Today's program

18:30 – 18:45	Walk-in
18:45 – 20:30	Workshop
20:30 – end	Networking

Welcome!

Worldwide organization

that promotes **diversity** in the

#rstats community via meetups

and mentorship in a **friendly** and

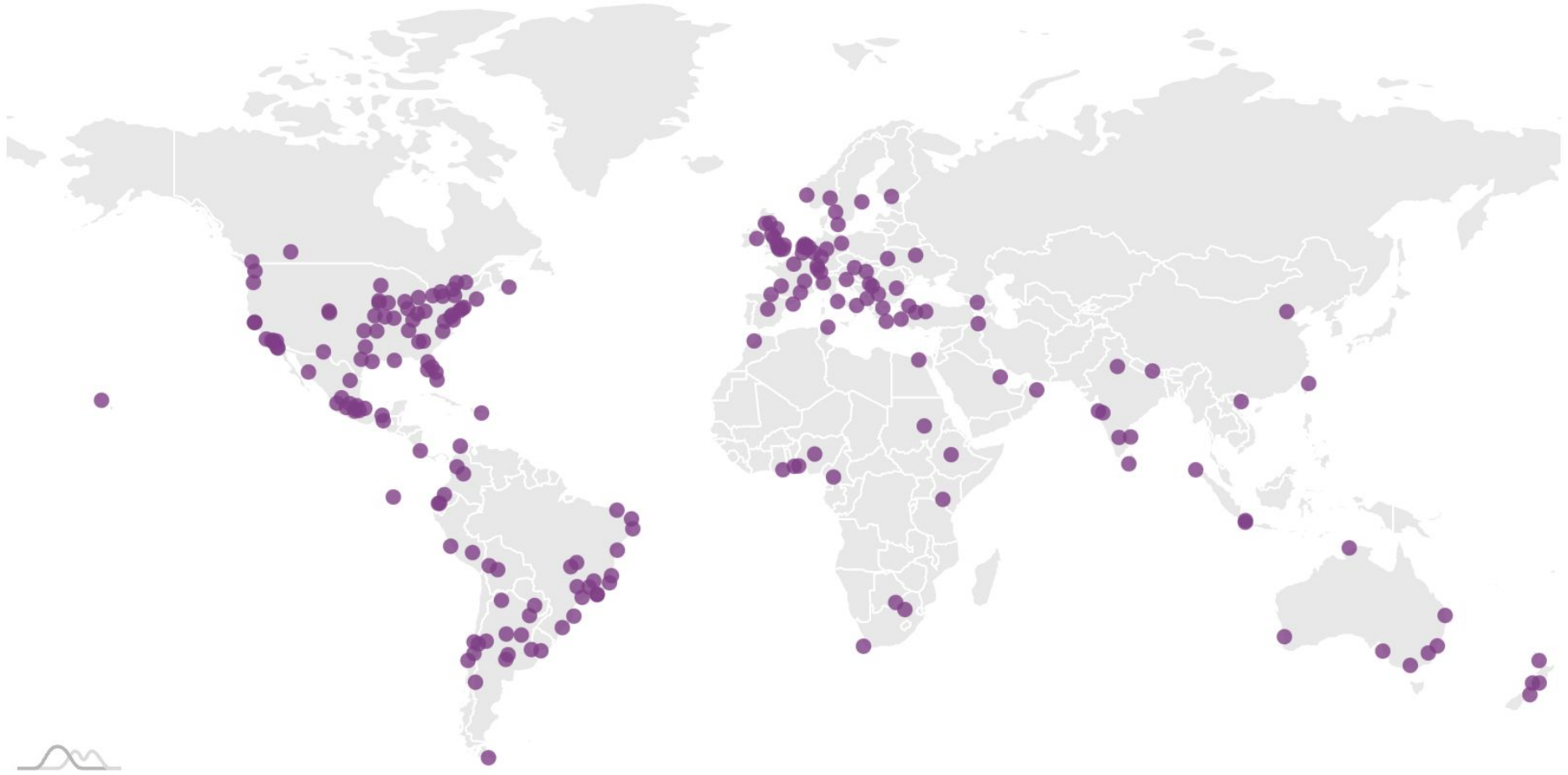
safe environment

Our mission

- more minority-genders coders
- more minority-genders developers
- more minority-genders speakers
- more minority-genders leaders

more minority-genders developing and creating R Packages and being involved in the R community

R-Ladies can be found all over the world!



Countries
63

Events
3943

Chapters
219

Code of Conduct

Everyone participating must agree on the code of conduct.

> *Excerpt:*
> R-Ladies is dedicated to providing a harassment-free experience for everyone. We do not tolerate harassment of participants in any form.

Read the full version here:
https://rladies.org/code-of-conduct/

Ensures that environment of meetups stay **safe** and **friendly!**

UNIVERSITY
OF AMSTERDAM

Thank you for hosting this event!

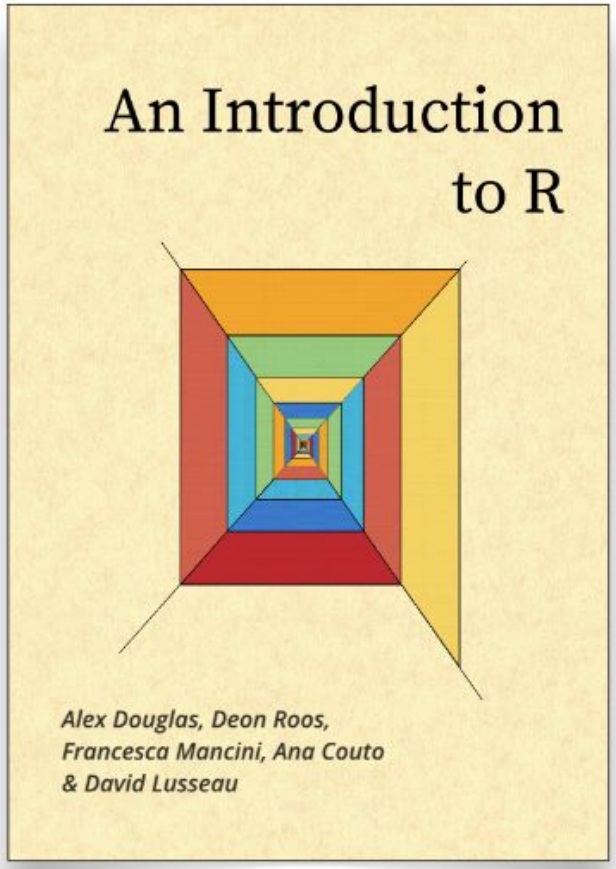
```r
library(dplyr)

rladies_global %>%
  filter(city == 'Your city')
```

Introduction to R

Getting Started

Introduction to R

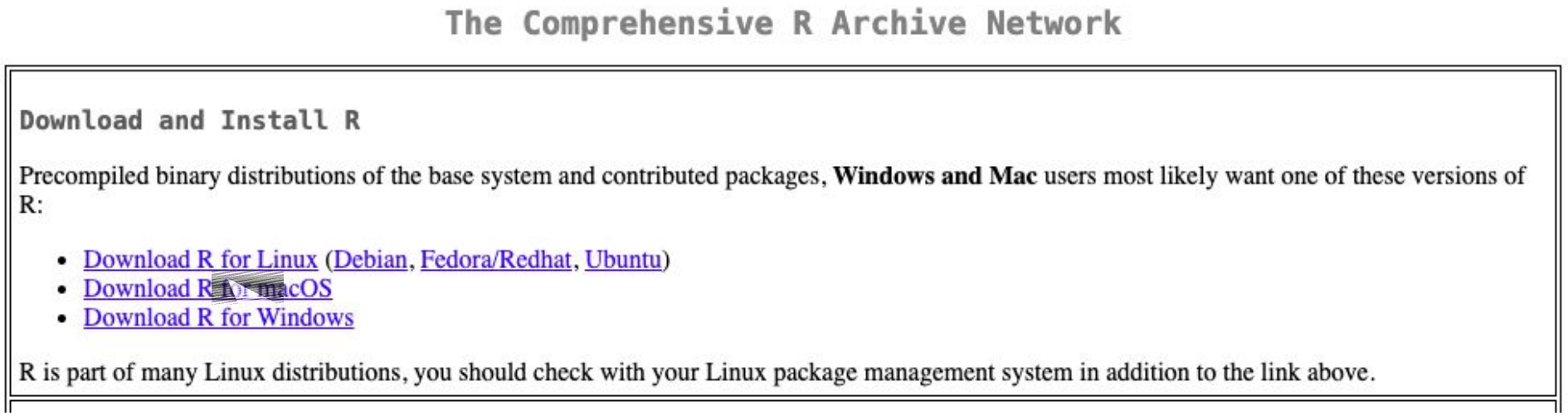
For this workshop, we will follow the book *An Introduction to R* by *A*lex Douglas, Deon Roos, Francesca Mancini, Ana Couto & David Lusseau



The book has a corresponding website: https://alexd106.github.io/intro2R/index.html

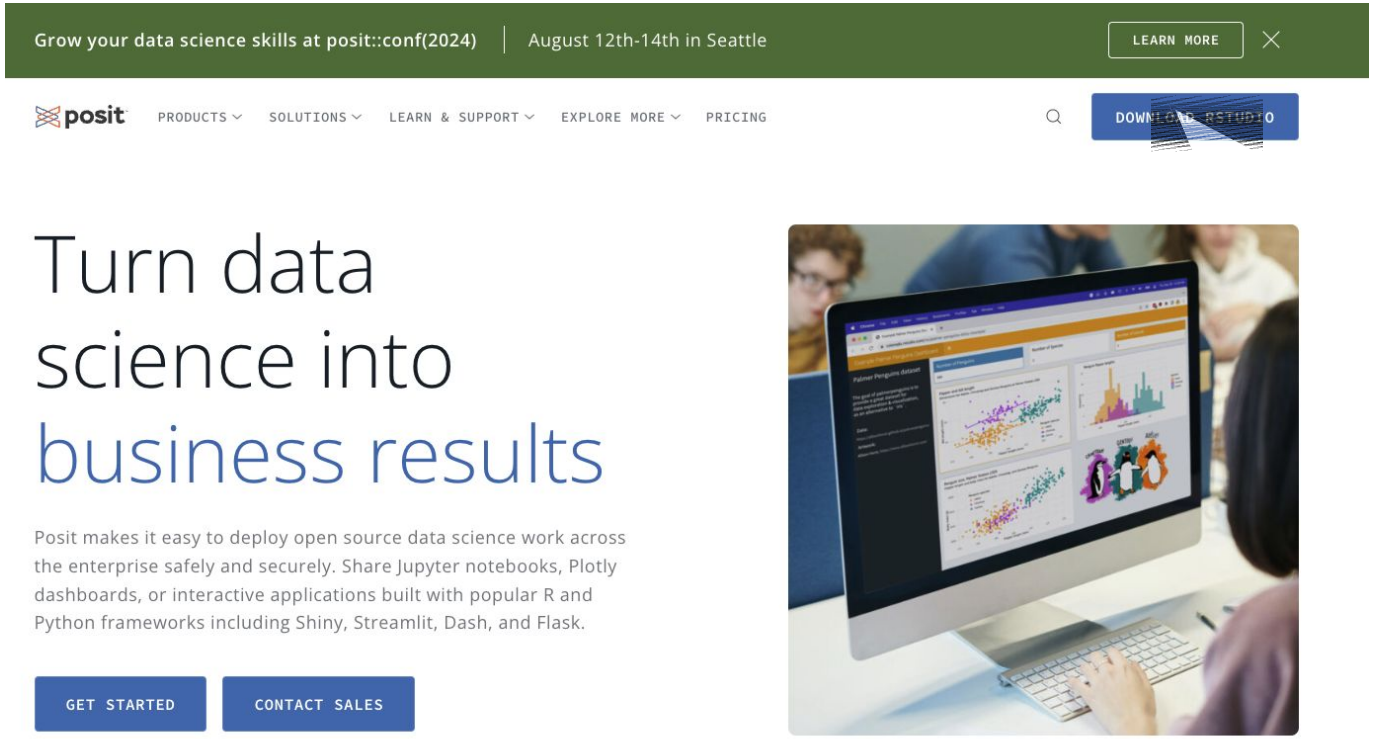
Installing R

To install R, visit the Comprehensive R Archive Network (CRAN) website: https://cran.r-project.org/

