Intro to Data Visualization and Statistics in R

Session #1

Genome Institute of Singapore

26th September 2019

OVERVIEW

Why learn how to code?

- Biology is becoming increasingly more data intensive
- Bioinformaticians are becoming increasingly rare
- Bioinformatician may not understand you.
  You may not understand them.
- Able to read and understand codes
- Automate/speed up some of your basic analysis
- Many bioinformaticians in GIS to learn from

The course objectives

Objectives:

1. Increase awareness and confidence
2. Build a network of learners
3. Build foundation for more specialized courses later
4. Identify useful resources

What we won't cover in this workshop

- analysis that require specialized preprocessing (e.g. transcriptomics, variant analysis)
- bioinformatics (e.g. promoter sequence, annotation mapping)
- machine learning, AI, text mining, etc

Your team

and Narasimhan Kothandaraman, Raden Kendarsari, Michelle Goh

R & RSTUDIO

What is R?

- **Language** and environment for statistical computing and graphics.
- Created by **R**oss Ihaka and **R**obert Gentleman[1] from University of Auckland, New Zealand.
- Initial release in 1995.
- Easily extended via **packages** from CRAN or Bioconductor etc.

[1] Robert Gentleman founded the Bioconductor project. He is now the vice president of computational biology at 23andMe.

Why R?

Generic reasons:

- Free
- Open source
- Community
- Documentation
- Plots (e.g. https://www.r-graph-gallery.com/)

Other reasons:

- Heavily used in computational biology
- Heavily used in rapid prototyping in data science
- Easy to develop packages and share
- Many courses and help online

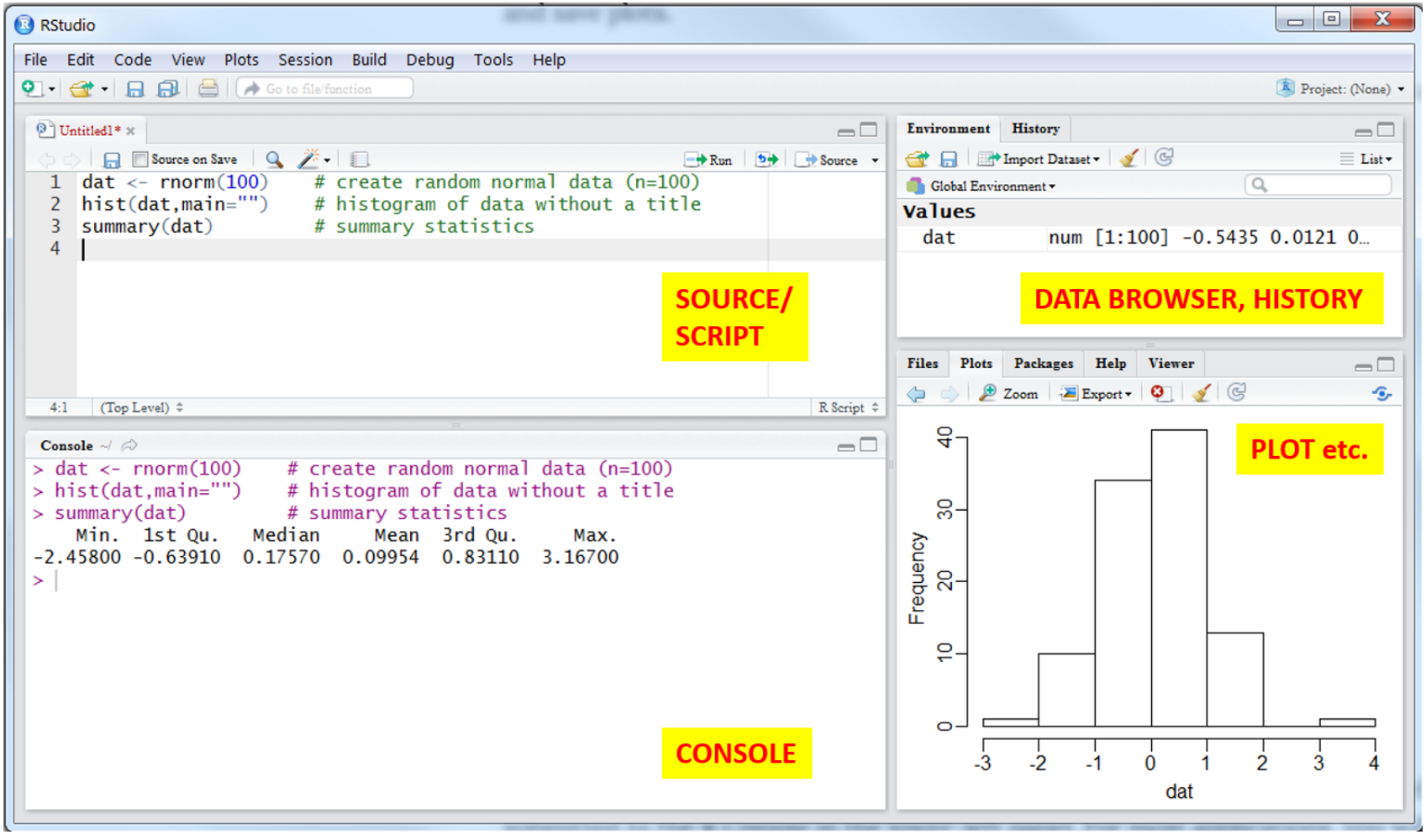
Disadvantages of R compared to Python

- Packages are not streamlined
- Data handling: objects are stored in physical memory
- Integration with web applications
- Speed, but this depends on coding skills and data size
- AI community prefers Python

There are solutions (e.g. wrappers for deep learning packages).

What is Rstudio?

It is an integrated development environment (IDE) for R.



Why Rstudio?

- Free
- Interface
- Free education materials (e.g. webinars)
- Cheatsheets
- Server version available
- Push boundaries on reproducibility (notebooks, R markdowns) and interactivity (Shiny)

Function and packages

Function
A set of codes to performs a specific task

- takes in some input values
- returns the desired output
- can be user-defined or from published packages

Package
A collection of R functions from other people. Installed once per machine. Main sources of packages:

- CRAN
- Bioconductor
- Github repositories (usually in development)

CRAN & Bioconductor

The Comprehensive R Archive Network (CRAN)

- https://cran.r-project.org/
- 14,913 software packages
- Installed via `install.packages("packageName")`

Bioconductor

- https://bioconductor.org/
- 1,741 software packages
- 948 annotation + 371 experiment data + 27 workflows
- Installed via `BiocManager::install("packageName")` [2]

Two ways to call installed packages:

- `library(packageName)`
- `pacman::p_load(packageName)`

---

[2] You may need to run `install.packages("BiocManager")` or `install.packages("pacman")` first.

SESSION 1

Learning objectives for Session 1

Objectives:

1. Using R as a calculator
2. Scripting
3. File import / export
4. Using tidyverse for plotting

R AS A CALCULATOR

R as a calculator | Basic arithmetic operators

- Add (+) or Subtract (-)
- Multiply (*) or Divide (/)
- Exponentiate (^ or **)

```
> 1 + 2
[1] 3
```

```
> 8^3 / 10                    # same as ( 8 * 8 * 8 ) / 10
[1] 51.2
```

See `help(Arithmetic)` for further operators.

Note: Any text after # is a **comment** and not executed.

R as a calculator | Operator precedence

What is the output of the following command?

```
> 1+2 * 6
```

R as a calculator | Operator precedence

What is the output of the following command?

```
> 1+2 * 6
```

Correct answer is 13.

R has interprets it as `1 + (2*6)`. Why?

R as a calculator | Operator precedence

What is the output of the following command?

```
> 1+2 * 6
```

Correct answer is 13.

R has interprets it as `1 + (2*6)`. Why?

Because of operator precedence. See `help(Syntax)` which includes (from highest to lowest):

1. Brackets (or)
2. Exponentiation (^)
3. Multiplication or division (* or /)
4. Addition or subtraction (+ or -)

Note: **Spaces** do not matter for R software interpretation but ***consistent* spacing** makes your code readable to you and others.

R as a calculator | Example

Open your Rstudio. Use the **console** to calculate the following:

1. What is the sum of 1234567 and 87654321 ?

R as a calculator | Example

Open your Rstudio. Use the **console** to calculate the following:

1. What is the sum of 1234567 and 87654321 ?

2. What is 50kg in pounds?
   Formula: 1 kg is approximately 2.2 pounds.

R as a calculator | Example

Open your Rstudio. Use the **console** to calculate the following:

1. What is the sum of 1234567 and 87654321 ?

2. What is 50kg in pounds?
   Formula: 1 kg is approximately 2.2 pounds.

3. What is 35 degrees Celsius in Fahrenheit?
   Formula: Fahrenheit $=$ 1.8 x Celsius $+$ 32.

R as a calculator | Example

Open your Rstudio. Use the **console** to calculate the following:

1. What is the sum of 1234567 and 87654321 ?

2. What is 50kg in pounds?
   Formula: 1 kg is approximately 2.2 pounds.

3. What is 35 degrees Celsius in Fahrenheit?
   Formula: Fahrenheit $=$ 1.8 x Celsius $+$ 32.

4. What is log base 10 of 1000?

R as a calculator | Logarithms

R has many built-in functions. This includes: `log2()`, `log()`, `log10()`, `sqrt()`, `exp()`.

```
log10(1000)
## [1] 3
```

```
log(1000, base=10)
## [1] 3
```

This just means you have to multiply 10 with itself **three times**: $10^3 = 10 * 10 * 10 = 1000$

SCALAR & SCRIPTS

How do call someone on the phone?

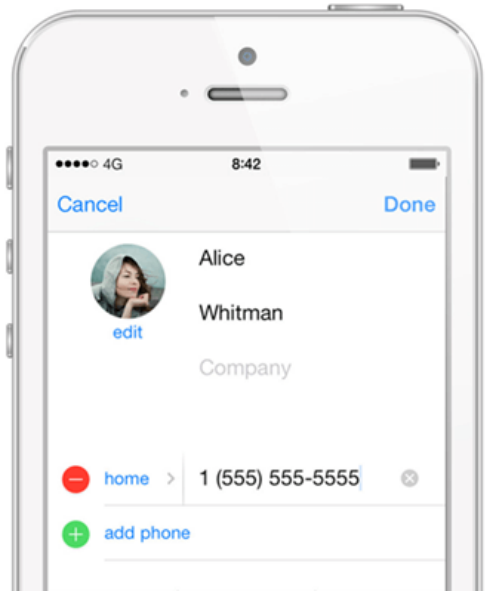
How do call someone on the phone?

Do you

1. Dial someones phone number in full from memory into the dialpad?

or

2. Search the "Contacts" and then click it? If yes, you need to first assign/save the number into a Contact detail.



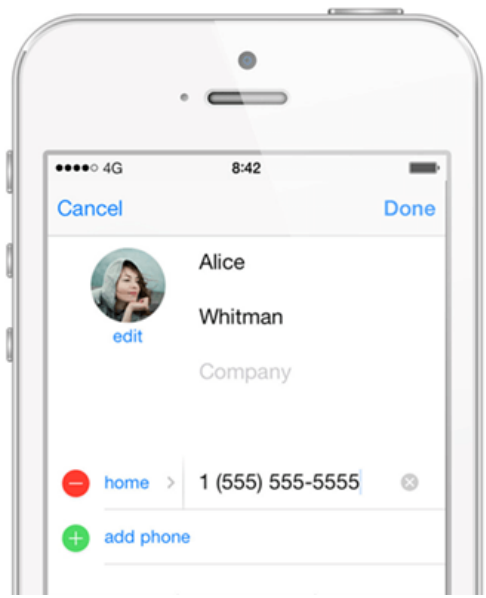
How do call someone on the phone?

Do you

1. Dial someones phone number in full from memory into the dialpad?

or

2. Search the "Contacts" and then click it? If yes, you need to first assign/save the number into a Contact detail.



Similarly in R, we can assign data into variables for use later.

Scalar | Assigning to a variable

You can assign a value to a variable. This allows you to write more flexible programs. The assignment operator in R is "<-".

```
pow <- 3
10^pow
## [1] 1000
```

You can work with as many variables as you like. e.g.

```
pow  <- 3
base <- 10
base^pow
## [1] 1000
```

Scalar | Assigning to a variable

You can also assign words and strings into variables

```
n.samples <- 3
cellline  <- "HEK 293"

paste("The experiment used ",
      cellline, " cellline (n=", n.samples, ")",
      sep="")
## [1] "The experiment used HEK 293 cellline (n=3)"
```

Scalar | Assigning to a variable

You can also assign words and strings into variables

```
n.samples <- 3
cellline  <- "HEK 293"

paste("The experiment used ",
      cellline, " cellline (n=", n.samples, ")",
      sep="")
## [1] "The experiment used HEK 293 cellline (n=3)"
```

You cannot apply arithmetic operations onto string variables, even if they look numeric. E.g.

```
X <- "1"
Y <- "2"
X + Y
## Error in X + Y : non-numeric argument to binary operato
```

Scalar | Assigning to a variable

R is case sensitive:.

```
what <- 111
WHAT <- 222
what
## [1] 111
WHAT
## [1] 222
```

Naming rules:

- *Must* start with a character
- *Must not* use the reserved words (e.g. if, NA, TRUE, FALSE, in, c). See `help(reserved)` for a fuller list.
- Recommend to use lower case and underscore (e.g. day_one) where possible

Scalar | Assigning to a variable

```
X <- 123
Y <- X
```

What is the value of Y?

