Called-Often-Linux-482416-2.jpg
- GNU logo. https://www.gnu.org/graphics/empowered-by-gnu.svg
- Copyleft. https://upload.wikimedia.org/wikipedia/commons/thumb/8/8b/Copyleft.svg/1024px-Copyleft.svg.png
- Statistics clipart. http://images.clipartpanda.com/statistics-clipart-statistics.png
- All ggplot2 sample graphs from: http://www.r-graph-gallery.com/portfolio/ggplot2-package/
- Google trends graph of statistical software. Screenshot of https://goo.gl/jyOViq
- RStudio screenshot. http://1.bp.blogspot.com/-BCAWGBV9ze4/USjitphaQoI/AAAAAAAAMzI/-hlfvxFfbVg/s1600/Screenshot+from+2013-02-23+09%3A38%3A38.png
- Running rooster. https://notadinnerblog.files.wordpress.com/2016/09/cropped-avian_influenza_running_chicken.jpg
- "Sure, just cut them up like regular chickens". Screenshot from *Eraserhead* by David Lynch. http://www.funnyjunk.com/Just+cut+them+up+like+regular+chickens/hdgifs/5674895#1486a9_5674451
- Marvin. http://pre04.deviantart.net/cd13/th/pre/f/2014/342/c/8/marvin_the_paranoid_android_by_wheelmaker42-d896526.png

THANK YOU.

ANY QUESTIONS?



YIMING LI, 15 MAR 2017