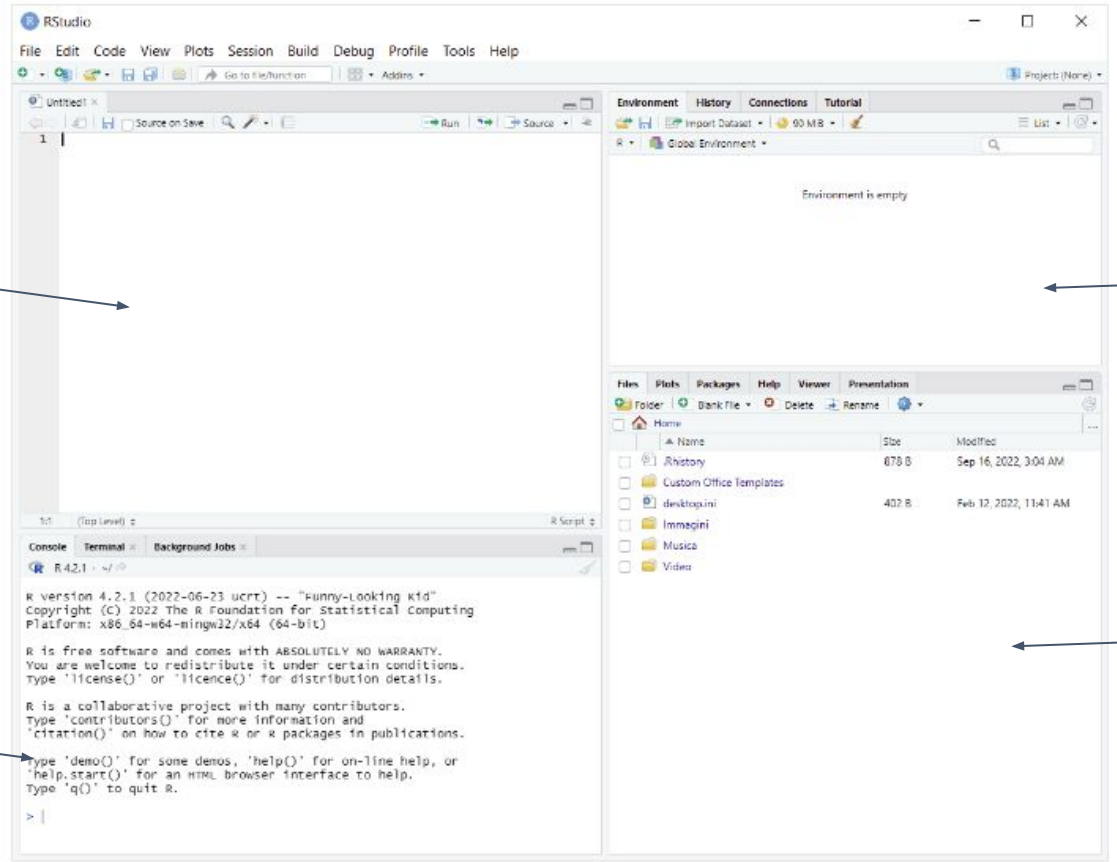


Installing RStudio

RStudio can be downloaded by visiting https://posit.co/ and clicking on “Download RStudio” in the top right corner.



RStudio Orientation



source

environment

directory/plot/
package viewer

console

*You can change the pane layout in the RStudio menu: View -> Panes -> Pane Layout

R Basics

R is a fancy calculator

```
> ### 2.1. R is a fancy calculator
> # R can perform arithmetic operations
> # For example, if we type the expression 2 + 2 and then source this line of code
> # we get the answer 4
> 2 + 2
[1] 4
> # Let's try a more complicated computations. R follows the usual mathematical
> # convention of order of operations.
> 2 + 3 * 4
[1] 14
> (2 + 3) * 4
[1] 20
>
```

Objects

- Almost everything in R is an *object*
- To create an object we:
  - Give it a name
  - Assign a value using the *assignment operator*: <-

```
> my_obj <- 48
> my_obj
[1] 48
> 
```

- Naming objects is hard!
  - Ideally names should be short and informative
  - Names cannot start with a number or “.”

Using functions in R

- You can think of a function as an object which contains a series of instructions to perform a specific task.
- The base installation of R comes with many functions

The **c()** function is short for concatenate and we use it to join together a series of values and store them in a data structure called a vector.

```
my_vec <- c(2, 3, 1, 6, 4, 3, 3, 7)
```

We can apply many other functions to my_vec, including **mean()**, **var()**, **sd()**, and **length()**

Vectors in R

- Extracting elements
  - *my_vec[1]* extracts the first element
- Replacing elements
  - *my_vec[1] <- 3* reassigns the first element of my_vec to the number 3
- Ordering elements
  - **sort()**, **order()**
- Vectorisation:
  - *my_vec*2* multiplies every element in my_vec by 2
- Missing elements

Getting Help

- You can search for more information on R functions within R
- There is detailed documentation for all (base) R functions which can be accessed by typing **?<function_name>** in the console (replace “<function_name>” with the correct function).

Data in R

Data Types

- **Numeric** are numbers that contain a decimal
- **Integers** are numbers without a decimal
- **Logicals** are TRUE or FALSE
- **Characters** represent string values, these are usually words or letters
- **Factors** are a specific type of character string

Data Types

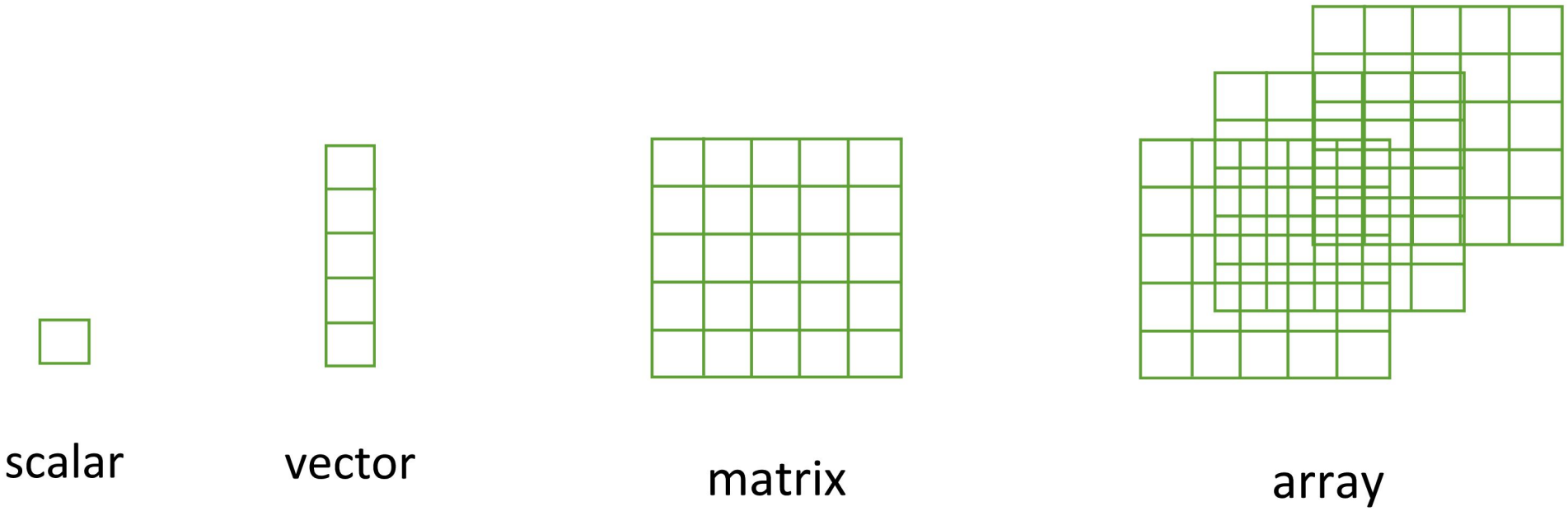
- We can find out what type of data we are dealing with by using the **class()** function
- Alternatively we can use functions like **is.numeric()** or **is.logical()** to figure out if our data is of a specific type
- We can also change data into a desired type by using functions like **as.numeric()**

Data Types

Type	Logical test	Coercing
Character	`is.character`	`as.character`
Numeric	`is.numeric`	`as.numeric`
Logical	`is.logical`	`as.logical`
Factor	`is.factor`	`as.factor`
Complex	`is.complex`	`as.complex`

Data Structures

- Data comes in different structures, the simplest being a vector



Data Structures

- Another useful type of data structure is the list
- Lists can store different types of data (unlike vectors and matrices)

```
list_1 <- list(c("black", "yellow", "orange"),
               c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE),
               matrix(1:6, nrow = 3))
list_1
## [[1]]
## [1] "black"  "yellow" "orange"
##
## [[2]]
## [1]  TRUE  TRUE FALSE  TRUE FALSE FALSE
##
## [[3]]
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Data Structures

- By far the most commonly used data structure is the data frame which we can create using the **data.frame()** function

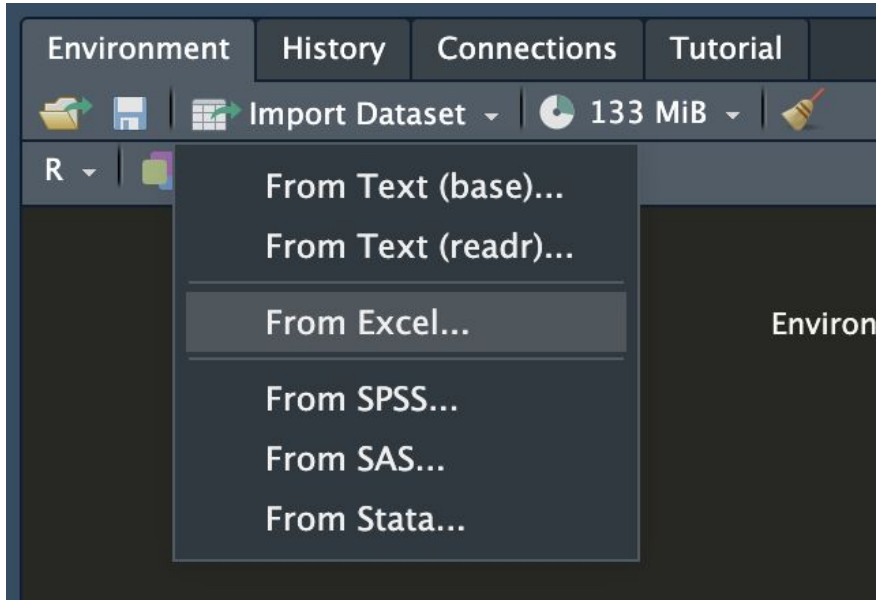
	treat	nitrogen	block	height	weight	leafarea	shootarea	flowers
1	tip	medium	1	7.5	7.62	11.7	31.9	1
2	tip	medium	1	10.7	12.14	14.1	46.0	10
3	tip	medium	1	11.2	12.76	7.1	66.7	10
4	tip	medium	1	10.4	8.78	11.9	20.3	1
5	tip	medium	1	10.4	13.58	14.5	26.9	4
6	tip	medium	1	9.8	10.08	12.2	72.7	9

Importing Data

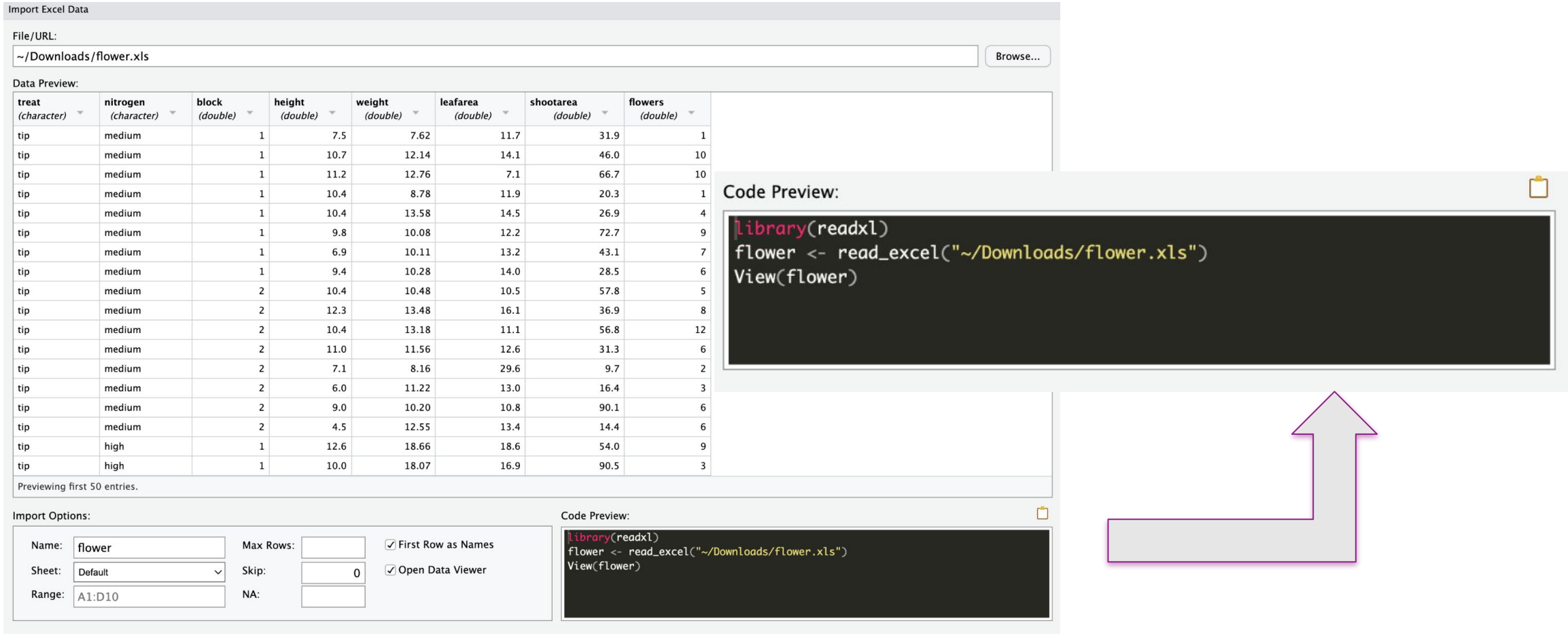
- We can import data using import functions such as **read.table()**
- The function you use depends on the format of your data file (e.g., **read.csv()**, **read.xlsx()**...)
- You can also use the GUI to import a data set which may be easier in the beginning

Importing Data

- Now it's time to import our first data set: **flower.xls** (Download here: https://alexd106.github.io/intro2R/data/flower.xls)



Importing Data



Wrangling Data Frames

- You can access variables in a dataset using the **$ operator**, e.g., *flowers$nitrogen*
- The **str()** function shows you the structure of the data set
- To access specific cells, you can use positional indexing through **squared brackets**, e.g., *flowers[1:10, 3]*