Scalar | Assigning to a variable

```
X <- 123
Y <- X
```

What is the value of Y?

```
X <- 0
```

What is the value of X and Y now?

Scalar | Assigning to a variable

```
X <- 123
Y <- X
```

What is the value of Y?

```
X <- 0
```

What is the value of X and Y now?

```
X
## [1] 0
Y
## [1] 123
```

Summary

1. Assignment to same variable name replaces the value
2. The **value** of the variable gets assigned (No pointers in R)

Scripting | What is it?

A script is a plain text document with lists of commands.

Scripting | Exercise

Exercise:

1. Create a folder called R_workshop/.
2. In Rstudio, open a new script (File -> New File -> R Script).
3. Save the script as **"sandbox.R"** in R_workshop folder.
4. Try some of the previous exercises in the script.
   Save the script. Close RStudio.
5. Open this file in Notepad. Make some edits and save.
6. Open the file back in RStudio.

Scripting | Keyboard shortcuts

1. **Ctrl-Enter** on a line sends it to the console for execution.

a. If code is incomplete on first line, it will continue automatically till end

b. You can execute multiple lines or a portion of the line if you highlight it and press **Ctrl-Enter**.

3. **Ctrl-S** to save the script.

Scripting | Benefits

Benefits of using a script over typing in the console:

- Store the final/relevant syntax
- Persistence across different R sessions
- Syntax highlighting and indentation
- Easy to share with others
- Can include extensive comments
- Comment/uncomment codes (Ctrl-Shift-C)

R as a calculator | qPCR Exercise

Condition	Target	CT (rep 1)	CT (rep 2)	CT (rep 3)
Wild type	AXIN2	25.8	25.7	25.9
Wild type	GAPDH	16.7	16.6	16.5
Knockout	AXIN2	28.2	28.0	28.1
Knockout	GAPDH	16.9	16.7	16.9

Exercise: Calculate the relative expression of AXIN2 after knockout using the $2^{-\Delta\Delta CT}$ rule.

R as a calculator | qPCR Exercise

Condition	Target	CT (rep 1)	CT (rep 2)	CT (rep 3)
Wild type	AXIN2	25.8	25.7	25.9
Wild type	GAPDH	16.7	16.6	16.5
Knockout	AXIN2	28.2	28.0	28.1
Knockout	GAPDH	16.9	16.7	16.9

Exercise: Calculate the relative expression of AXIN2 after knockout using the $2^{-\Delta\Delta CT}$ rule.

Steps:

1. Calculate the average for each row (condition and target)

R as a calculator | qPCR Exercise

Condition	Target	CT (rep 1)	CT (rep 2)	CT (rep 3)
Wild type	AXIN2	25.8	25.7	25.9
Wild type	GAPDH	16.7	16.6	16.5
Knockout	AXIN2	28.2	28.0	28.1
Knockout	GAPDH	16.9	16.7	16.9

Exercise: Calculate the relative expression of AXIN2 after knockout using the $2^{-\Delta\Delta CT}$ rule.

Steps:

1. Calculate the average for each row (condition and target)
2. Calculate the ΔCT (= AXIN2 average - GAPDH average) for knockout and for wild type

R as a calculator | qPCR Exercise

Condition	Target	CT (rep 1)	CT (rep 2)	CT (rep 3)
Wild type	AXIN2	25.8	25.7	25.9
Wild type	GAPDH	16.7	16.6	16.5
Knockout	AXIN2	28.2	28.0	28.1
Knockout	GAPDH	16.9	16.7	16.9

Exercise: Calculate the relative expression of AXIN2 after knockout using the $2^{-\Delta\Delta CT}$ rule.

Steps:

1. Calculate the average for each row (condition and target)
2. Calculate the ΔCT (= AXIN2 average - GAPDH average) for knockout and for wild type
3. Calculate the $\Delta\Delta CT$ (Knockout ΔCT - Wild type ΔCT)

R as a calculator | qPCR Exercise

Condition	Target	CT (rep 1)	CT (rep 2)	CT (rep 3)
Wild type	AXIN2	25.8	25.7	25.9
Wild type	GAPDH	16.7	16.6	16.5
Knockout	AXIN2	28.2	28.0	28.1
Knockout	GAPDH	16.9	16.7	16.9

Exercise: Calculate the relative expression of AXIN2 after knockout using the $2^{-\Delta\Delta CT}$ rule.

Steps:

1. Calculate the average for each row (condition and target)
2. Calculate the ΔCT (= AXIN2 average - GAPDH average) for knockout and for wild type
3. Calculate the $\Delta\Delta CT$ (Knockout ΔCT - Wild type ΔCT)
4. Exponentiate to base 2

R as a calculator | qPCR solution

Condition	Target	CT (rep 1)	CT (rep 2)	CT (rep 3)
Wild type	AXIN2	25.8	25.7	25.9
Wild type	GAPDH	16.7	16.6	16.5
Knockout	AXIN2	28.2	28.0	28.1
Knockout	GAPDH	16.9	16.7	16.9

```
dCT_wt <- (25.8 + 25.7 + 25.9)/3 - (16.7 + 16.6 + 16.5)/3
dCT_KO <- (28.2 + 28.0 + 28.1)/3 - (16.9 + 16.7 + 16.9)/3
```

```
print( ddCT <- dCT_KO - dCT_wt ) # Delta delta CT
## [1] 2.066667
print( RE   <- 2^-ddCT )         # relative expression
## [1] 0.2387104
```

R as a calculator | Importance of good coding

> Good coding is like using correct punctuation. You can manage without it, but it makes things easier to read.
>
> Hadley Wickham

Bad example 1: Not using variables

```
2^-( ( (28.2 + 28.0 + 28.1)/3 - (16.9 + 16.7 + 16.9)/3 )
     - ( (25.8 + 25.7 + 25.9)/3 - (16.7 + 16.6 + 16.5)/3 )
## [1] 0.2387104
```

You still get the correct answer but it is:

- Difficult to figure out logic and order of things
- Very long expression if written on one line
- Easier to make mistakes
- Difficult to change values

Bad example 2: Not using consistent spacing

```
dCT_wt<-(25.8+25.7+25.9)/3-(16.7+16.6+16.5)/3
dCT_KO <-(28.2  + 28.0  + 28.1 )/
  3-(16.9  + 16.7  + 16.9 )/3

print(  ddCT        <-        dCT_KO-dCT_wt )
## [1] 2.066667
print(RE<-2^ - ddCT)
## [1] 0.2387104
```

Space does not cost much. Please use them and use them wisely.

Some programming wisdoms

Always code as if the guy who ends up maintaining it will be a violent psychopath who knows where you live. Code for readability.

John Woods (1991)

Some programming wisdoms

Always code as if the guy who ends up maintaining it will be a violent psychopath who knows where you live. Code for readability.

John Woods (1991)

Any code of your own that you haven't looked at for last 6 months might as well have been written by someone else.

Eagleson's Law of Programming

R as a calculator | Comparison operators

To compare two values. See `help(Comparison)`

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

Note: The equality comparison is "==" (double equal sign).

R as a calculator | Comparison operators and control flow

Comparison operators comes in handy when you want to trigger some action based on output. Examples:

- `if()`
- `ifelse()`
- `if() { ... } else { ... }`. The `else` must not start on a newline.

R as a calculator | Comparison operators and control flow

Comparison operators comes in handy when you want to trigger some action based on output. Examples:

- `if()`
- `ifelse()`
- `if() { ... } else { ... }`. The `else` must not start on a newline.

Here is the predicted air quality for next week:

```
aq <- c(Mon=9, Tue=50, Wed=65, Thu=4, Fri=6, Sat=8, Sun=55)
## Mon Tue Wed Thu Fri Sat Sun
##   9  50  65   4   6   8  55
ifelse( aq <= 50, "ok", "stay indoors")
##  Mon  Tue              Wed  Thu  Fri  Sat              Sun
## "ok" "ok" "stay indoors" "ok" "ok" "ok" "stay indoors"
```

R as a calculator | Comparison operators and control flow

Comparison operators comes in handy when you want to trigger some action based on output. Examples:

- `if()`
- `ifelse()`
- `if() { ... } else { ... }`. The `else` must not start on a newline.

Here is the predicted air quality for next week:

```
aq <- c(Mon=9, Tue=50, Wed=65, Thu=4, Fri=6, Sat=8, Sun=55)
## Mon Tue Wed Thu Fri Sat Sun
##   9  50  65   4   6   8  55
ifelse( aq <= 50, "ok", "stay indoors")
##  Mon  Tue              Wed  Thu  Fri  Sat              Sun
## "ok" "ok" "stay indoors" "ok" "ok" "ok" "stay indoors"
```

Other control flows structures: `for()`, `stop()`, `stopifnot()`, `break()`, `return()`, `switch()`, `while()`, `next()`

Comparison operators and control structures | Example

Here is one way of putting the relative expression in English (e.g. for automated report):

```
RE
## [1] 0.2387104
Dir <- ifelse( RE < 1, "down", "up" )
magnitude <- ifelse( RE < 1, 1/RE, RE )
magnitude <- format(magnitude, digits=2) # make it pretty

paste("AXIN2 is ", magnitude, "-fold ", Dir,
      "regulated in knockout.", sep="")
## [1] "AXIN2 is 4.2-fold downregulated in knockout."
```

Comparison operators and control structures | Example

Solution 2 - This might be better for longer codes

```
if(RE > 1){

  paste("AXIN2 is ", format(RE, digits=2), "-fold ",
        "upregulated in knockout.", sep="")

} else {

  paste("AXIN2 is ", format(1/RE, digits=2), "-fold ",
        "downregulated in knockout.", sep="")
}
## [1] "AXIN2 is 4.2-fold downregulated in knockout."
```

Notice how the `else()` statement is surrounded by curly brackets.

Keeping your workspace clean

We generated a lot of objects in our environment:

```
ls()
##  [1] "base"      "cellline"  "dCT_KO"    "dCT_wt"    "d
##  [6] "Dir"       "magnitude" "n.samples" "pow"       "R
## [11] "what"      "WHAT"      "X"         "Y"
```

Cluttered workspace increases risk of coding errors. Let's remove a few objects:

```
rm(what, WHAT)
ls()
##  [1] "base"      "cellline"  "dCT_KO"    "dCT_wt"    "d
##  [6] "Dir"       "magnitude" "n.samples" "pow"       "R
## [11] "X"         "Y"
```

Keeping your workspace clean

We generated a lot of objects in our environment:

```
ls()
##  [1] "base"      "cellline"  "dCT_KO"    "dCT_wt"    "d
##  [6] "Dir"       "magnitude" "n.samples" "pow"       "R
## [11] "what"      "WHAT"      "X"         "Y"
```

Cluttered workspace increases risk of coding errors. Let's remove a few objects:

```
rm(what, WHAT)
ls()
##  [1] "base"      "cellline"  "dCT_KO"    "dCT_wt"    "d
##  [6] "Dir"       "magnitude" "n.samples" "pow"       "R
## [11] "X"         "Y"
```

Or if you want to remove everything

```
rm( list=ls() )
```

DATA IMPORT EXPORT

Setup | Part 1

First we need to setup our working environment

1. Create a subfolder called **data** in R_workshop/ folder
2. Place the "session1_data.xlsx" file into R_workshop/data/
3. Create an new R script. Save it **"iris.R"** in R_workshop/

Setup | Part 2 (getwd)

Execute the following and discuss the outputs:

```
getwd()
list.files()
```

Setup | Part 2 (getwd)