```
> str(flowers)
tibble [96 × 8] (S3: tbl_df/tbl/data.frame)
 $ treat    : chr [1:96] "tip" "tip" "tip" "tip" ...
 $ nitrogen : chr [1:96] "medium" "medium" "medium" "medium" ...
 $ block    : num [1:96] 1 1 1 1 1 1 1 1 2 2 ...
 $ height   : num [1:96] 7.5 10.7 11.2 10.4 10.4 9.8 6.9 9.4 10.4 12.3 ...
```

Summarizing Data

- You can use the **summary()** function to summarize a data set or individual variables
- For numerical data you can use simple statistical functions like **mean()** and **sd()**
- You can access counts for categorical data using the **table()** function

```
table(flowers$nitrogen)
##
##    low medium   high
##     32     32     32
```

Exporting Data

- We can export data using export functions such as **write.table()**
- Analogous to import functions, these depend on the data format we want to use

```
> write.table(flowers, file = 'flowers.txt', col.names = TRUE,
+             row.names = FALSE, sep = "\t")
```

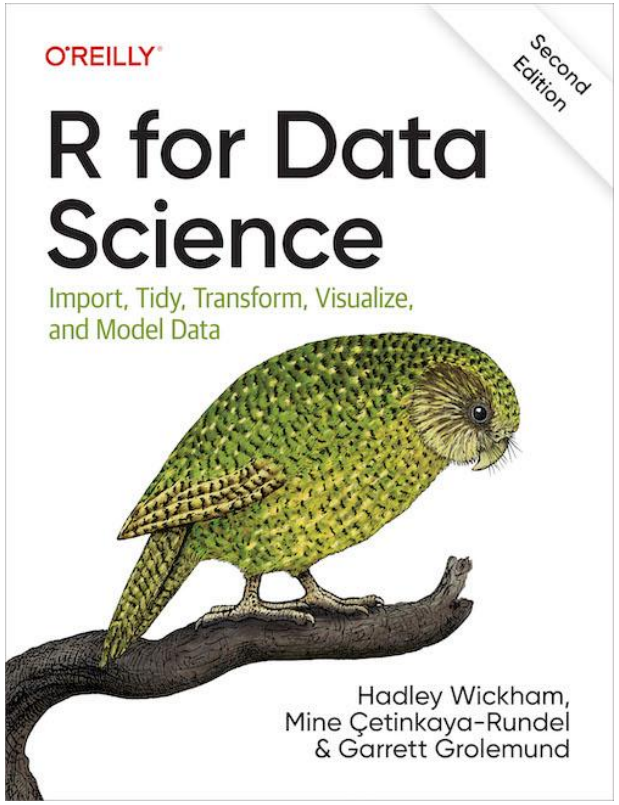
flowers.txt

"treat"	"nitrogen"	"block"	"height"		"weight"		"leafarea"	"shootarea"	"flowers"
"tip"	"medium"	1	7.5	7.62	11.7	31.9	1		
"tip"	"medium"	1	10.7	12.14	14.1	46	10		
"tip"	"medium"	1	11.2	12.76	7.1	66.7	10		
"tip"	"medium"	1	10.4	8.78	11.9	20.3	1		
"tip"	"medium"	1	10.4	13.58	14.5	26.9	4		
"tip"	"medium"	1	9.8	10.08	12.2	72.7	9		
"tip"	"medium"	1	6.9	10.11	13.2	43.1	7		
"tip"	"medium"	1	9.4	10.28	14	28.5	6		
"tip"	"medium"	2	10.4	10.48	10.5	57.8	5		
"tip"	"medium"	2	12.3	13.48	16.1	36.9	8		
"tip"	"medium"	2	10.4	13.18	11.1	56.8	12		

If you want to know more…

- If you want to learn R to process and analyze data, you might want to learn more about the **tidyverse**

- The book **R for Data Sciene** may be a good place to start (freely available here: https://r4ds.hadley.nz/)



Visualizing Data in R: Graphics with Base R

"The simple graph has brought more information to the data analyst's mind than any other device." — John Tukey

Base R plots

- Base R allows you to quickly plot simple visualizations
- Generally, your plots will most likely be composed of two parts:
  - **High-level plotting function:** functions such as **plot()** which create your plot
  - **Low-level customizing functions:** while **plot()** let's you quickly graph your data, you will need supporting functions such as **lines()** or **text()** to customize the plot

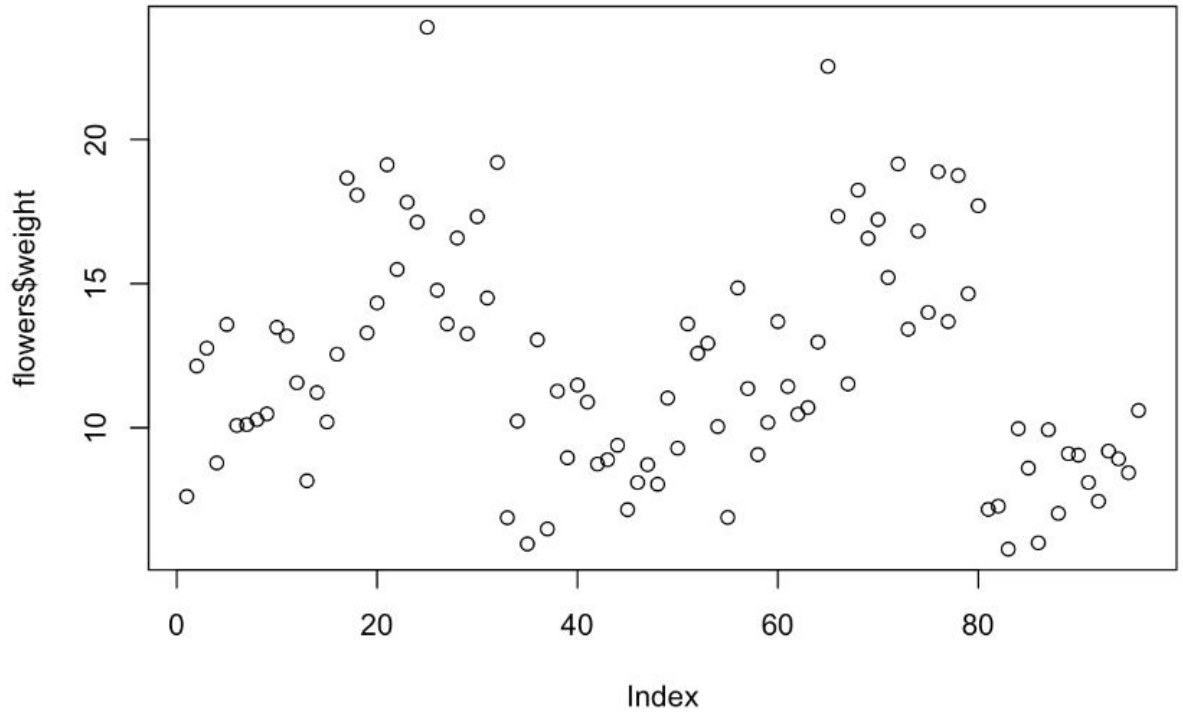
The simplest plot

- **plot()** - produces the most commonly used **scatterplot**

Creates a data table for the plot

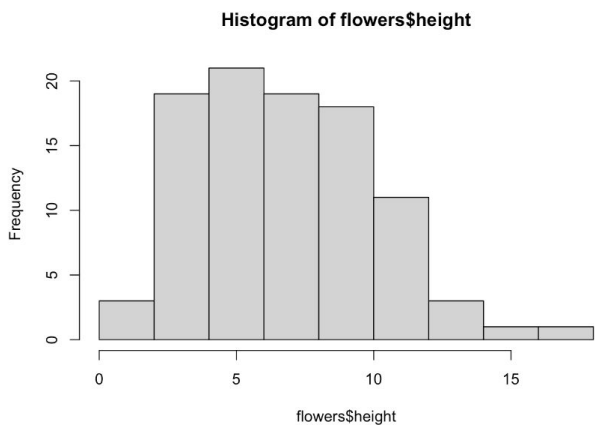
```
flowers <- read.table(file = 'data/flower.txt',
                      header = TRUE, sep = "\t",
                      stringsAsFactors = TRUE)