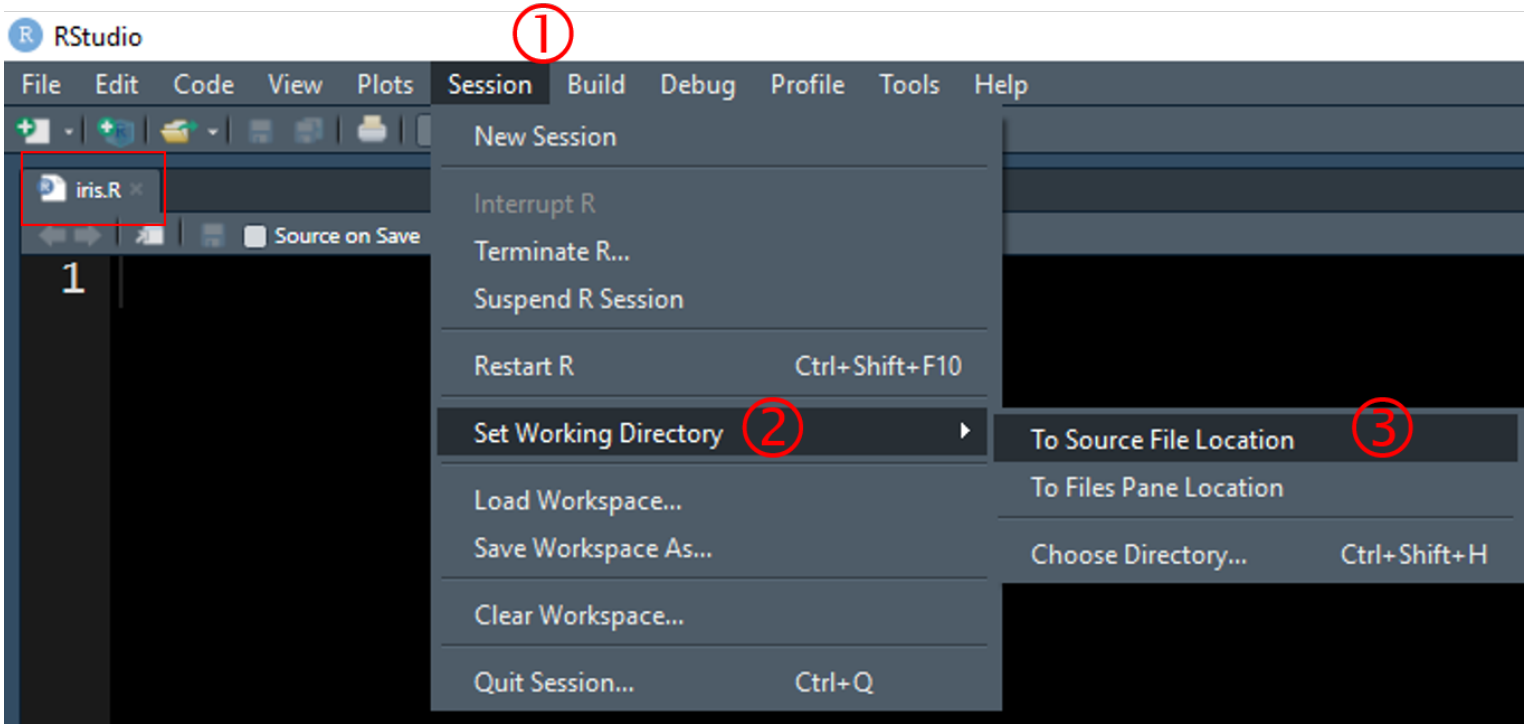
Execute the following and discuss the outputs:

```
getwd()
list.files()
```

Working directory:
Default location where R will look for and write files.

Setup | Part 2 (setwd)

1. Change the working directory so it is easy to find your files. (Session ⇒ Set Working Dir. ⇒ To Source File Location)



2. Copy the syntax in the console (without the "$>$" prompt) into your script for future use (to skip Step 1). It ***might*** look like: `setwd("C:/Users/aramasamy/Desktop/R_workshop")`

What do you see with `list.files(recursive=TRUE)`?

Setup | Part 3 (loading packages)

We need to load a few packages in order to make use of the functions contained within these packages.

```
pacman::p_load(tidyverse, readxl, writexl, janitor, broom)
```

Setup | Part 3 (loading packages)

We need to load a few packages in order to make use of the functions contained within these packages.

```
pacman::p_load(tidyverse, readxl, writexl, janitor, broom)
```

Note: Run the following if you get an error message that there is no package called pacman (once per machine).

```
install.packages("pacman")
```

Setup | Part 3 (loading packages)

We need to load a few packages in order to make use of the functions contained within these packages.

```
pacman::p_load(tidyverse, readxl, writexl, janitor, broom)
```

Note: Run the following if you get an error message that there is no package called pacman (once per machine).

```
install.packages("pacman")
```

Note: `p_load()` is the same as `library()` except

- tries to install a package from CRAN if neccessary.
- a bit more compact to write

`p_unload()` and `p_install()` also available.

Data import | The Iris data

Here is the command to read in the iris data from the Excel file[3].

```
iris <- read_excel("data/session1_data.xlsx", sheet="iris")
```

Describe the data. The following commands might be useful:

```
dim(iris)
View(iris)  ## Same as clicking on the data browser
head(iris)  ## or tail(iris)
colnames(iris)
str(iris)
glimpse(iris)
summary(iris)
```

Data import | The Iris data

Here is the command to read in the iris data from the Excel file[3].

```
iris <- read_excel("data/session1_data.xlsx", sheet="iris")
```

Describe the data. The following commands might be useful:

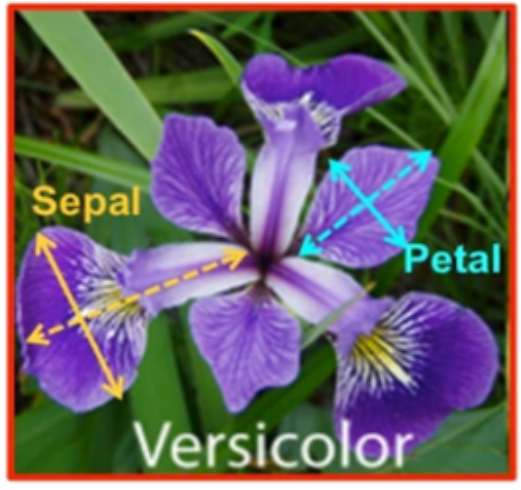
```
dim(iris)
View(iris)  ## Same as clicking on the data browser
head(iris)  ## or tail(iris)
colnames(iris)
str(iris)
glimpse(iris)
summary(iris)
```

To see the sheet names (without opening it in Excel):

```
excel_sheets("data/session1_data.xlsx")
## [1] "iris"          "chick_weight"  "sim_gene"      "he
## [5] "qPCR_example"
```

The iris data | Description

- Three iris species: setosa, virginica and versicolor
- 50 flower samples from each species
- Four features measured from each flower (in cm)
  - Sepal Length
  - Sepal Width
  - Petal Length
  - Petal Width



Type `help(iris)` for more details.

Data import | Learning more about a function

1. Full documentation: `help(read_excel)`
2. Autocompletion (press Tab inside function name parentheses)
3. Quick way to find out the possible input parameters:

```
args(read_excel)
## function (path, sheet = NULL, range = NULL, col_names =
##     col_types = NULL, na = "", trim_ws = TRUE, skip = 0
##     guess_max = min(1000, n_max), progress = readxl_prog
##     .name_repair = "unique")
## NULL
```

You might spot a few useful arguments: `col_names`, `na`, `skip`.

Data export | Write to CSV

Here is one way to write a subset of the data into comma-separated file (CSV)[4]

```
iris %>%
  filter(Species=="setosa") %>%
  select(Sepal.Length, Sepal.Width) %>%
  write_csv("iris_setosa_sepalLW.csv")
```

[4] You can also use the `write_xlsx()` from the writexl package.

Data export | Write to CSV

Here is one way to write a subset of the data into comma-separated file (CSV)[4]

```
iris %>%
  filter(Species=="setosa") %>%
  select(Sepal.Length, Sepal.Width) %>%
  write_csv("iris_setosa_sepalLW.csv")
```

The **pipe command (%>%)**

- passes an intermediate result onto next function (chaining)
- without storing the intermediate result
- The keystroke "Ctrl-Shift-M" generates this symbol

[4] You can also use the `write_xlsx()` from the writexl package.

readr package for plain text | https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf

Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append)

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)

String to file
write_file(x, path, append = FALSE)

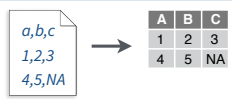
String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE)

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)

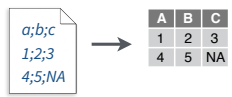
Read Tabular Data - These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive()**)**



A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv")
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")



A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv")
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

A	B	C
1	2	3
4	5	NA

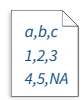
Files with Any Delimiter
read_delim("file.txt", delim = "|")
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5))
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

A	B	C
1	2	3
4	5	NA

Tab Delimited Files
read_tsv("file.tsv") Also **read_table()**.
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS



Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

A	B	C
NA	2	3
4	5	NA

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale())

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive())

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())

Read a file into a raw vector
read_file_raw(file)

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems.
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing.
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime(**format = ""**)** Also **col_date(**format = ""**)**, **col_time(**format = ""**)**
- **col_factor(**levels, ordered = FALSE**)**
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

x <- read_csv("file.csv", col_types = cols(
A = col_double(),
B = col_logical(),
C = col_factor()))

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

x$A <- parse_number(x$A)

Data import & export | General packages

Package	Format
readr	**ASCII (plain text)**
readxl, writexl	**excel files (.xls and .xlsx)**
haven	SPSS, Stata, and SAS files
DBI	SQL database
jsonlite	json
xml2	XML
httr	Web APIs
rvest	HTML (Web Scraping)

VISUALIZATION WITH TIDYVERSE

ggplot2

ggplot2 is an R packages based on the **g**rammar of **g**raphics.

Essential Elements

- Data: A dataframe where columns are used for aesthetics
- Aesthetics: position (x, y) and attributes (color, shape, size)
- Geometries: the data is displayed (bars, points, lines)

ggplot2

ggplot2 is an R packages based on the **g**rammar of **g**raphics.

Essential Elements

- Data: A dataframe where columns are used for aesthetics
- Aesthetics: position (x, y) and attributes (color, shape, size)
- Geometries: the data is displayed (bars, points, lines)

Optional Elements

- Facets: splits the graph by subsets
- Statistics: add means, medians, etc
- Coordinates: Transforms axes (e.g. log10)
- Themes: change the graphics background, axis size, header, etc

ggplot2 | https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

required

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION> (mapping = aes( <MAPPINGS> ),
  stat = <STAT>, position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point**)** Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z**))** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group**))**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1**))** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900**))** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1**))**
b + geom_hline(aes(yintercept = lat**))**
b + geom_vline(aes(xintercept = long**))**

b + geom_segment(aes(yend=lat+1, xend=long+1**))**
b + geom_spoke(aes(angle = 1:1155, radius = 1**))**

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy**))** x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**,
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

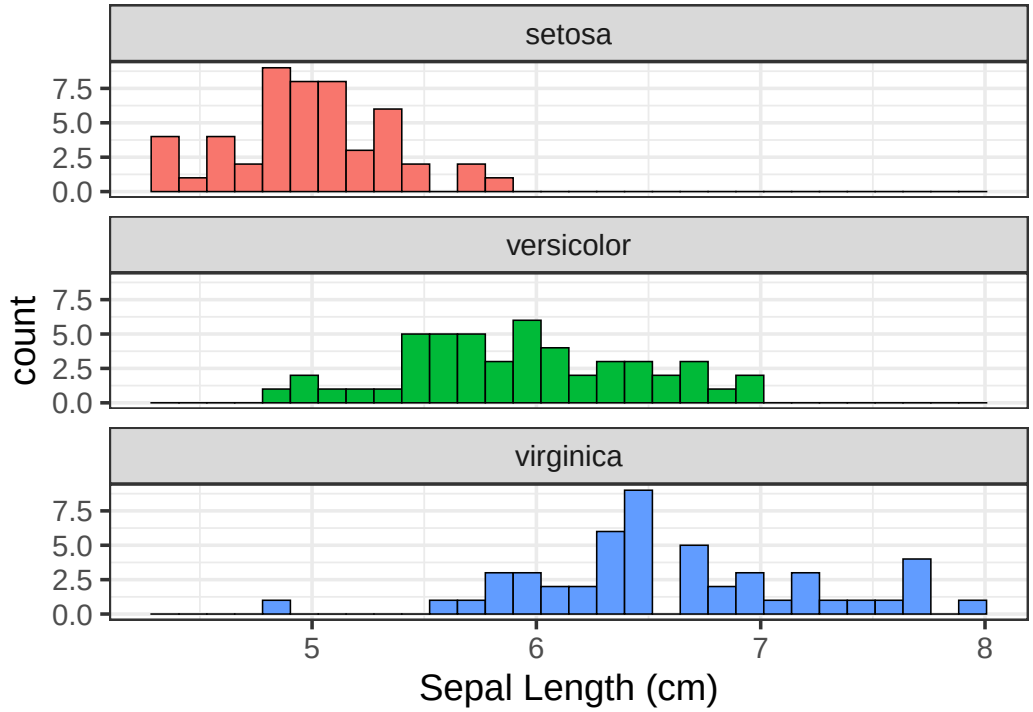
l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

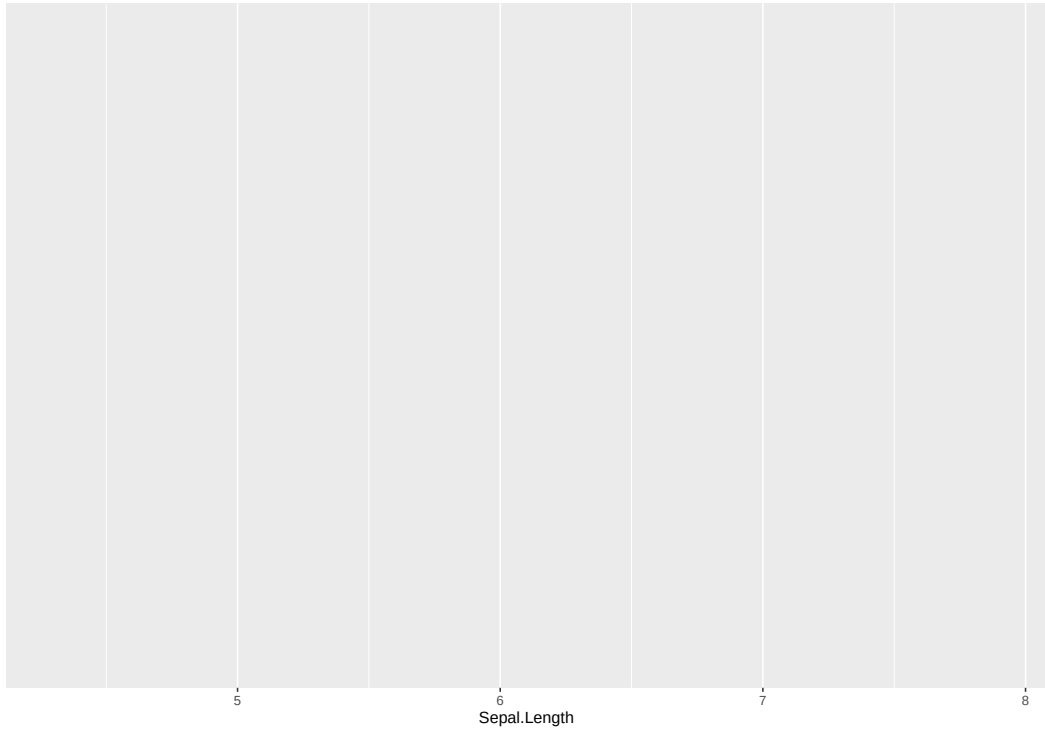
ggplot2 | Step-by-step

The following graph shows how the sepal lengths are distributed within each species.

Let us try to build this graph up step by step next.



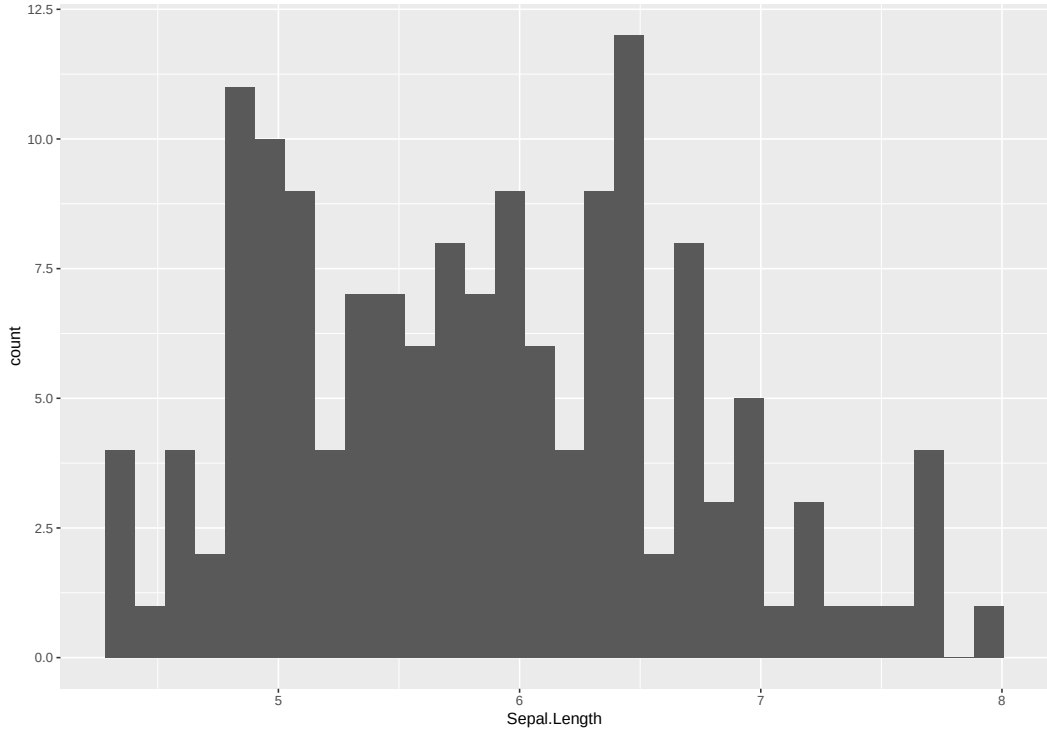
ggplot2 | Step 1 (setup empty plot)




```
ggplot(data=iris, aes(x=Sepal.Length))
```

Any variable names in `aes()` will be read from the data. So "`aes(x=Sepal.Length)`" means plot the Sepal.Length column from the iris object on the x-axis.

ggplot2 | Step 2 (add a geom)

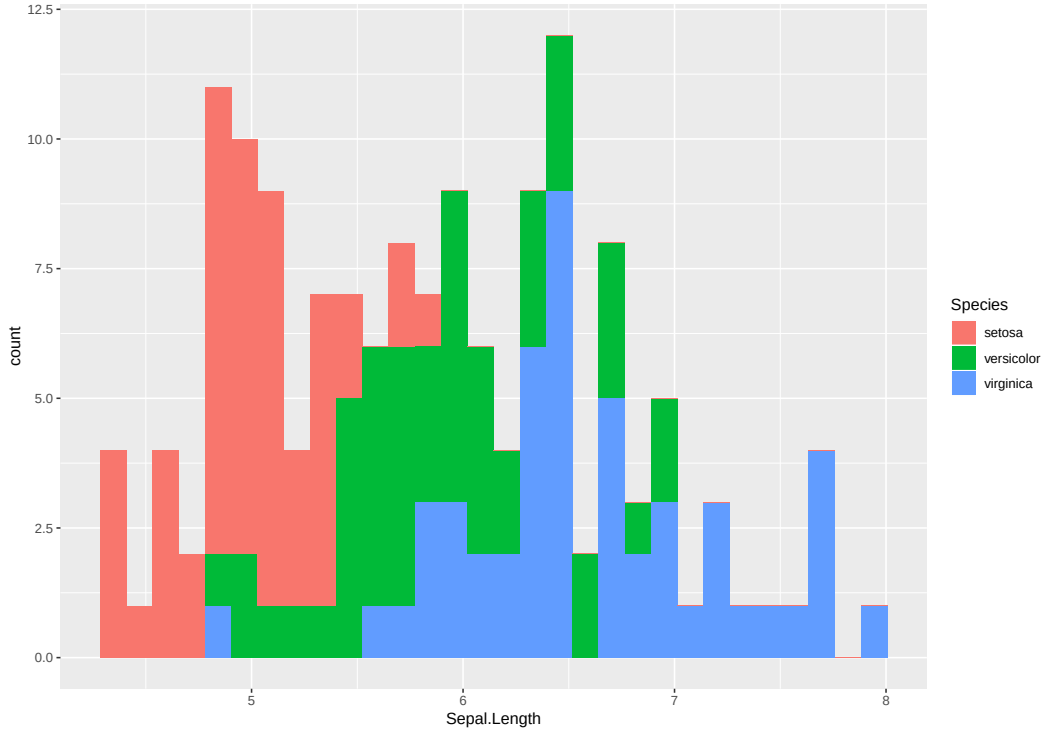



```
ggplot(data=iris, aes(x=Sepal.Length)) +
  geom_histogram()
```

The "+" simply means add a layer.

OK but not very useful for comparing the distribution by species.

ggplot2 | Step 3 (add a fill)

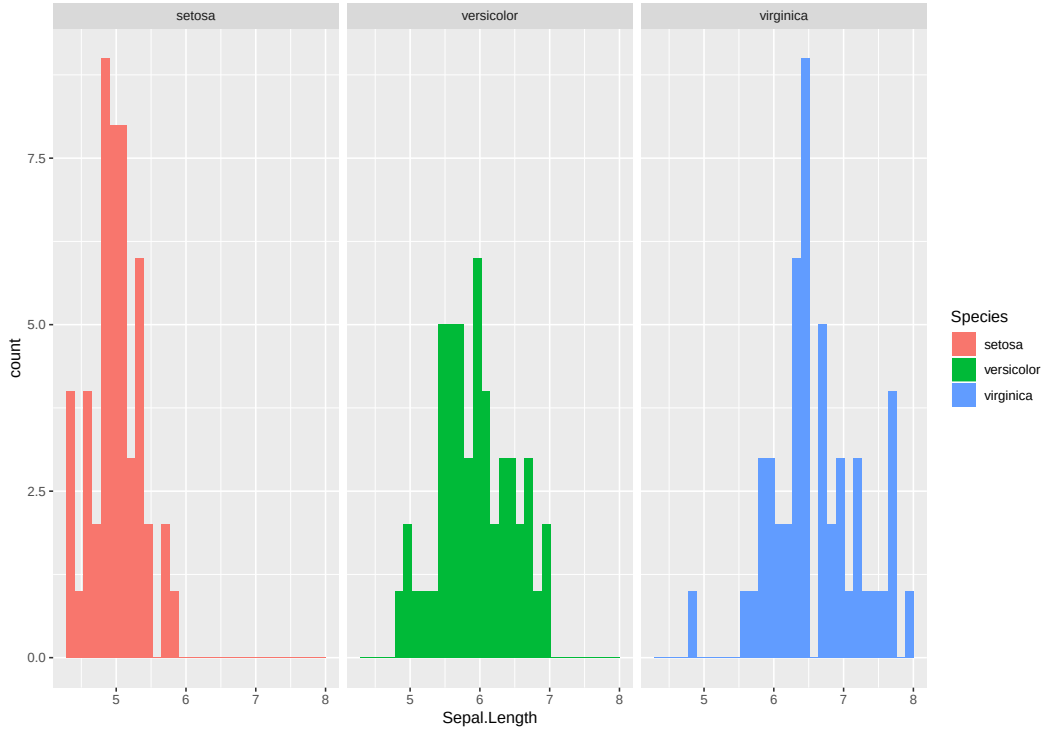



```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram()
```

Note: R is case sensitive (i.e. species is not same as Species).

This is better but the histograms are overlapping too much.

ggplot2 | Step 4 (subset and plot)

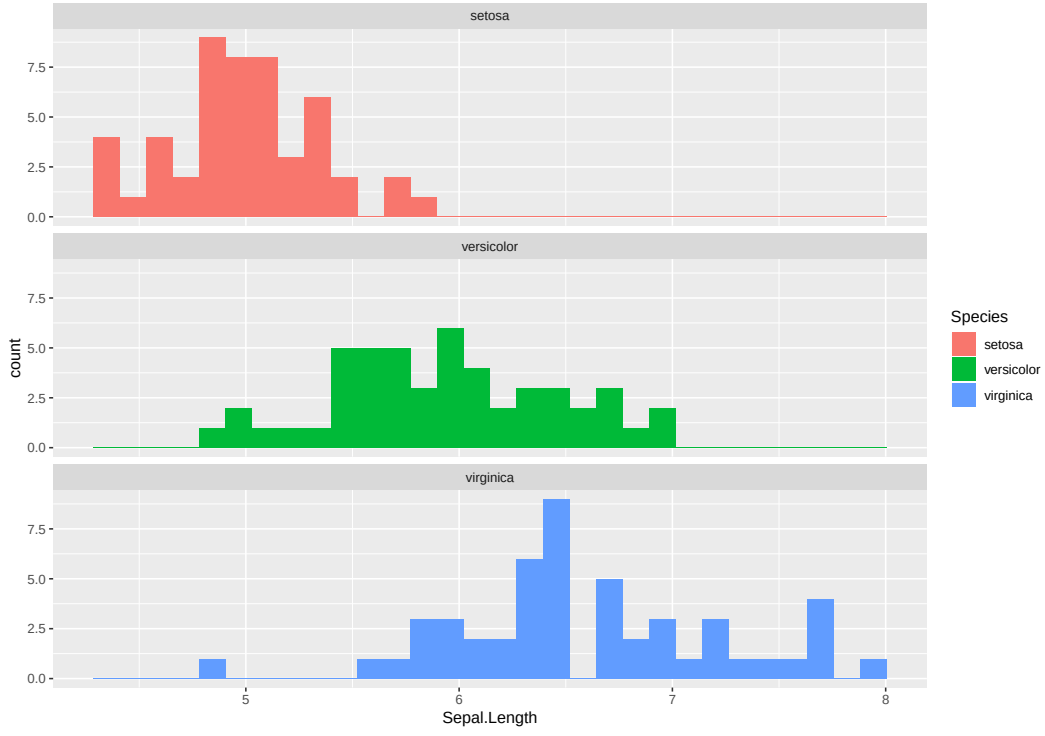



```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram() +
  facet_wrap( ~ Species)
```

`facet_wrap()` subset by Species and plot for each subset.