plot(flowers$weight)
```

Creates a plot for the *weight* variable in data

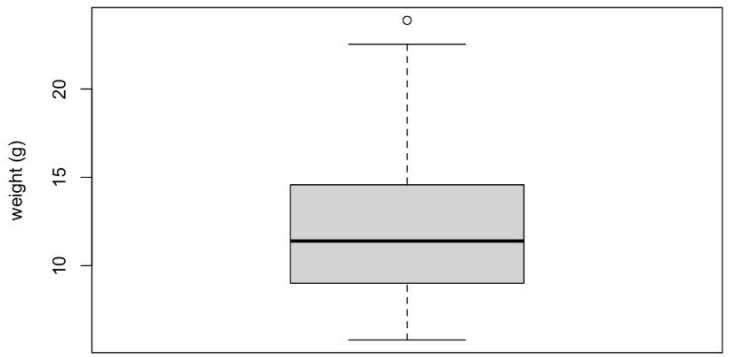


Other simple plotting functions

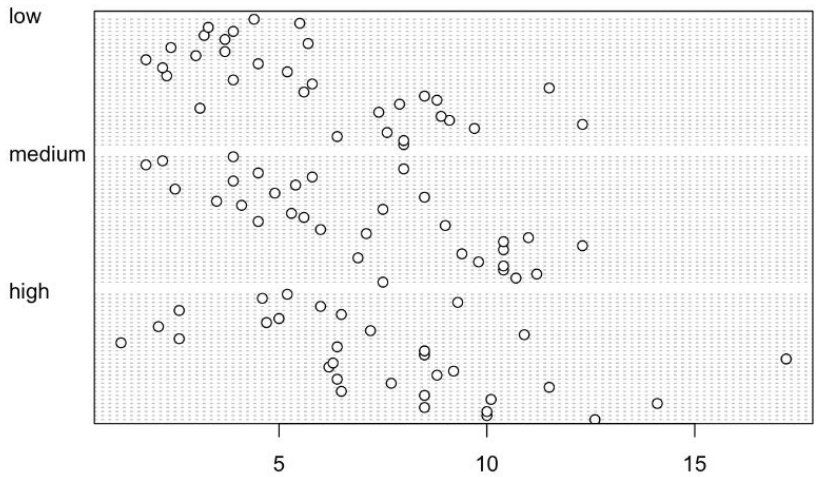
- **hist()** - creates **histograms**
- **boxplot()** - creates a boxplot
- **dotchart()** - creates a dotchart, helpful in identifying outliers



hist(flowers$height)



boxplot(flowers$weight, ylab = "weight (g)")



dotchart(flowers$height, groups = flowers$nitrogen)

Customizing plots

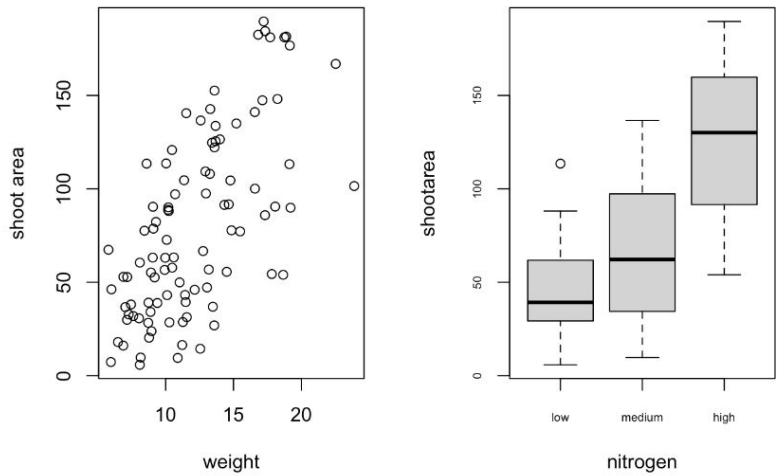
- There are two general ways of customizing plots:
  - Using **function arguments**:
    - E.g.: boxplot(flowers$weight, **ylab = "weight (g)"**)

  - Building further **layers** of the plot (adding supporting functions outside of the plotting function)
    - **points()**, **text()**, **lines()**, etc.

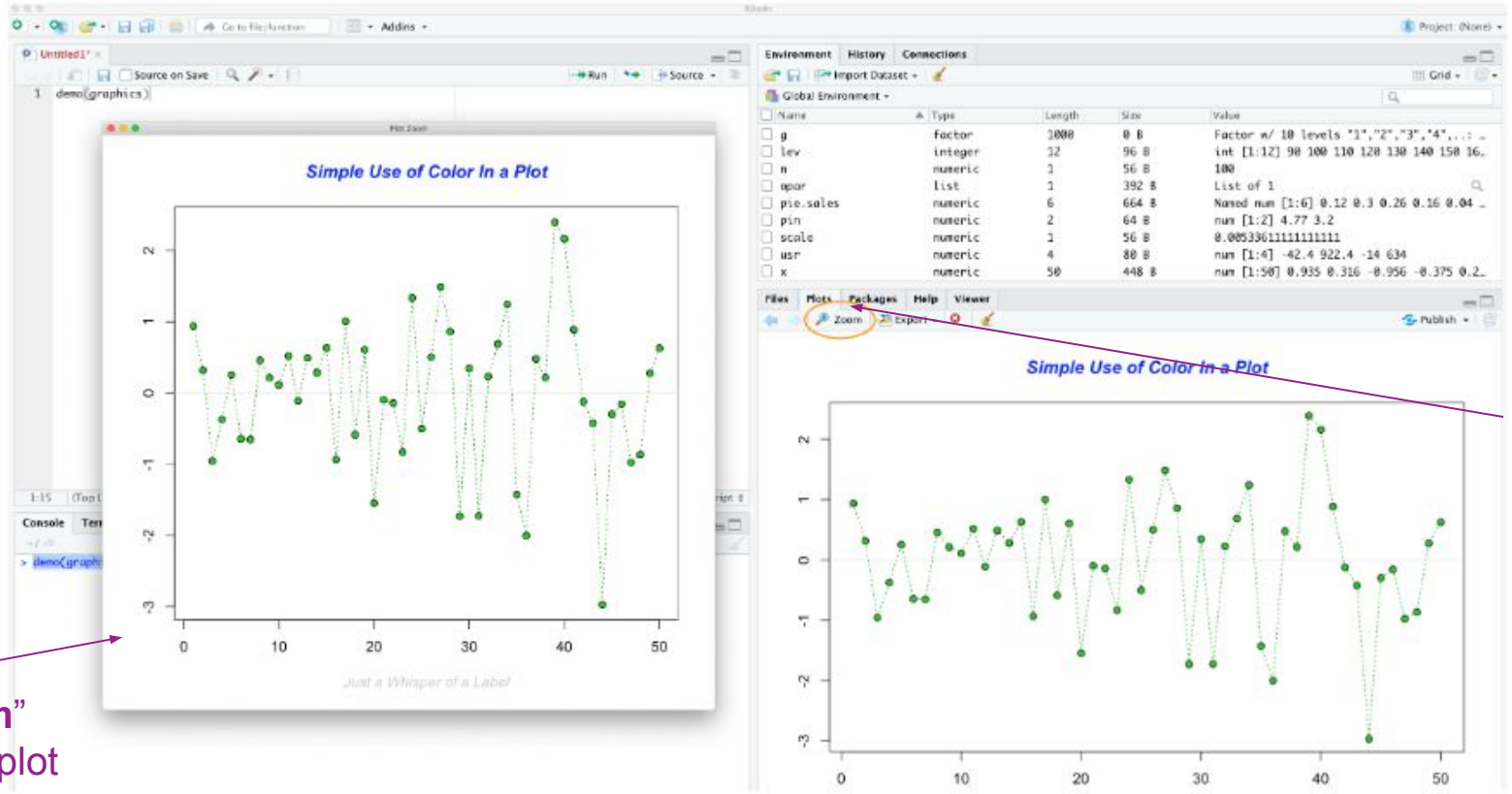
Multiple graphs

- It is possible to create multiple graphs in the same plot using, e.g., **pairs()**, **coplot()**, **xyplot()**
- If you want to display multiple plots as panels in the same graphic, use **par()**
  - The *mfrow* argument allows you to configure the plots into rows and columns

```
par(mfrow = c(1, 2))
plot(flowers$weight, flowers$shootarea, xlab = "weight",
     ylab = "shoot area")
boxplot(shootarea ~ nitrogen, data = flowers, cex.axis = 0.6)
```



Plots in RStudio



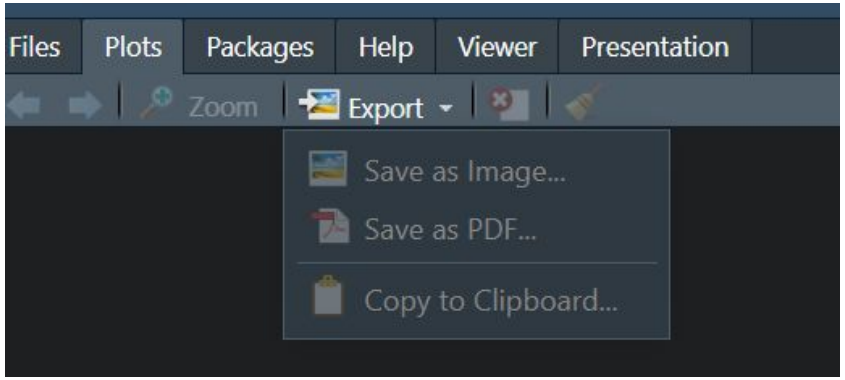
Clicking on the "**Zoom**" button will open your plot in a new window

All your plots will show up in the **Plots tab,** you can switch between all the created tabs using the arrows

Exporting plots

- One approach is to use the **Export** option in the Plots tab

- Another option is to save your plot using functions: **pdf()**, **jpeg()**, **png()**, etc.

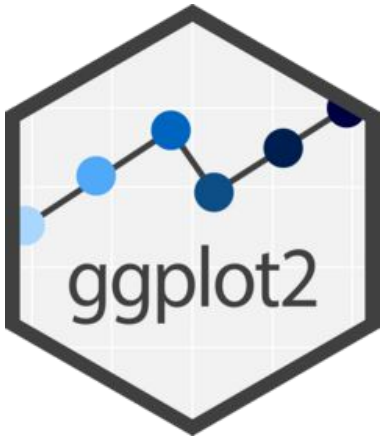



```
pdf(file = 'output/my_plot.pdf')
plot(flowers$weight, flowers$shootarea,
       xlab = "weight (g)")
dev.off()
```

Base R plots: Recap