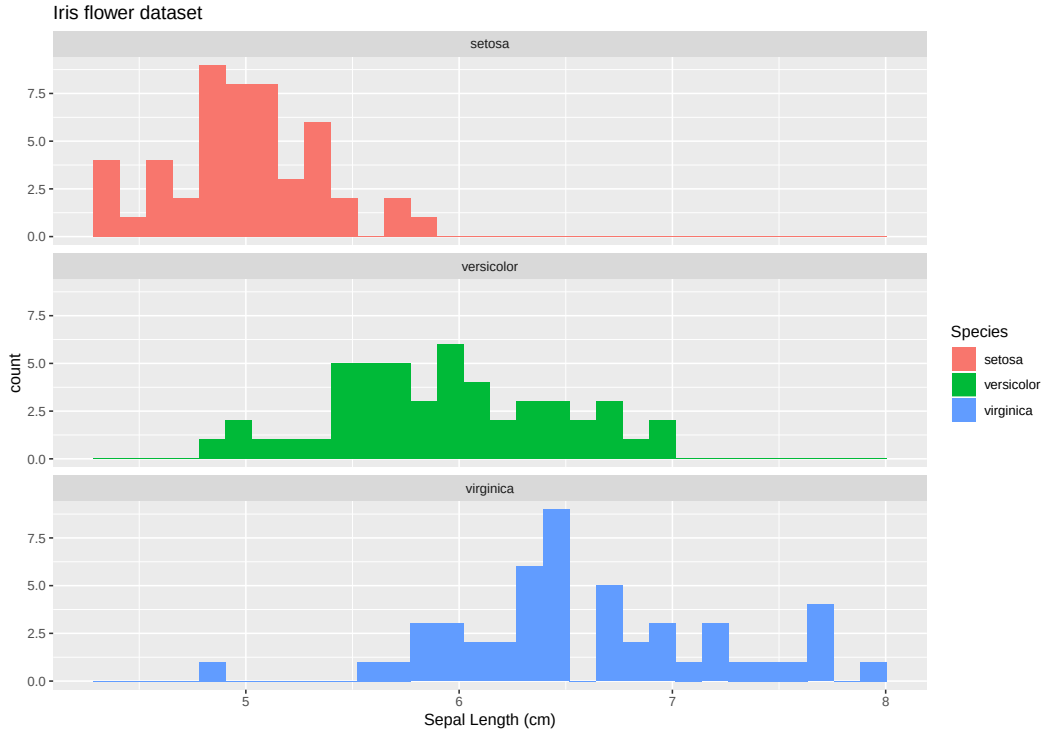
Improved but it is now difficult to visually compare the x-axis.

ggplot2 | Step 5 (re-arrange the panels in facet_wrap)



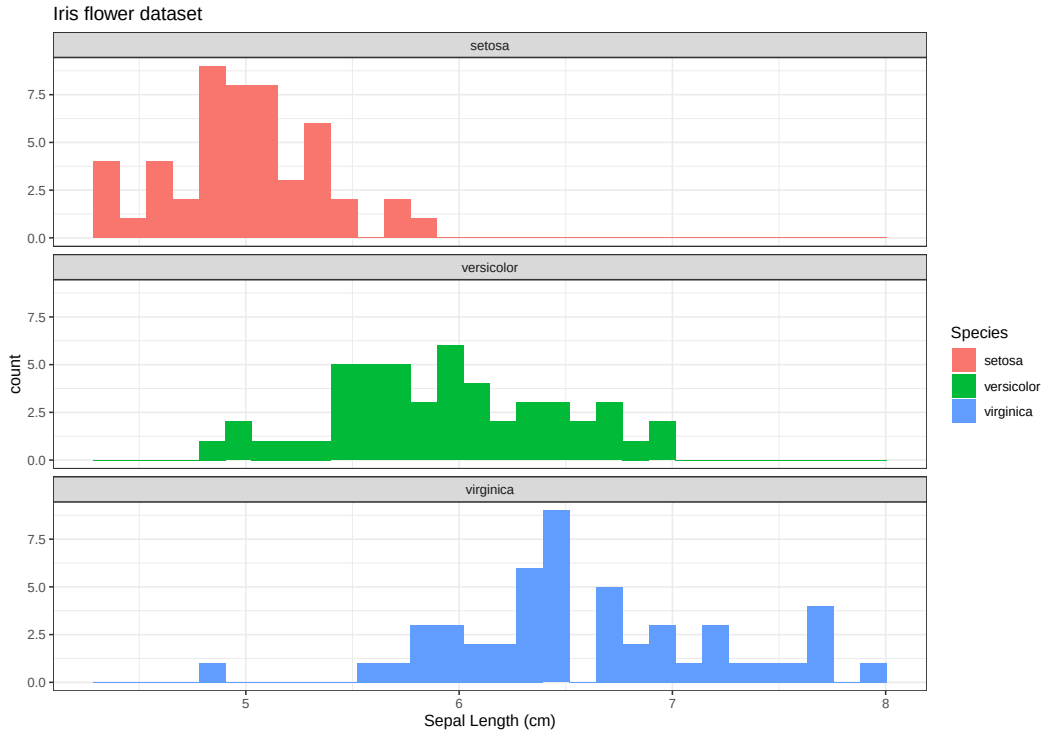

```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram() +
  facet_wrap( ~ Species, ncol=1)
```

ggplot2 | Step 6 (modify labels and title)




```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram() +
  facet_wrap( ~ Species, ncol=1) +
  labs(x="Sepal Length (cm)", title="Iris flower dataset")
```

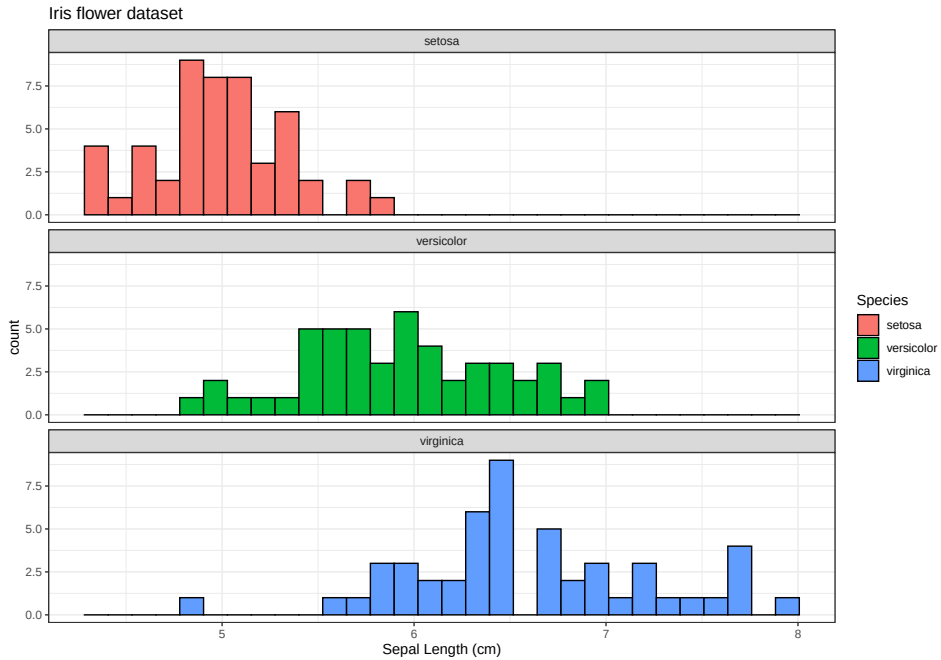
ggplot2 | Step 7 (remove the grey background to save ink)




```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram() +
  facet_wrap( ~ Species, ncol=1) +
  labs(x="Sepal Length (cm)", title="Iris flower dataset")
  theme_bw()
```

Now, the bars in the histograms look a bit ghostly.

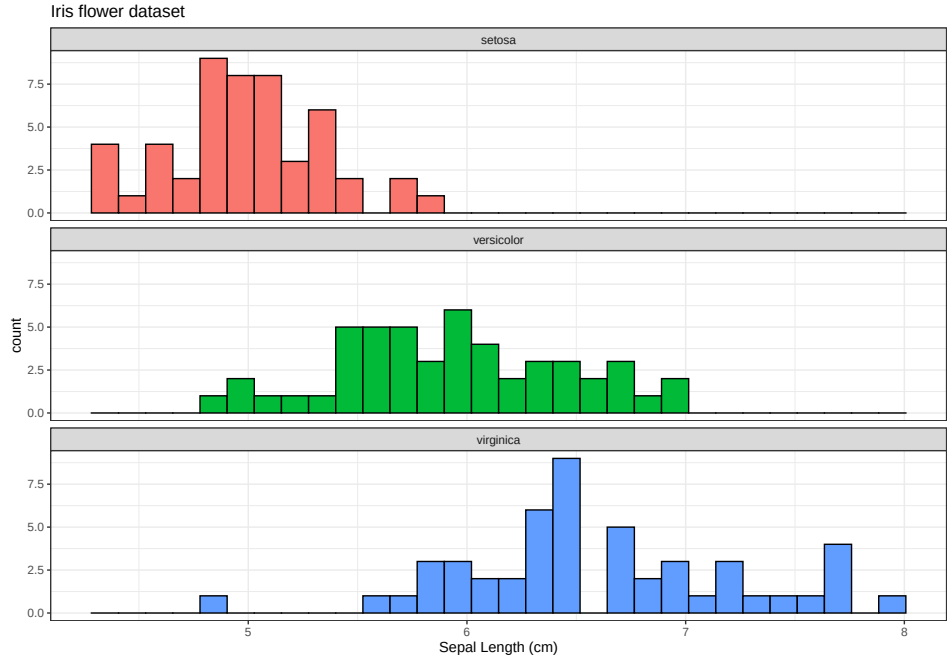
ggplot2 | Step 8 (add black borders to the bars)




```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram(col="black") +
  facet_wrap( ~ Species, ncol=1) +
  labs(x="Sepal Length (cm)", title="Iris flower dataset") +
  theme_bw()
```

Note: `col="black"` should **not** be inside the `aes()` as it is a fixed value (i.e. does not come from the data).

ggplot2 | Step 9 (remove the legend)

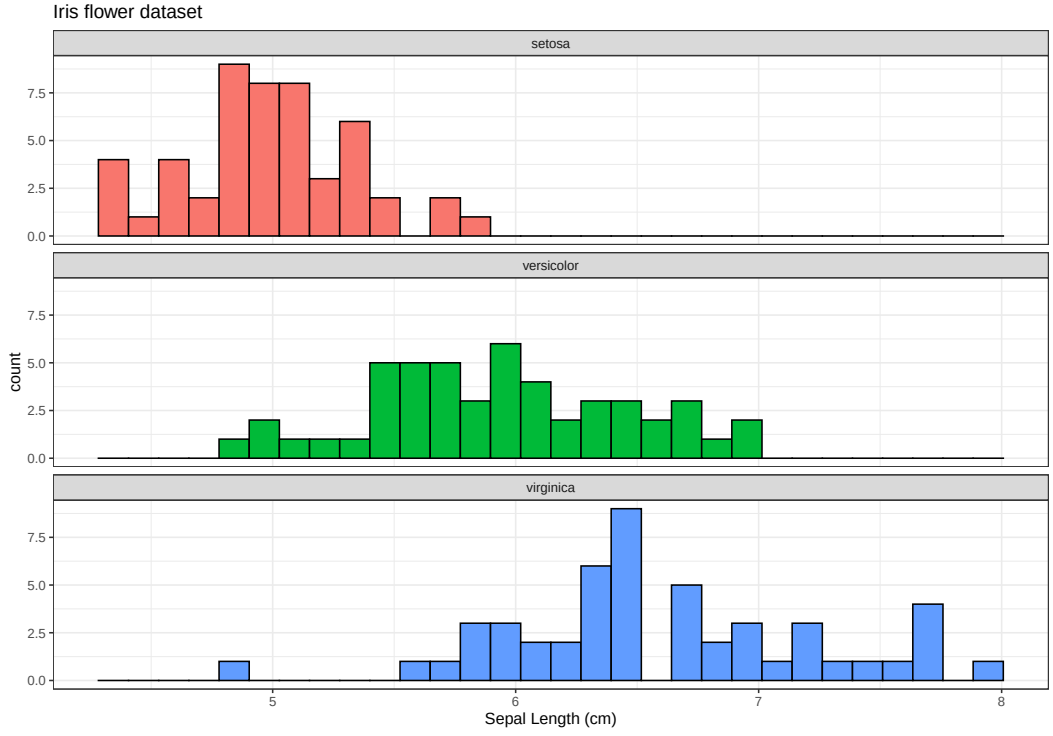



```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram(col="black") +
  facet_wrap( ~ Species, ncol=1) +
  labs(x="Sepal Length (cm)", title="Iris flower dataset")
  theme_bw() + theme(legend.position="none")
```

Legend is unnecessary here as the subpanels already have a title.
Note that `theme()` must come after `theme_bw()`.

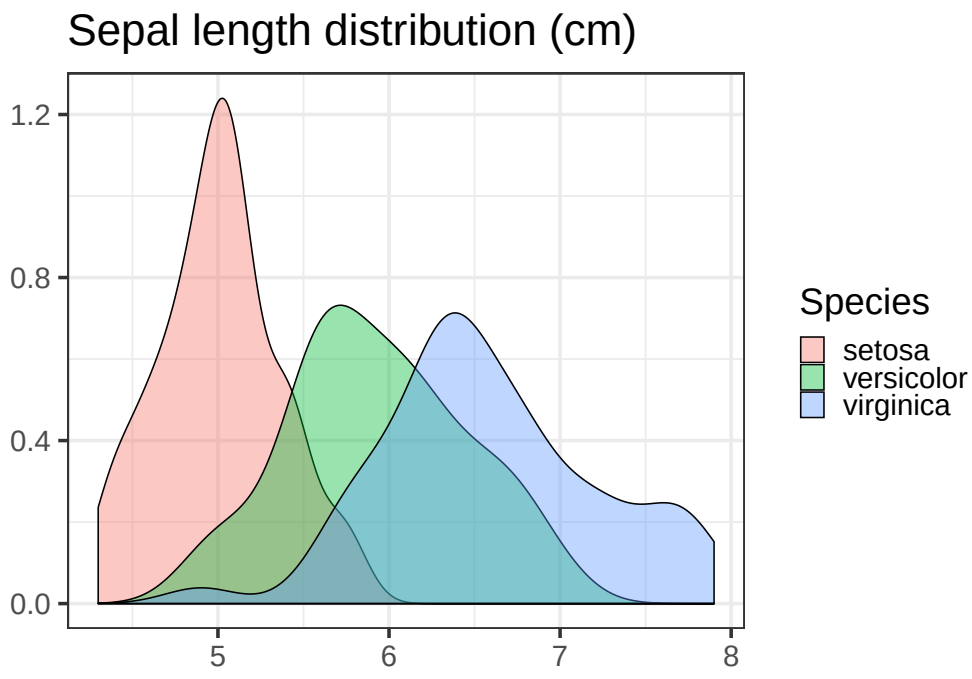
ggplot2 | Histogram - final output

```
ggplot(data=iris, aes(x=Sepal.Length, fill=Species)) +
  geom_histogram(col="black") +
  facet_wrap( ~ Species, ncol=1) +
  labs(x="Sepal Length (cm)", title="Iris flower dataset")
  theme_bw() + theme(legend.position="none")
```



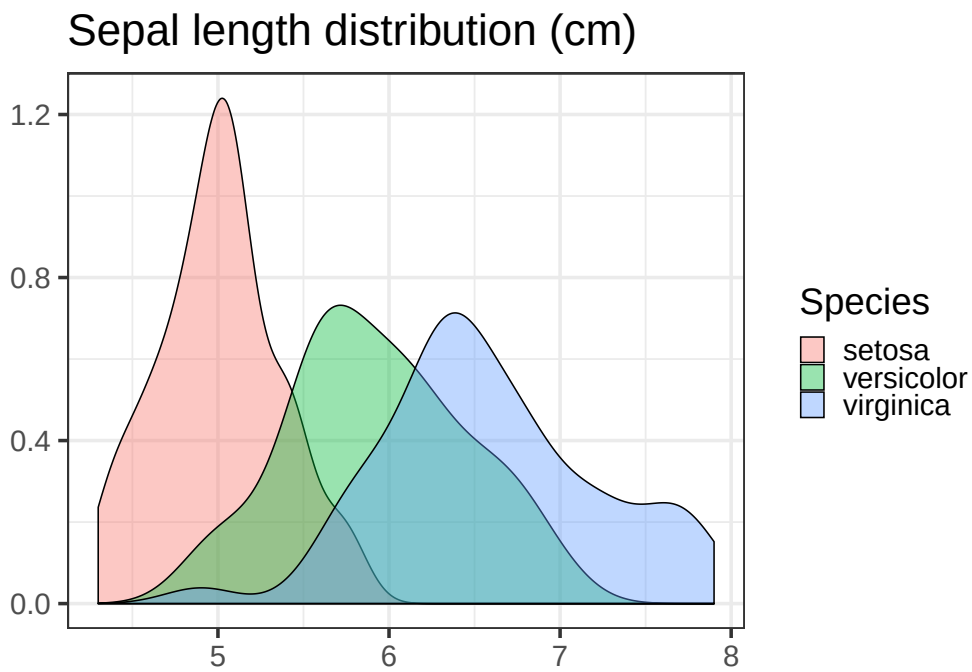
ggplot2 | Density

Reproduce the following using `geom_density()` instead.



ggplot2 | Density

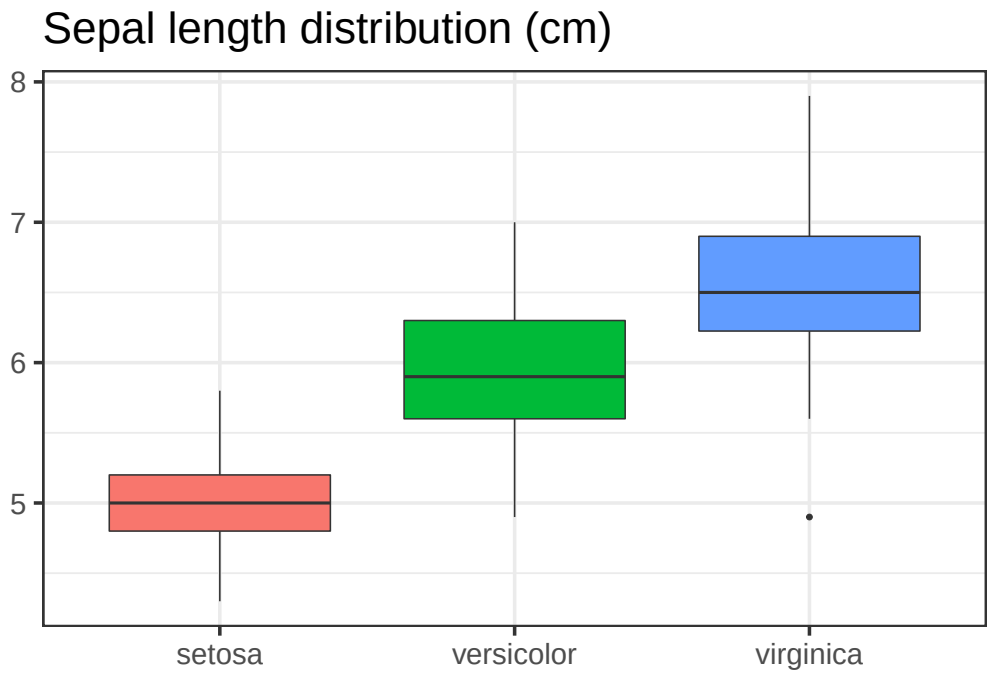
Reproduce the following using `geom_density()` instead.




```
ggplot(iris, aes(x=Sepal.Length, fill=Species)) +
  geom_density(alpha=0.4) +
  labs(x="", y="", title="Sepal length distribution (cm)") +
  theme_bw()
```

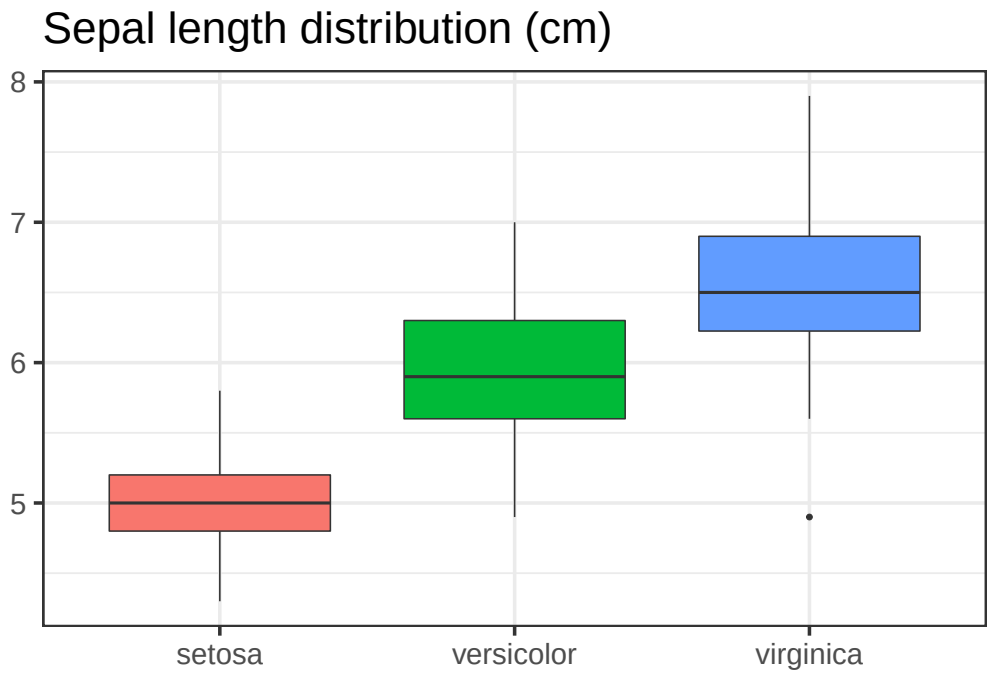
ggplot2 | Boxplot

Reproduce the following. What is the x-axis and y-axis?



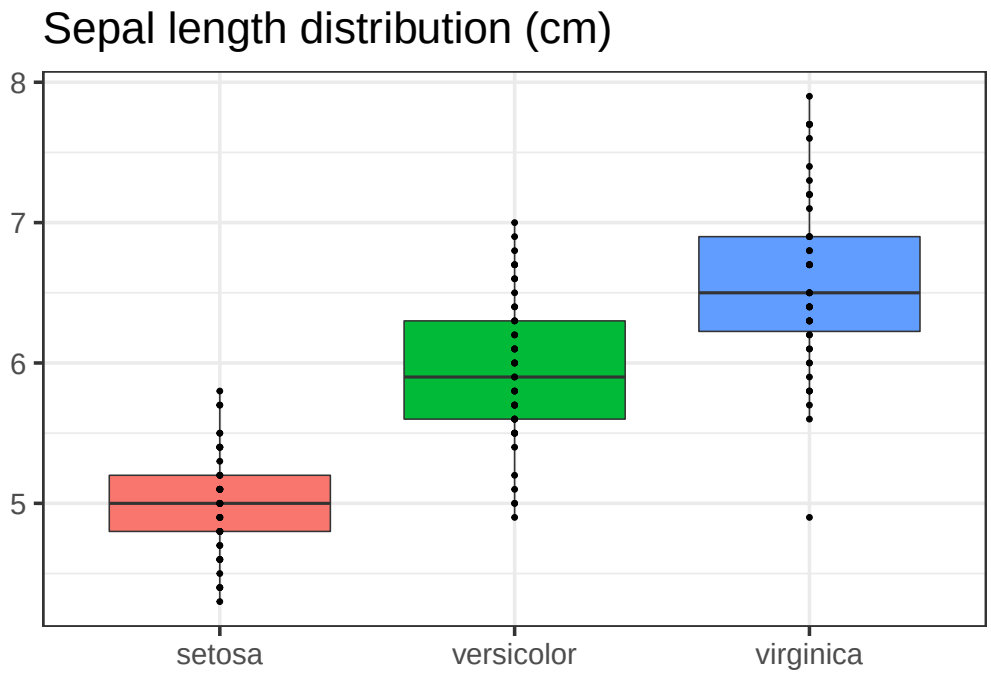
ggplot2 | Boxplot

Reproduce the following. What is the x-axis and y-axis?




```
ggplot(iris, aes(x=Species, y=Sepal.Length, fill=Species))
  geom_boxplot() +
  labs(x="", y="", title="Sepal length distribution (cm)")
  theme_bw() + theme(legend.position="none")
```

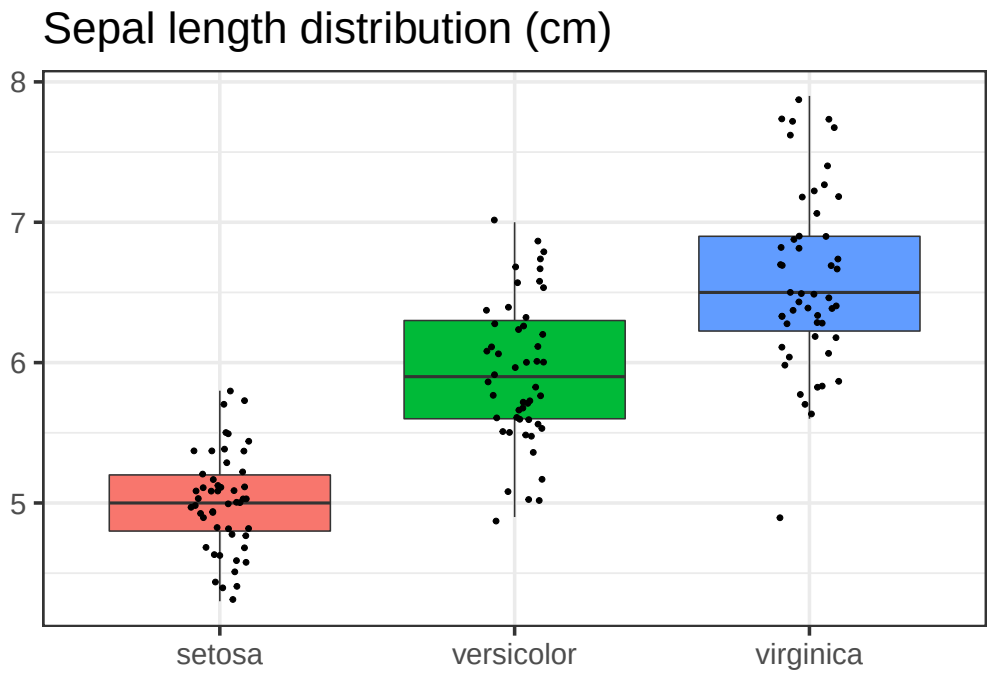
ggplot2 | Boxplot + points




```
gb <- ggplot(iris, aes(x=Species, y=Sepal.Length, fill=Spec
  geom_boxplot(outlier.shape=NA) +  # prevents double pts
  labs(x="", y="", title="Sepal length distribution (cm)")
  theme_bw() + theme(legend.position="none")
gb + geom_point()
```

Yuck! This is a terrible plot. Let's improve on it with jitter.