```
pdf(file = 'output/my_plot.pdf')
par(mar = c(4.1, 4.4, 4.1, 1.9), xaxs="i", yaxs="i")
plot(flowers$weight, flowers$shootarea,
      xlab = "weight (g)",
      ylab = expression(paste("shoot area (cm"^"2",")")),
      xlim = c(0, 30), ylim = c(0, 200), bty = "l",
      las = 1, cex.axis = 0.8, tcl = -0.2,
      pch = 16, col = "dodgerblue1", cex = 0.9)
text(x = 28, y = 190, label = "A", cex = 2)
dev.off()
```

Visualizing with ggplot2
From basic to fancy

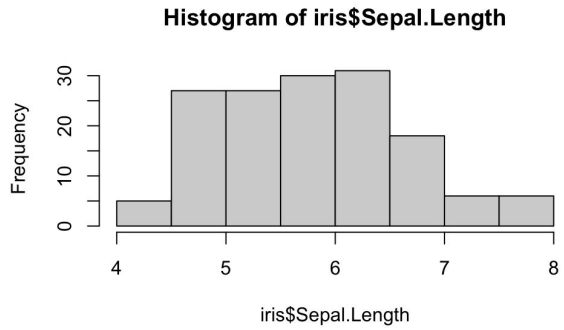


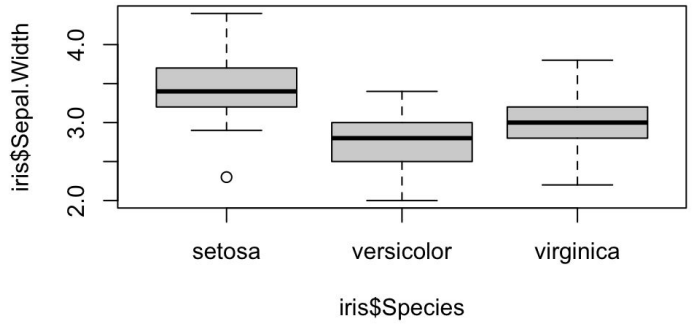
“ggplot2 embodies a deep philosophy of visualisation”

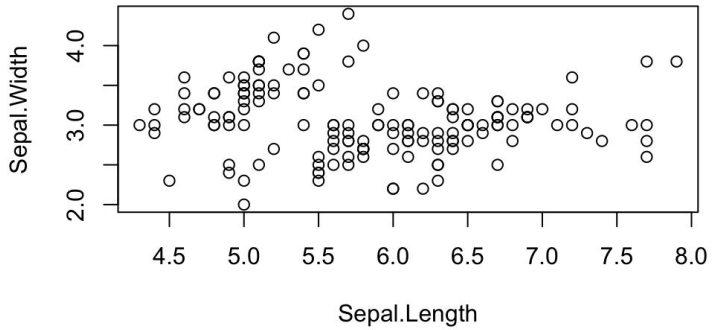
Same data, different visualization

data(iris)

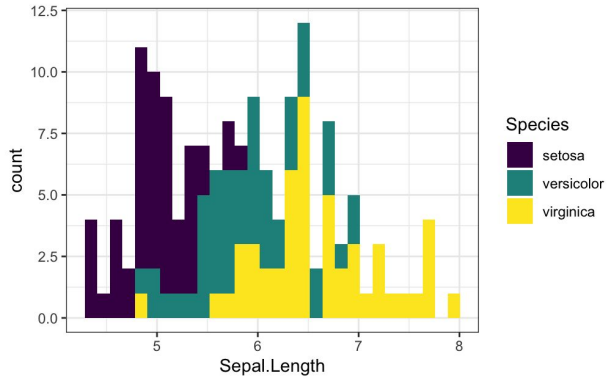
base R plotting

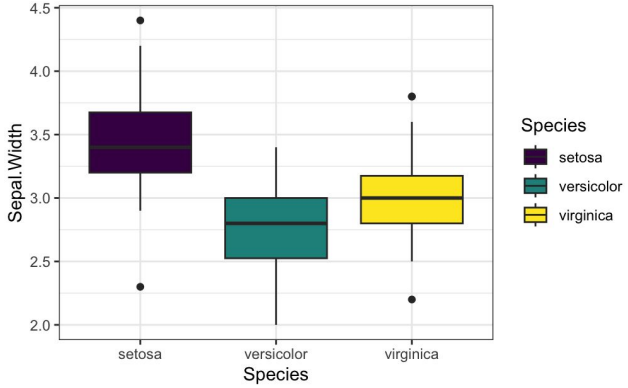


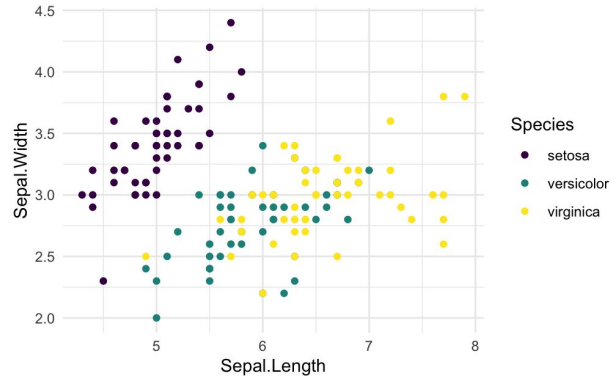




ggplot-ting





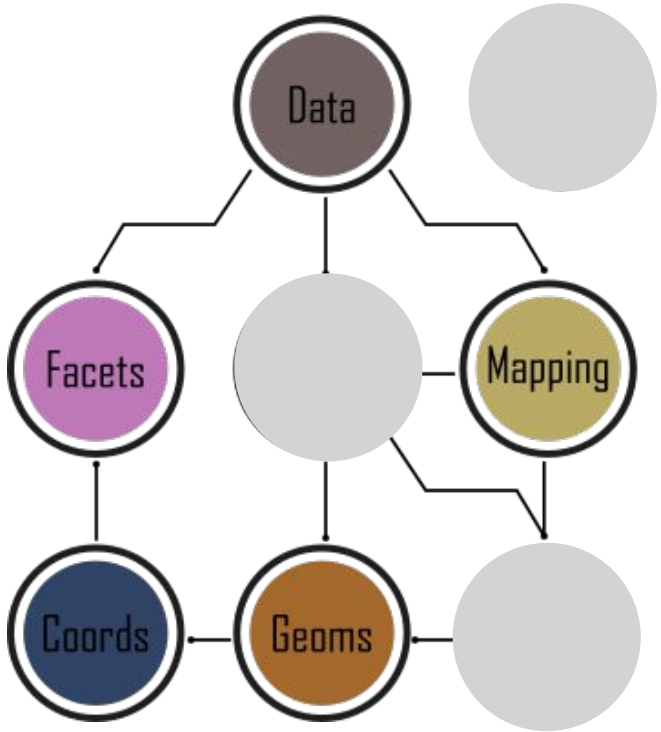


Why ggplot2 and not ggplot

ggplot2 is called *ggplot**2*** because once upon a time there was just a library *ggplot*. However, the developer noticed that it used an inefficient set of functions. In order for not to break the API, the authors introduced a successor package *ggplot2*. However, the central function in this package is still called `ggplot()`, not `ggplot2()`!

The framework of ggplot2

Input:
-the information we want to plot



Either use standardized theme or create any theme you want

Splitting plot in different groups

Aesthetics: what do you want displayed on the plot and where (i.e. x , y)

Apply statistics

Adapting x and y axis

How you want to plot.
Plot type: scatter, bar, boxplot

Starting with the end

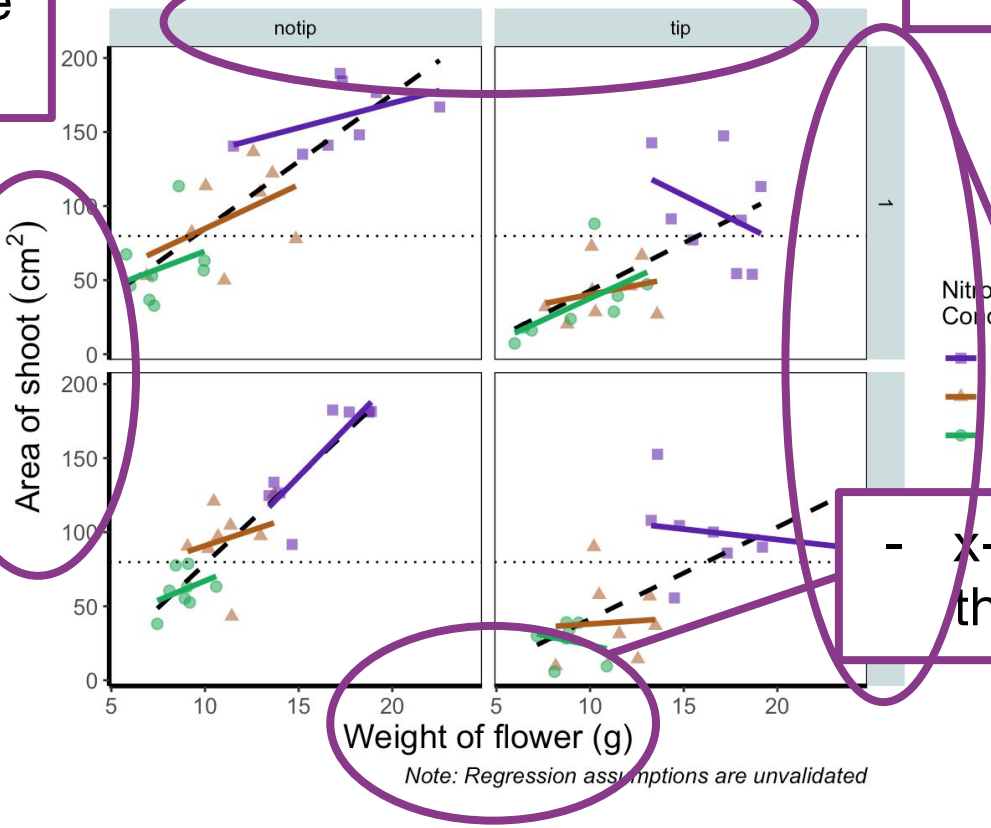
What do you see?

- y-axis represents the area of shoots

- The columns if the tip of the flower is removed (notip) or retained (tip)

- If the plants were grown in block **1** or block **2** of the greenhouse are divided in the rows

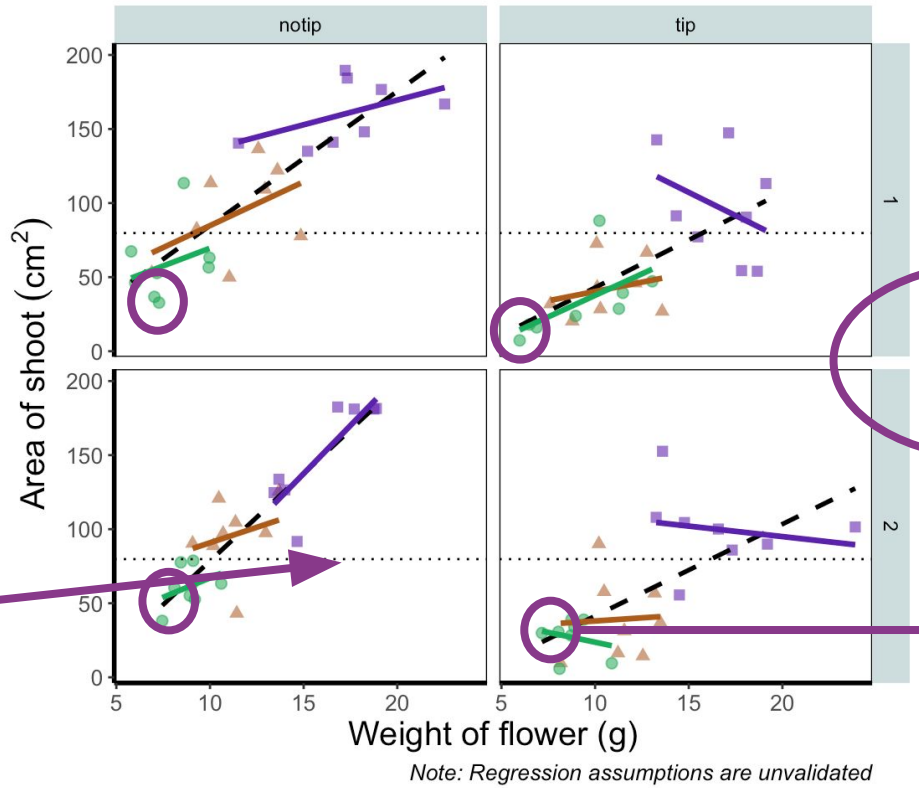
- x-axis represents the weight of flower



Starting with the end

and there is even more…..

- different colors
- different shapes



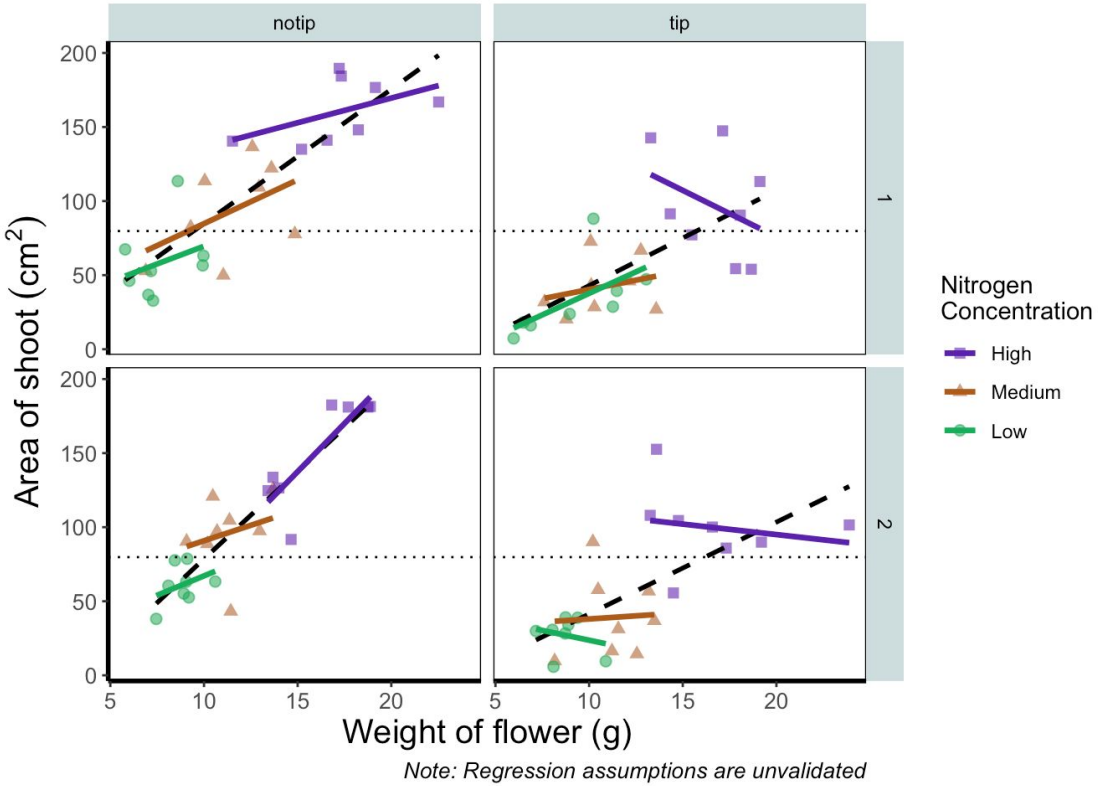
The color and shape are assigned to the different nitrogen concentration

A grey dotted line is drawn for the overall mean of the shoot area

Shapes and color are shared over all the plots

Starting with the end

And even more…..
Trendlines using a linear model



Now: let's start

1. Installing the ggplot2 package

```
install.packages("ggplot2")
library(ggplot2)
```

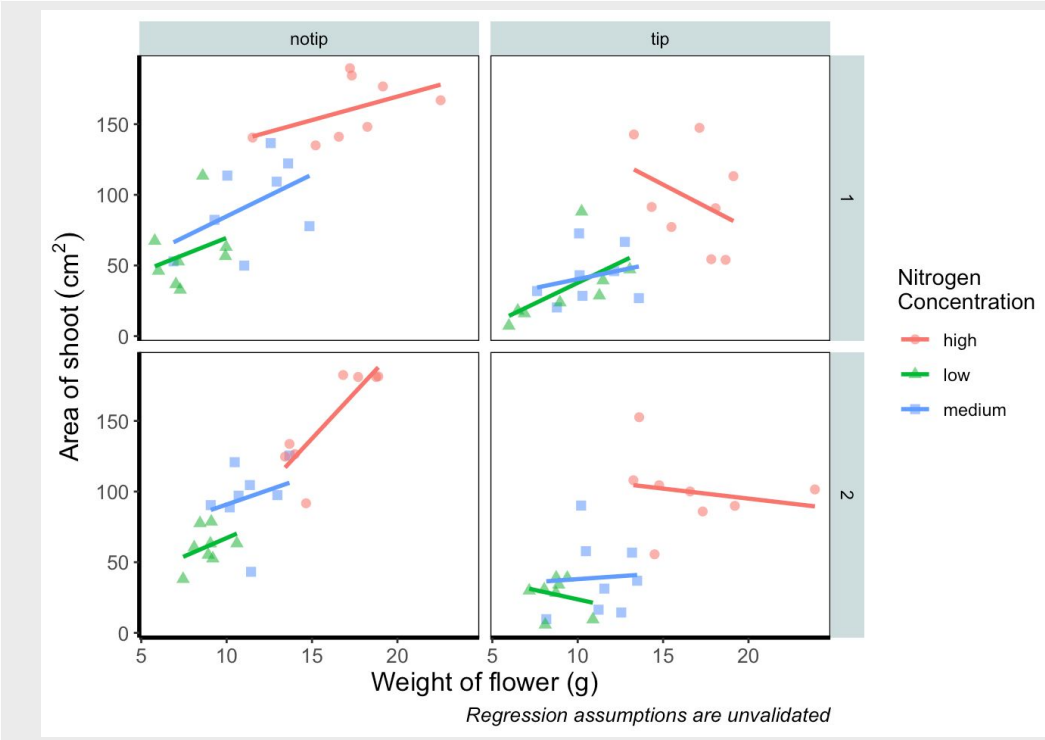
2. Let's make a ggplot*:

```
ggplot()
```

*When asked a group of students what the **`ggplot()`** function does, the answer was :
Obviously, it makes a **`ggplot`**

WOW

The amazing perfect `ggplot`



….. but nothing like this

So, let's come back to the flower data

from the 'final figure' we know that we want to place :

- the `weight` on the x-axis
- the `shootarea` on the y-axis