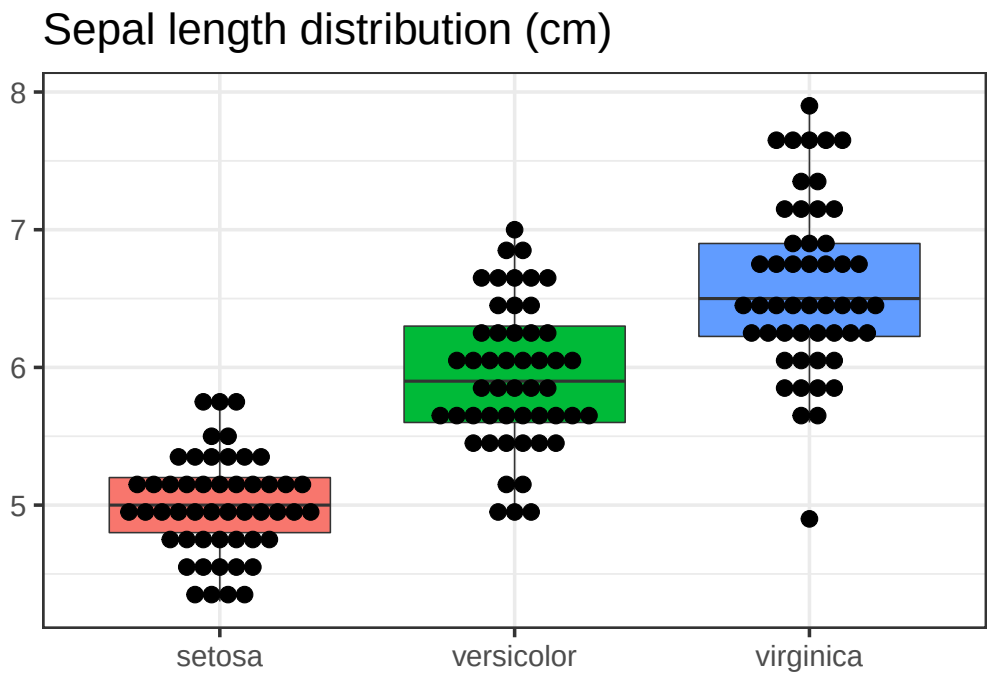
ggplot2 | Boxplot + jittered points




```
gb + geom_jitter(width=0.1)
rm(gb)
```

The order of geom_boxplot and geom_point / geom_jitter matters!

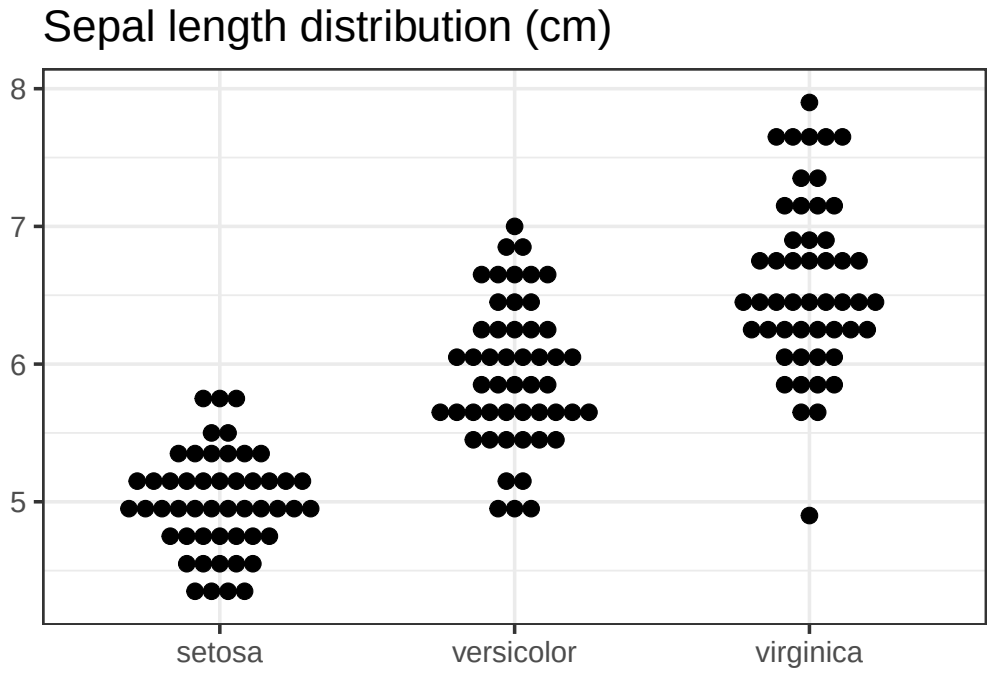
ggplot2 | Boxplot + dotplot




```
gb +
 geom_dotplot(binaxis="y", stackdir="center", fill="black")
rm(gb)
```

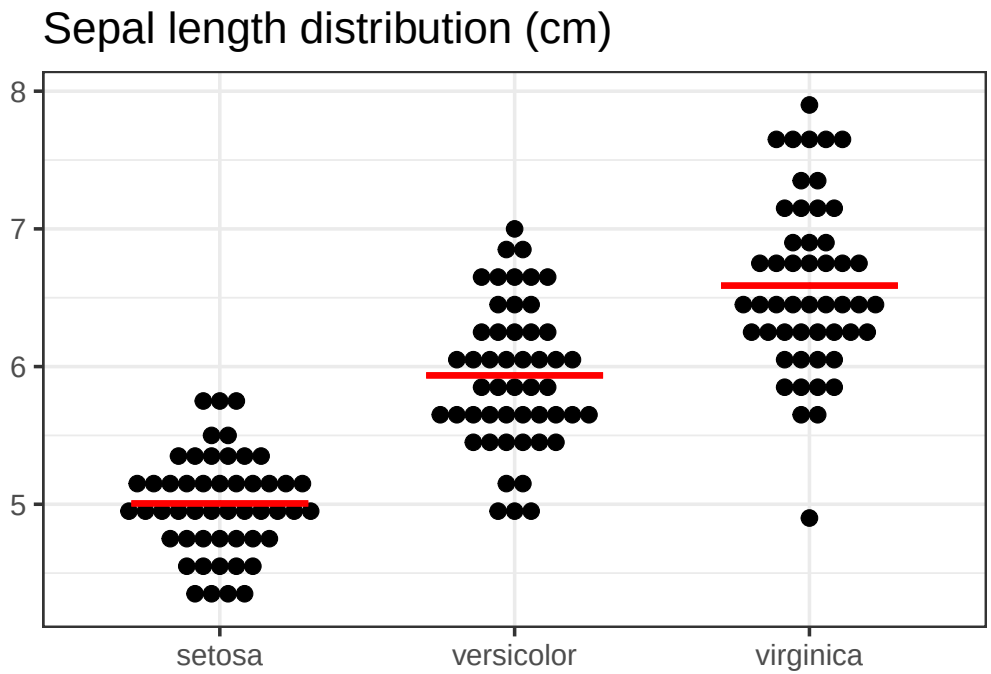
If you have very few data points, you can omit the boxplot entirely.

ggplot2 | Dotplot points




```
gd <- ggplot(iris, aes(x=Species, y=Sepal.Length)) +
 geom_dotplot(binaxis="y", stackdir="center") +
 labs(x="", y="", title="Sepal length distribution (cm)")+
 theme_bw() + theme(legend.position="none")
print(gd)
```

ggplot2 | Dotplots with mean and sd

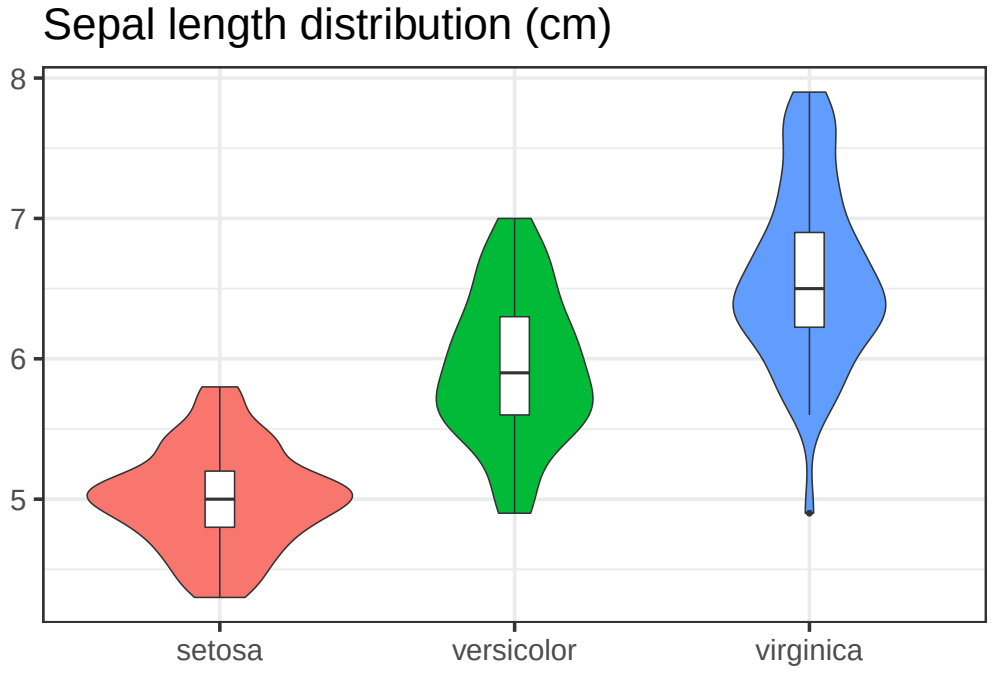



```
gd+ stat_summary(fun.data=mean_sdl, fun.args=list(mult=0),
                 geom="errorbar", col="red", width=0.6)
```

The red horizontal line represents the mean.

- Set `mult=1` if you want to display mean $\pm$ sd.
- Set `mult=2` if you want to display mean $\pm$ 2 sd.

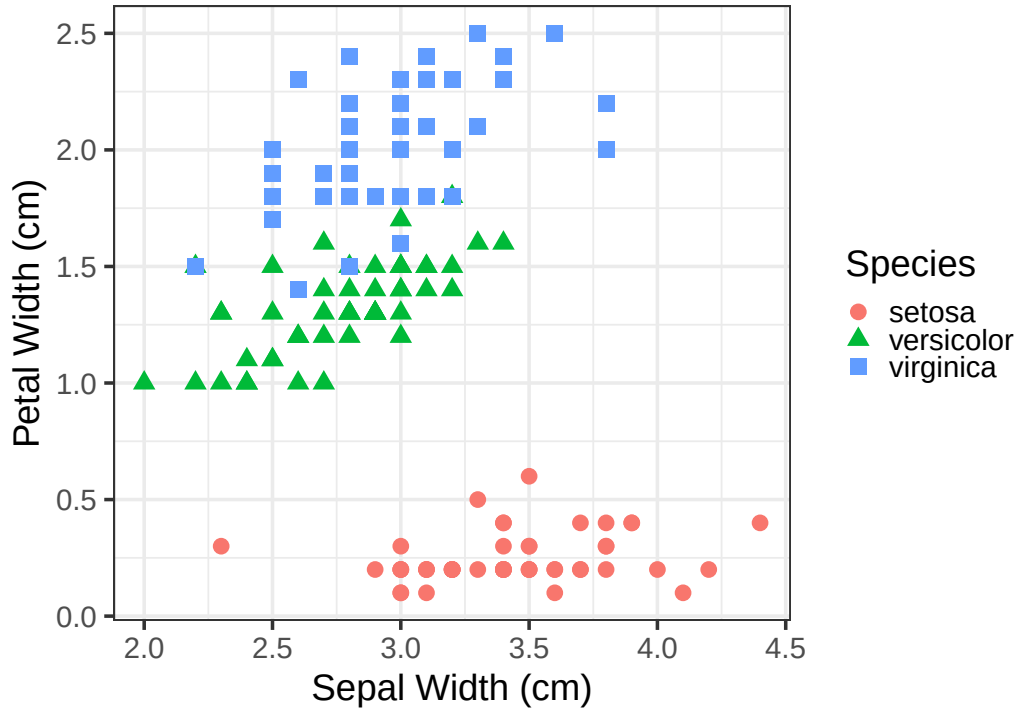
ggplot2 | Violinplot + boxplot




```
ggplot(iris, aes(x=Species, y=Sepal.Length, fill=Species))
  geom_violin() + geom_boxplot(width=0.1, fill="white") +
  labs(x="", y="", title="Sepal length distribution (cm)")
  theme_bw() + theme(legend.position="none")
```