```
flower <- read.table("data/flower.csv", stringsAsFactors = TRUE,
                                        header = TRUE, sep = ",")
str(flower)
## 'data.frame':    96 obs. of  8 variables:
##  $ treat    : Factor w/ 2 levels "notip","tip": 2 2 2 2 2 2 2 2 2 2 ...
##  $ nitrogen : Factor w/ 3 levels "high","low","medium": 3 3 3 3 3 3 3 3 3 3 ...
##  $ block    : int  1 1 1 1 1 1 1 1 2 2 ...
##  $ height   : num  7.5 10.7 11.2 10.4 10.4 9.8 6.9 9.4 10.4 12.3 ...
##  $ weight   : num  7.62 12.14 12.76 8.78 13.58 ...
##  $ leafarea : num  11.7 14.1 7.1 11.9 14.5 12.2 13.2 14 10.5 16.1 ...
##  $ shootarea: num  31.9 46 66.7 20.3 26.9 72.7 43.1 28.5 57.8 36.9 ...
##  $ flowers  : int  1 10 10 1 4 9 7 6 5 8 ...
```

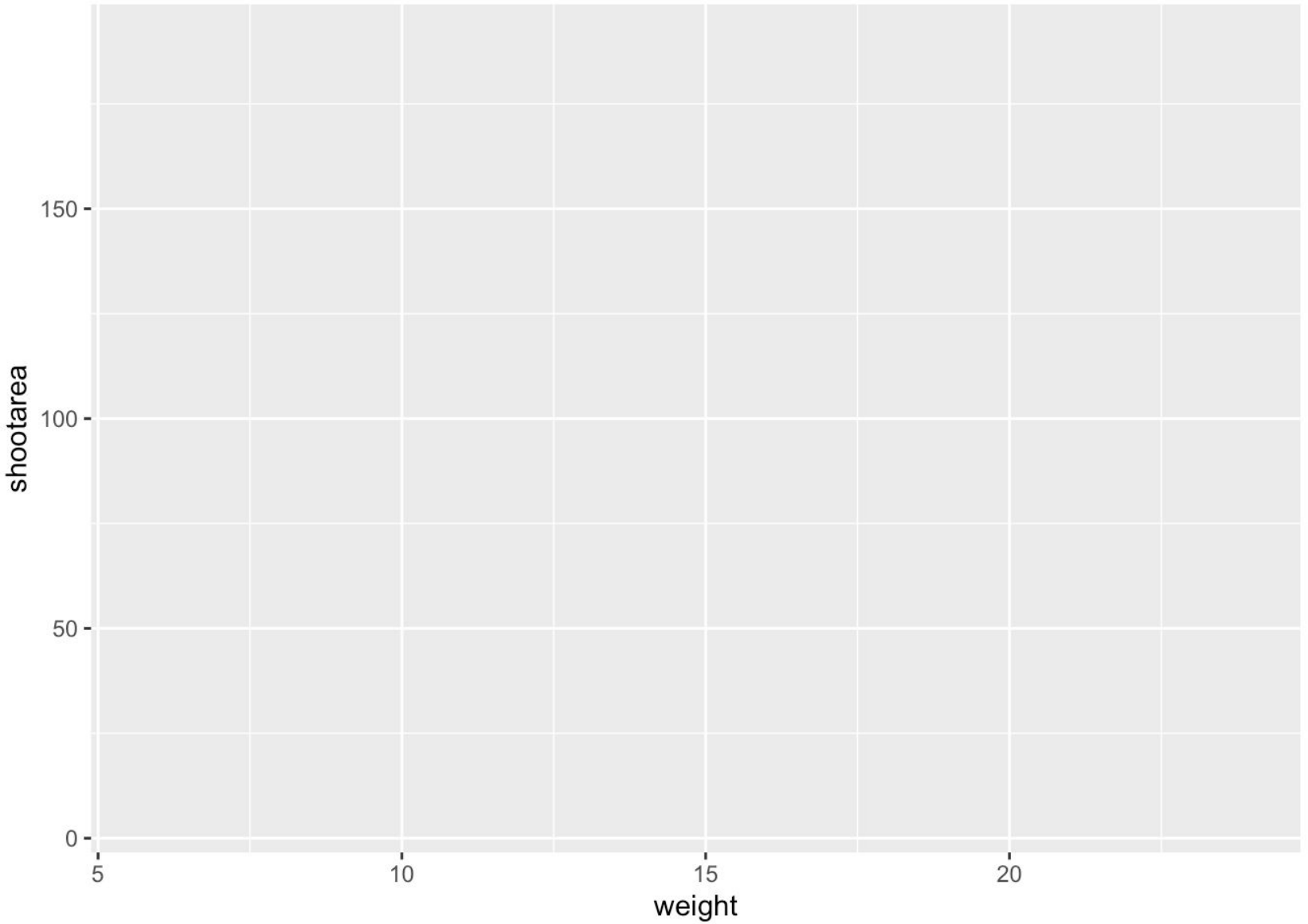
ggplot works a bit different.
To plot we need to make use of the `aes()`* f
and also add the `data =` argument

*`aes()` is short for aesthetics, in here we will define what we want on the plot

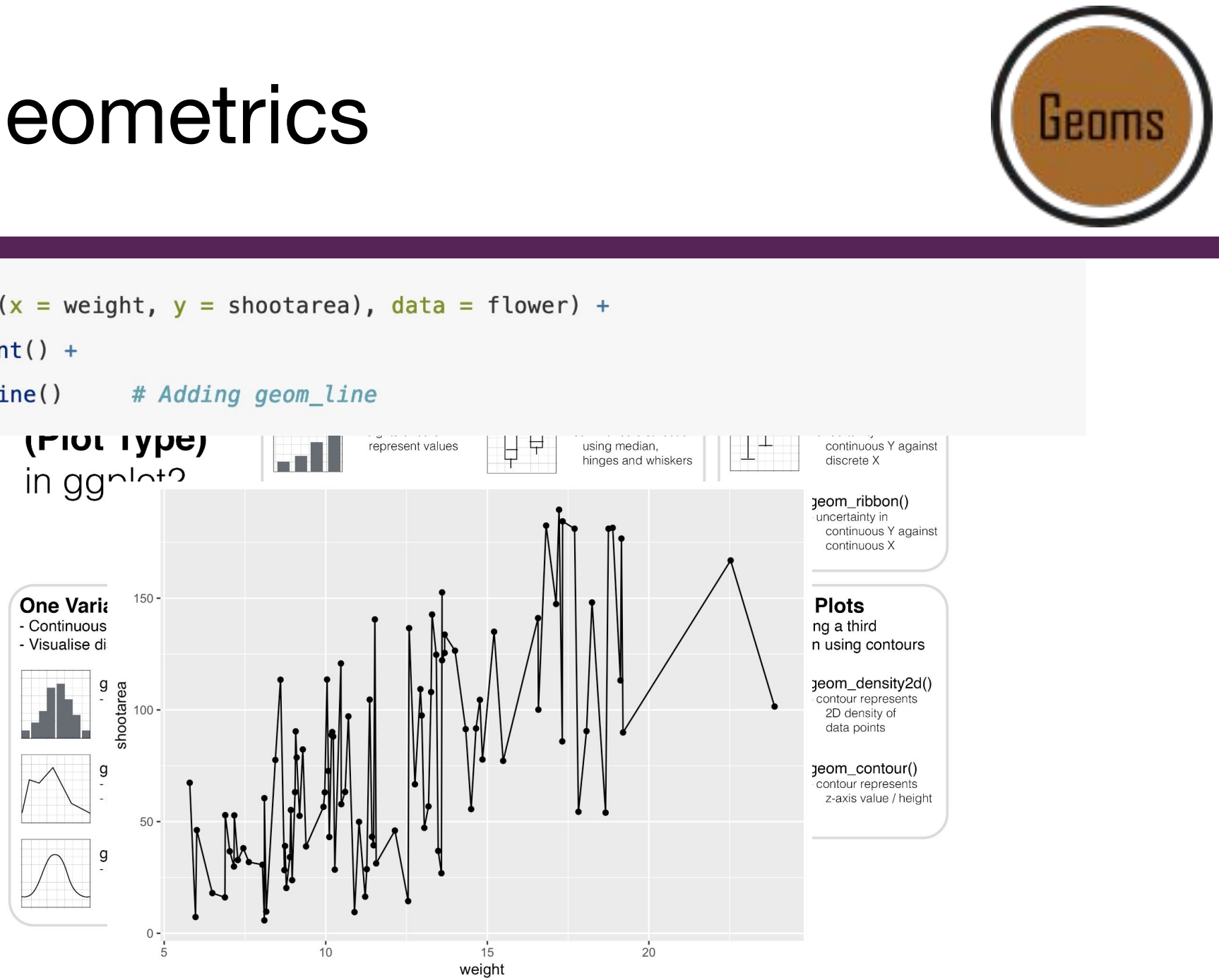
Coding example

from the ‘final
that we want t

- the `weigh`
- the `shoot`

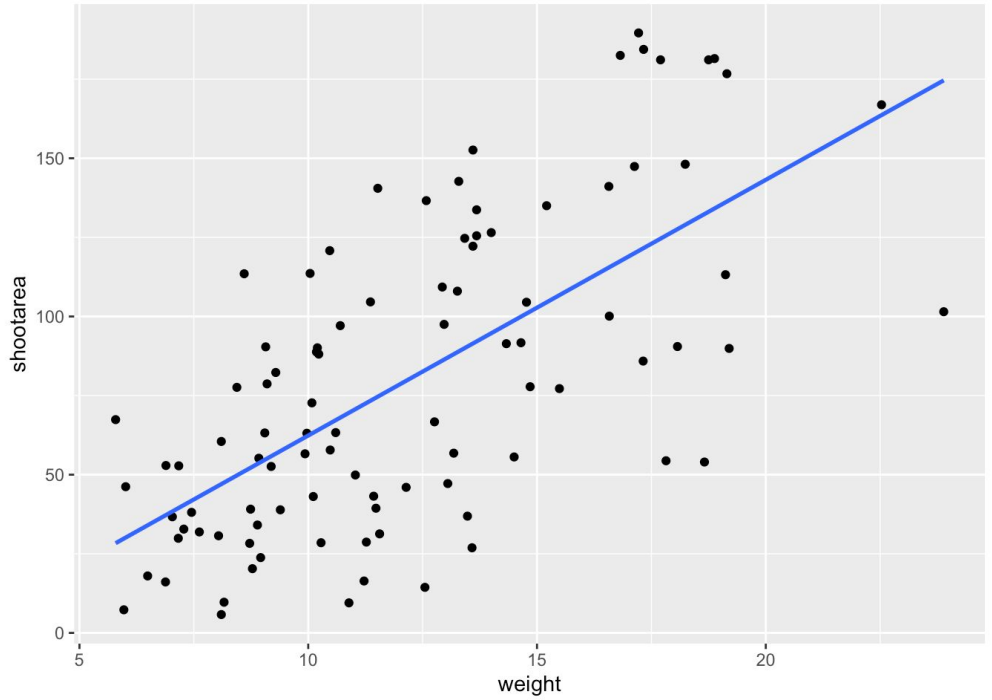


Next; geometrics



Next; geometrics + stats

```
ggplot(aes(x = weight, y = shootarea), data = flower) +
  geom_point() +
    geom_smooth(method = "lm", se = FALSE)    # method and se
```

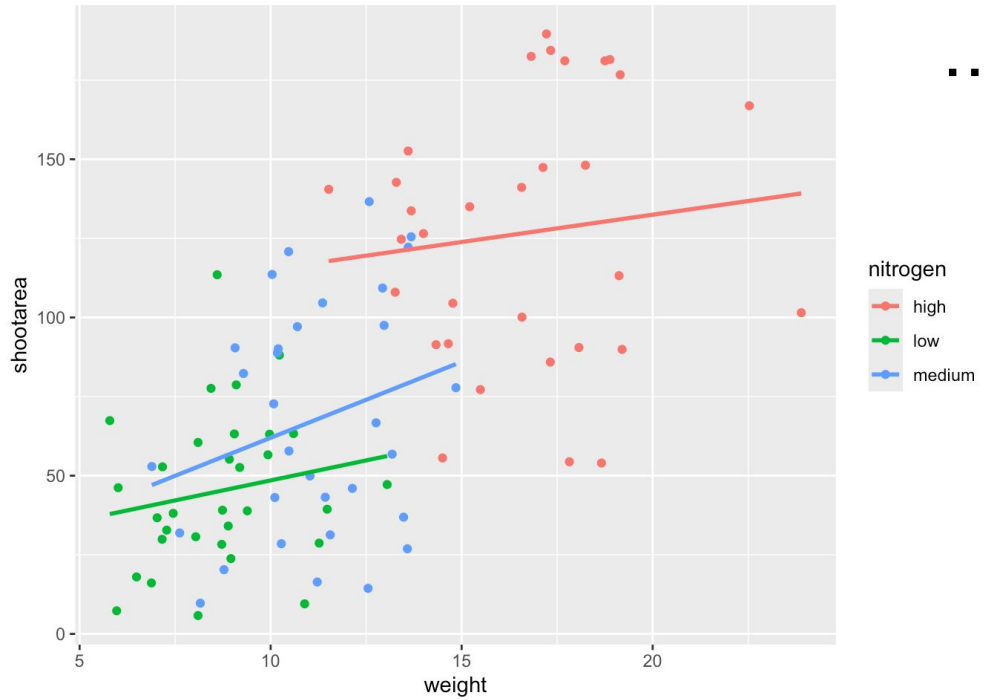


Now, let's make it prettier


```
# Moved colour = nitrogen into the universal ggplot()
ggplot(aes(x = weight, y = shootarea, colour = nitrogen), data = flower) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

So simple…..

…and prettier



Almost there……

split according to the treat and/or block variables

```
facet_wrap(~ treat)
```

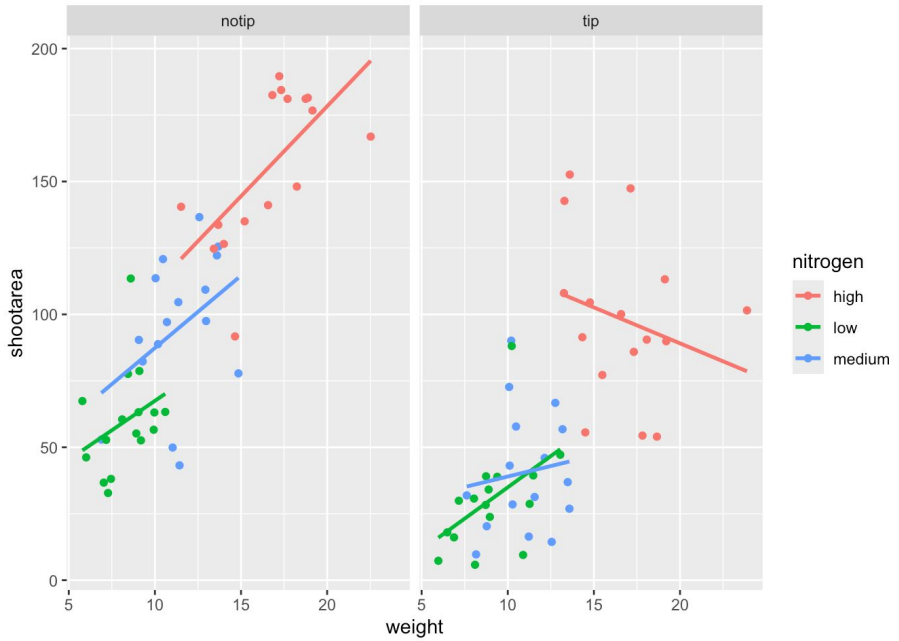
```
facet_grid(~ treat + block)
```

facet_wrap() : “wraps” a 1d ribbon of panels into 2d.
facet_grid() : produces a 2d grid of panels defined by variables which form the rows and columns.

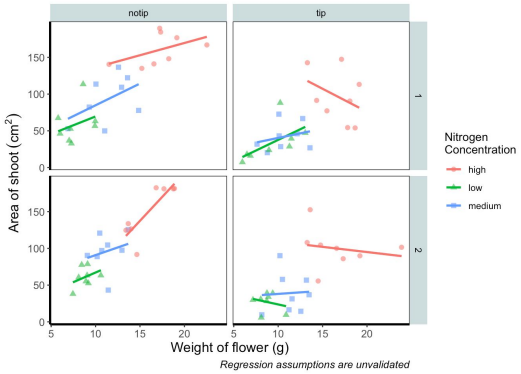
Almost there……

split according to the treat and block variables

```
ggplot(aes(x = weight, y = shootarea, colour = nitrogen), data = flower) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
    # Splitting the single figure into multiple depending on treatment
  facet_wrap(~ treat)
```

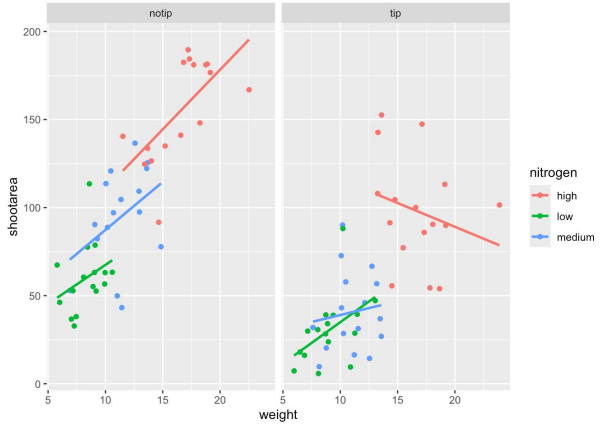


Find the figure closest to:

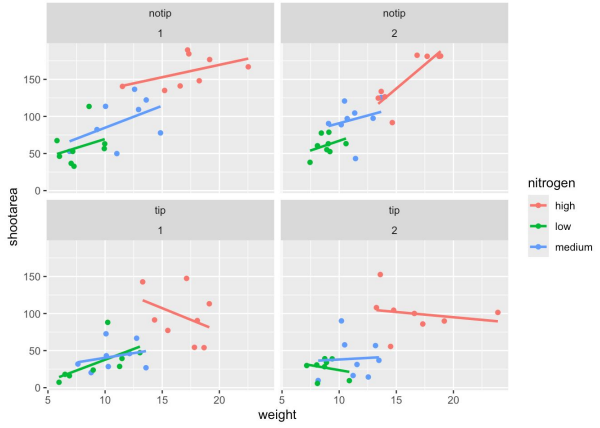



```
ggplot(aes(x = weight, y = shootarea, colour = nitrogen), data = flower) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
    # Changing to facet_grid
```

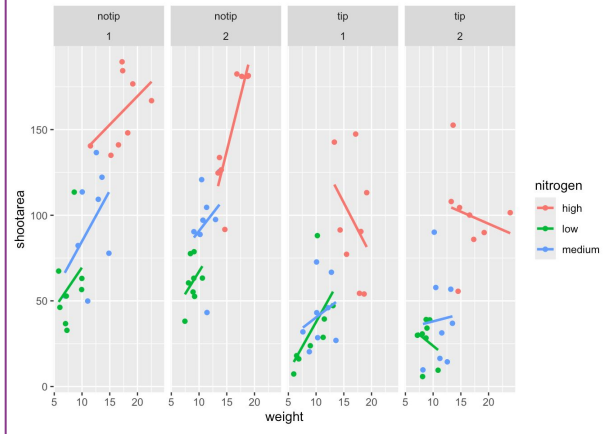
`facet_wrap(~ treat)`



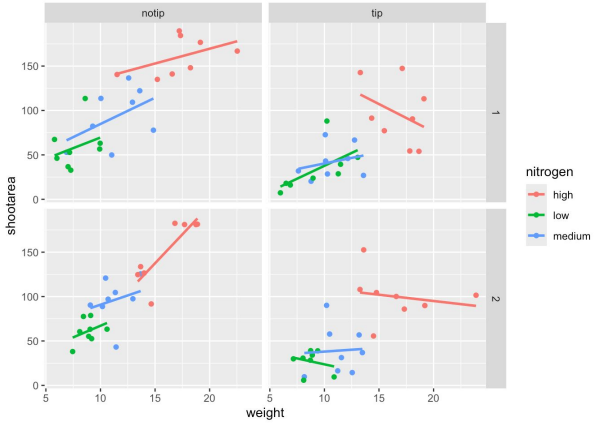
`facet_wrap(~ treat + block)`



`facet_grid(~ treat + block)`



`facet_grid(block ~ treat)`



Finishing touches

You can use standard themes such as `theme_classic()`, `theme_bw()`, `theme_minimal()`, `theme_light()`
Or create your own.
The theme for the final plot is as follow:

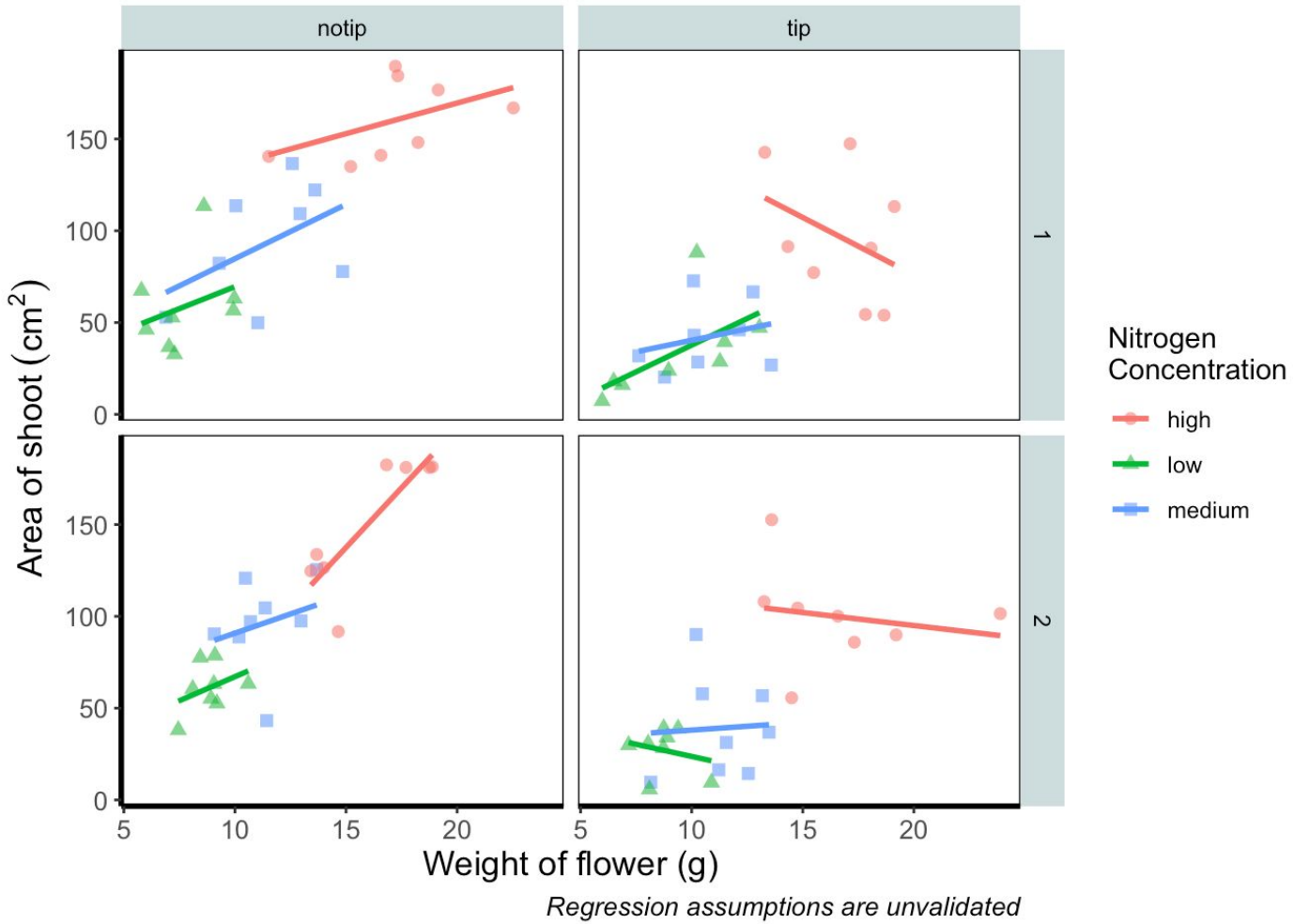
```
theme_rbook <- function(base_size = 13, base_family = "", base_line_size = base_size/22,
                        base_rect_size = base_size/22) {
  theme(
    axis.title = element_text(size = 13),
    axis.text.x = element_text(size = 10),
    axis.text.y = element_text(size = 10),
    plot.caption = element_text(size = 10, face = "italic"),
    panel.background = element_rect(fill="white"),
    axis.line = element_line(size = 1, colour = "black"),
    strip.background =element_rect(fill = "#cddcdd"),
    panel.border = element_rect(colour = "black", fill=NA, size=0.5),
        strip.text = element_text(colour = "black"),
        legend.key=element_blank()
  )
}
```

```
ggplot(aes(x = weight, y = shootarea, colour = nitrogen), data = flower) +
    geom_point(aes(shape = nitrogen), size = 2, alpha = 0.6) +
    geom_smooth(method = "lm", se = FALSE) +
    facet_grid(block ~ treat) +
    xlab("Weight of flower (g)") +
*   ylab(bquote("Area of shoot"~(cm^2))) +
*   labs(shape = "Nitrogen\nConcentration", colour = "Nitrogen\nConcentration",
             caption = "Regression assumptions are unvalidated") +
    # Updated theme to our theme_rbook
    theme_rbook()   # use our new theme
```