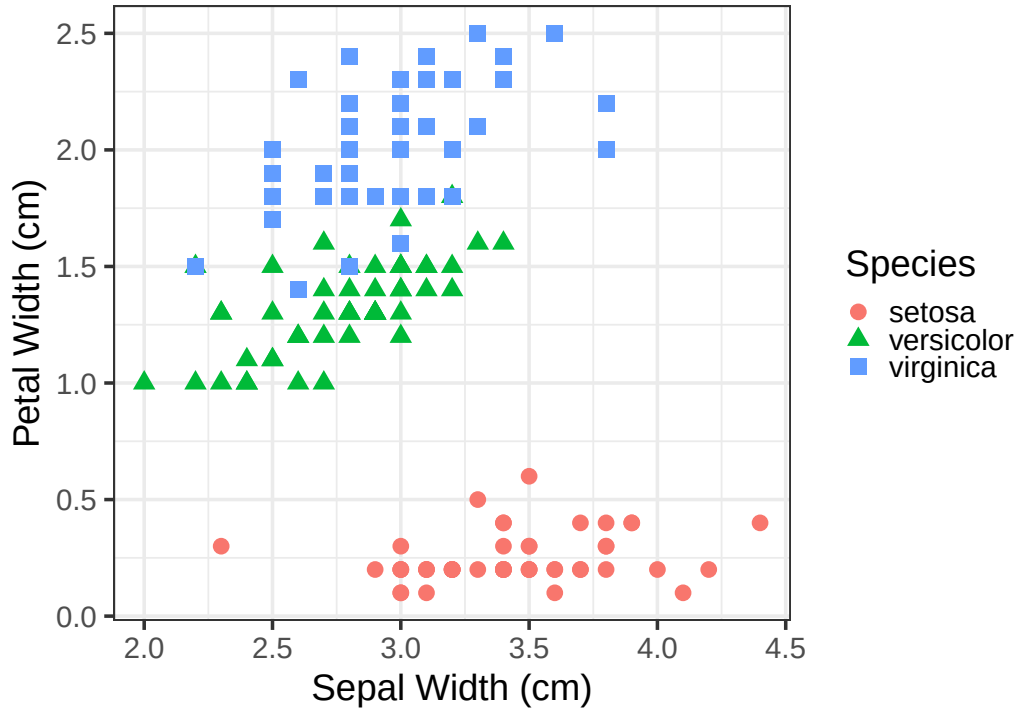
ggplot2 | Scatterplot

Reproduce the following.



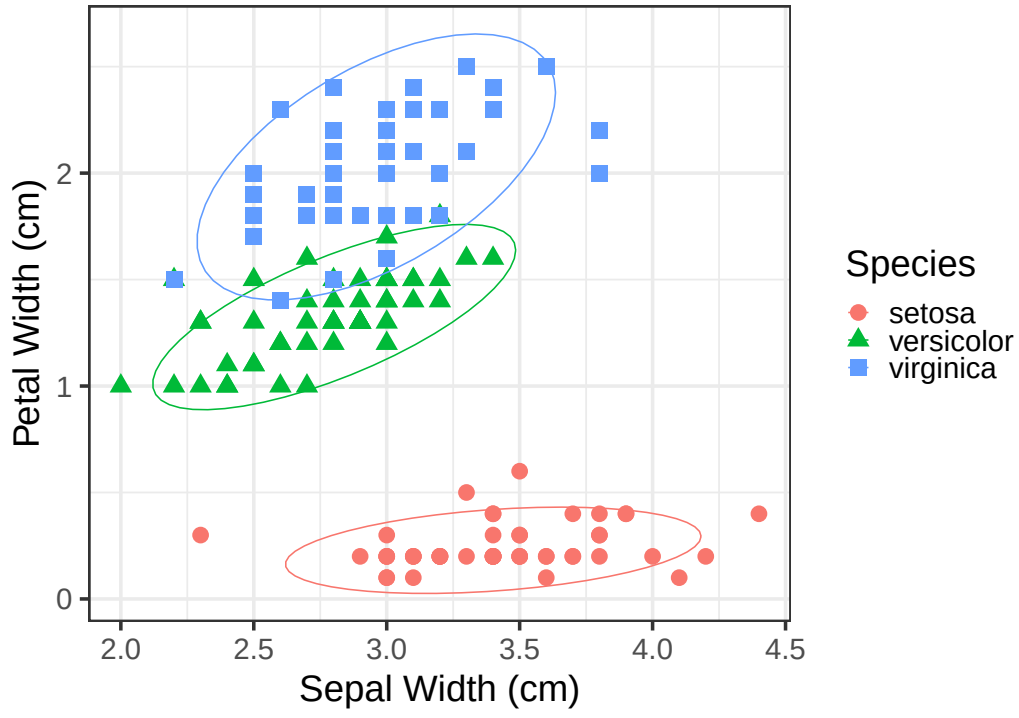
ggplot2 | Scatterplot

Reproduce the following.



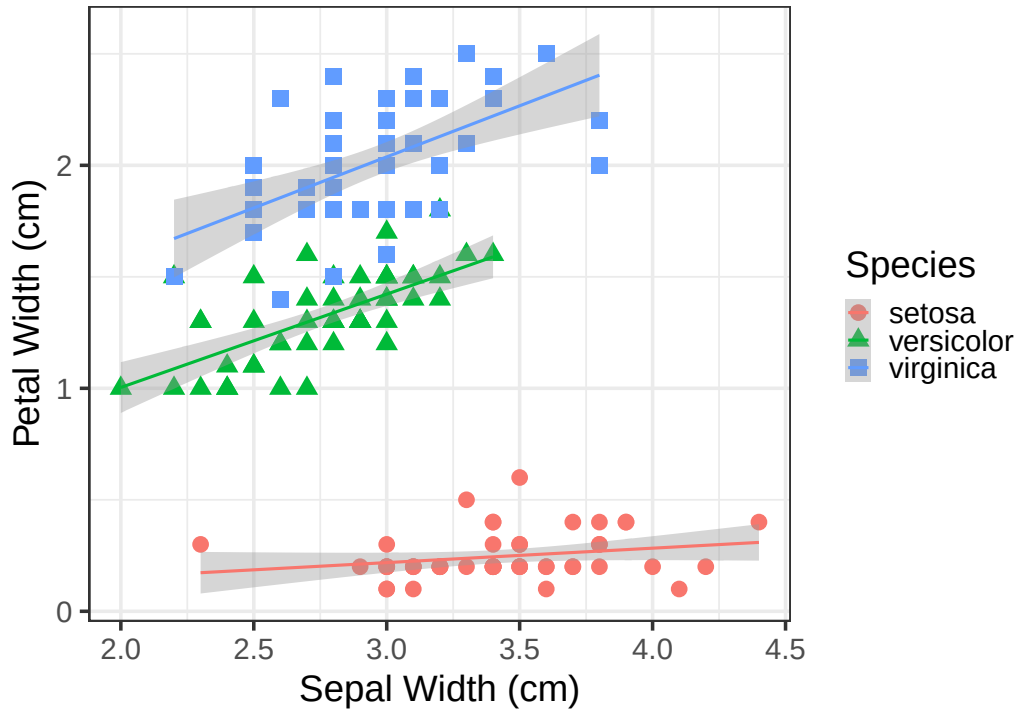

```
gs <- ggplot(iris, aes(x=Sepal.Width, y=Petal.Width,
                       col=Species, pch=Species)) +
  geom_point() + theme_bw() +
  labs(x="Sepal Width (cm)", y="Petal Width (cm)")
print(gs)
```

ggplot2 | Scatterplot + stat_ellipse



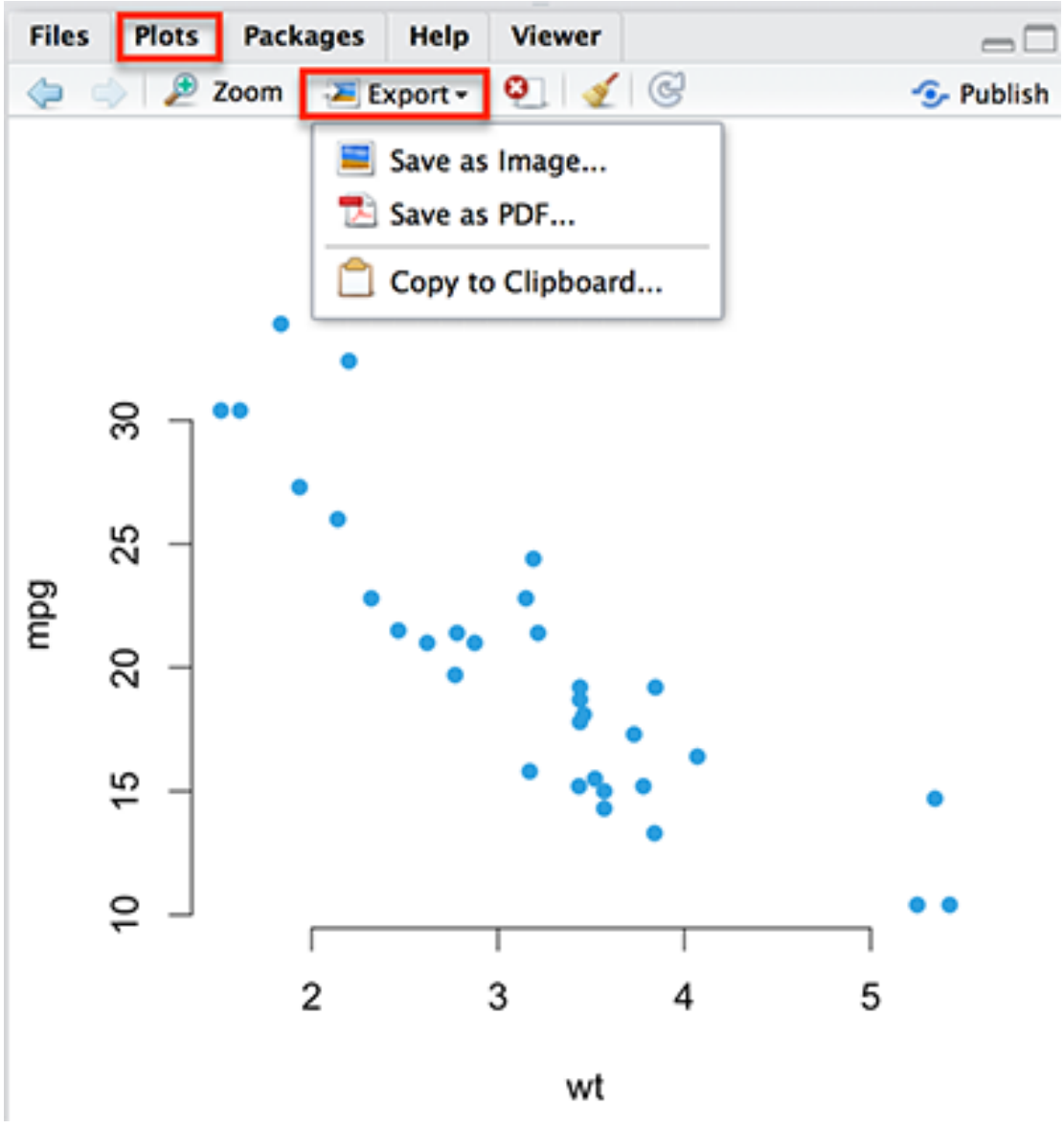

```
gs + stat_ellipse()
```

ggplot2 | Scatterplot + geom_smooth




```
gs + geom_smooth(method="lm")
rm(gs)
```

Saving plots | Manually



Saving plots | Programmatically

```
pdf("density.pdf")  ## Multi-page PDF

ggplot(iris, aes(x=Sepal.Length, fill=Species)) +
  geom_density(alpha=0.4)

ggplot(iris, aes(x=Petal.Length, fill=Species)) +
  geom_density(alpha=0.4)

dev.off()
## pdf
##   2
```

Can also use `bitmap()`, `jpeg()`, `png()`, `tiff()` but these do not have multipage support.

SUMMARY

What we learnt today

Programming aspects:

- R and Rstudio
- R as a calculator
- Scalars and assignment
- Scripting
- Control flow

Visualization:

- continuous vs categorical (histogram, density, boxplot)
- continuous vs continuous (scatterplots)

Statistics:

- Exponentian

What we will learn next week

- Review of assignment
- Data manipulation and summary using tidyverse
- Basic statistics (t-test)
- Basic probability
- Other data types briefly