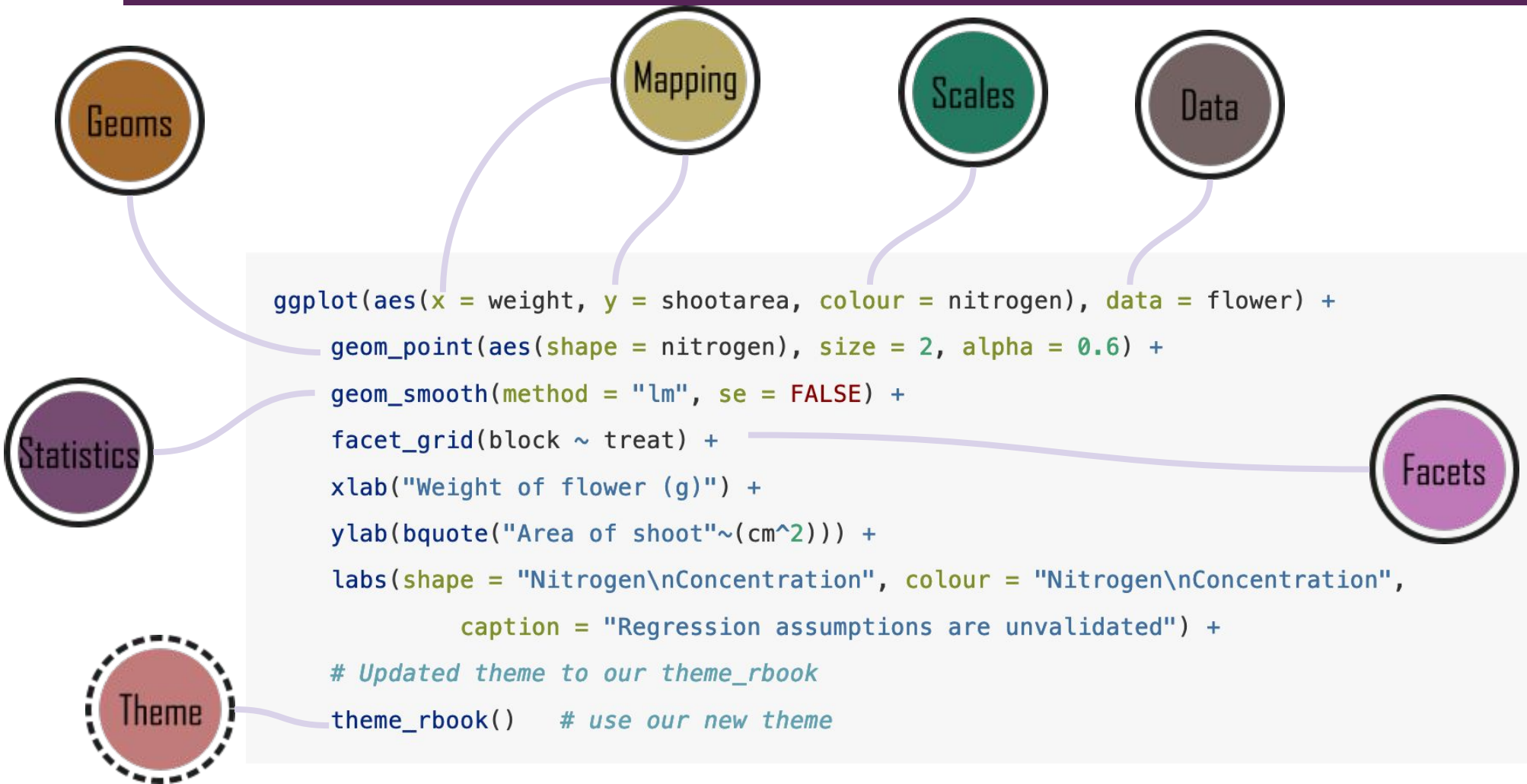
Manually adjusting the x,y axis and legend and adding the beautiful theme (:

* Using bquote to get mathematically correct formatting
* Including \n to split legend title over two lines

Voilà


notip
tip
1
2
Area of shoot (cm^2)
0
50
100
150
5
10
15
20
Weight of flower (g)
Nitrogen Concentration
high
low
medium
Regression assumptions are unvalidated

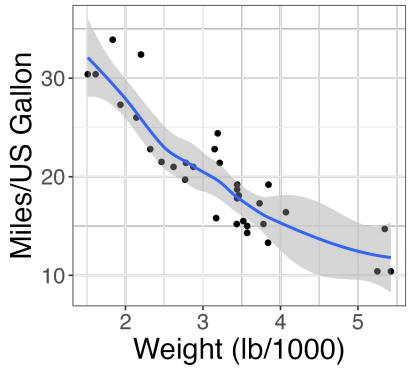
Summary



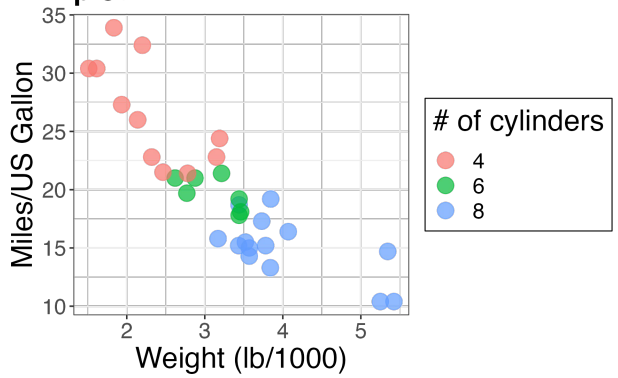

```
ggplot(aes(x = weight, y = shootarea, colour = nitrogen), data = flower) +
    geom_point(aes(shape = nitrogen), size = 2, alpha = 0.6) +
    geom_smooth(method = "lm", se = FALSE) +
    facet_grid(block ~ treat) +
    xlab("Weight of flower (g)") +
    ylab(bquote("Area of shoot"~(cm^2))) +
    labs(shape = "Nitrogen\nConcentration", colour = "Nitrogen\nConcentration",
             caption = "Regression assumptions are unvalidated") +
    # Updated theme to our theme_rbook
    theme_rbook()   # use our new theme
```

Exercise time: reproduce the following plots

plot 1

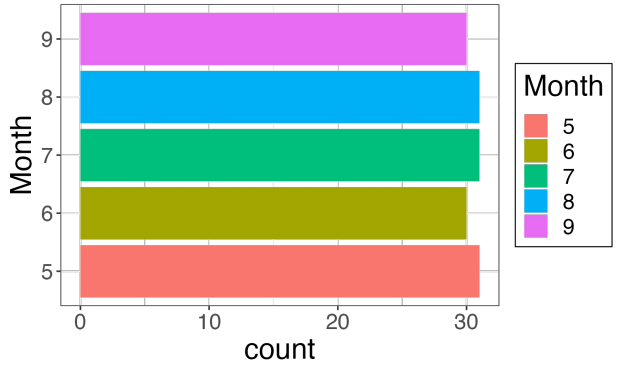


plot 2

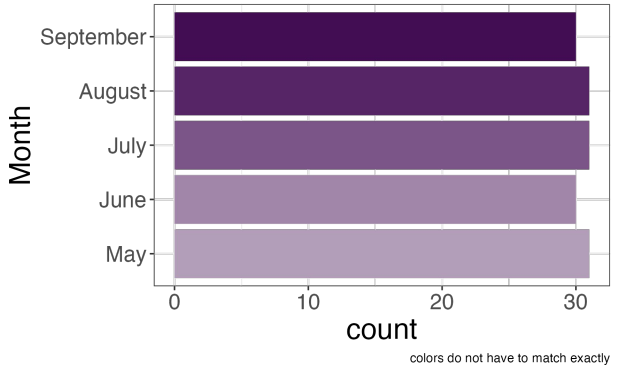


Use **`data(mtcars)`**

plot 3

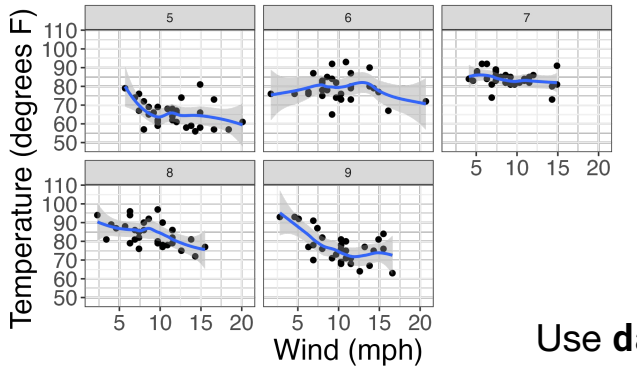


plot 3.1 (optional)



colors do not have to match exactly

plot 4

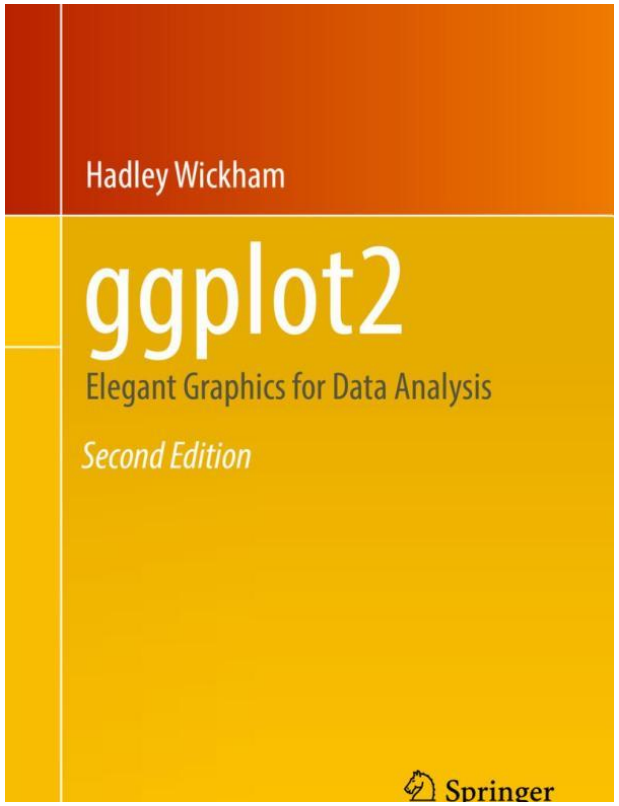


Use **`data(airquality)`**

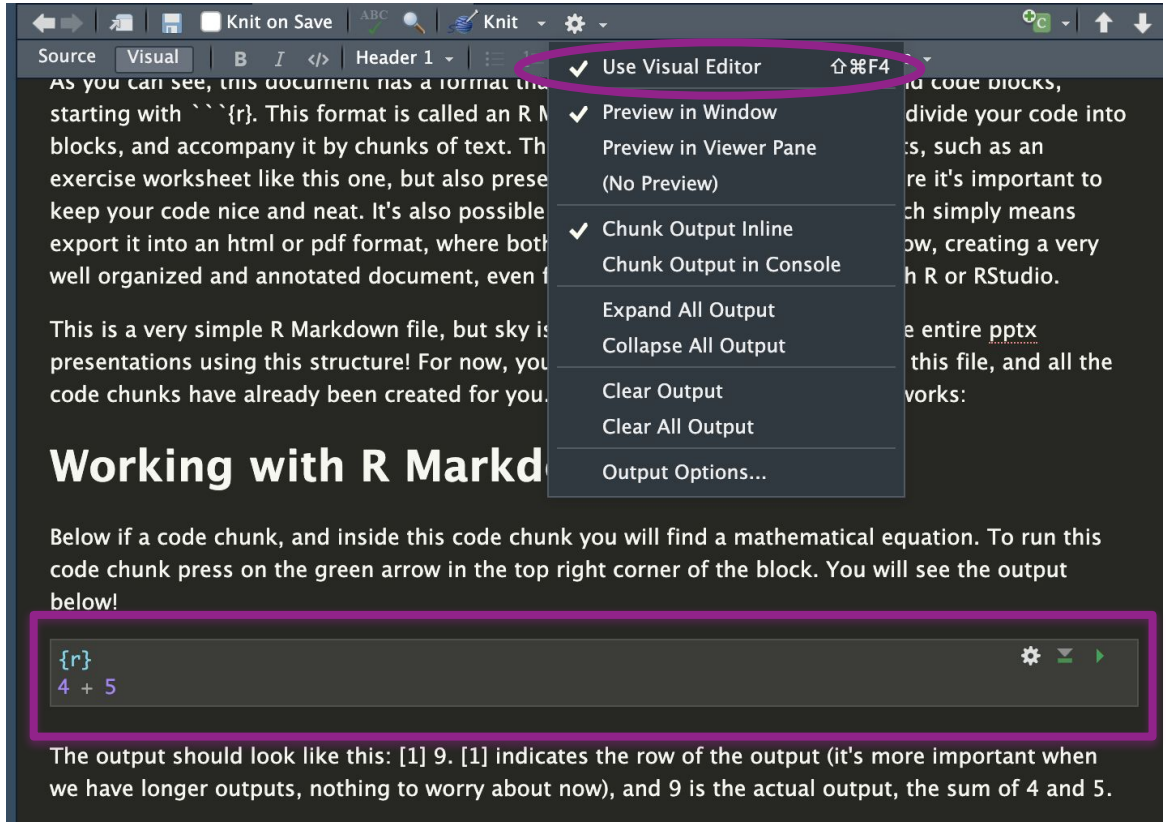
If you want to know more…

- We recommend this book on **ggplot** if you want to make kick-ass visualizations

- You can find it here: https://ggplot2-book.org/



Your turn!



Download the file Intro_to_R.Rmd from the Github Repository

You should write your code into these boxes, which are called chunks!