7316 - INTRODUCTION TO DATA ANALYSIS WITH R

MODULE 3: DESCRIBING DATA.

Mickaël Buffart (mickael.buffart@hhs.se)

Table of contents

1. MODULE 3: Describing data

Now, you know how to import data, create and clean a dataset. In this module, we will learn how to describe a dataset with visuals and with summary statistics.

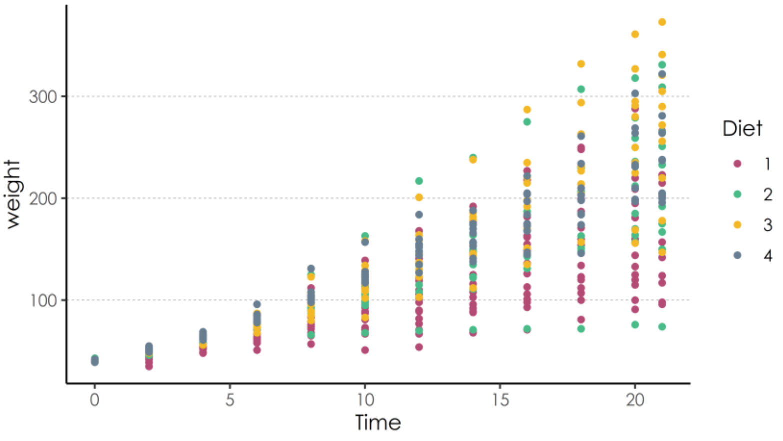
1.1 Graphs in R

1.1.1 Data visualization overview

One of the strong points of R is creating very high-quality data visualization. R is very good at both "static" data visualization and interactive data visualization designed for web use. Today, we will cover static data visualization.

1.1.2 ggplot2 for data visualization

The main package for publication-quality static data visualization in R is `ggplot2`, which is part of the tidyverse collection of packages. With `ggplot2` you control all the elements of your graphs, from the data you visualize and the way you represent it, to the style of the graph and the quality of the output (high-resolution...).



Example of ggplot2 graph, with SSE stylesheet

The workhorse function of ggplot2 is `ggplot2::ggplot()`. The `gg` stands for "*grammar of graphics*". In each `ggplot()` call, the appearance of the graph is determined by specifying:

- the `data`(frame) to be used,
- the `aes`(thetics) of the graph: like size, color, *x* and *y* variables,
- the `geom`(etry) of the graph: the chosen representation (*e.g.* `point`, `histogram`, `bar`, `qq`, `curve`, `density`, `abline`, `boxplot`, `map`, etc., etc.)
- You may then add other functions to your graph, to define, `labs` (the labels), `ggtitle` (the title), `ggtheme` (the theme), etc.

1.1.3 Scatterplots

First, let's look at a simple scatterplot, which is defined by using the geometry `geom_point()`.

Note: `ggplot2` requires many functions to run. I recommend you in this case to load the full library once for all before using it, or you will spend your time writing `ggplot::` everywhere.

```
# Loading the grammar of graphs
library(ggplot2)
```

Then, you can plot data. Let's use, for example, `graph_reg.Rds` (available on Canvas, in the module 3 data file).

```
df <- rio::import("data/graph_reg.Rds")

# Defining data
ggplot(df) +

  # defining aesthetics: x and y
  aes(x = iv, y = dv) +

  # defining geometry: geom_point is for scatterplot
  geom_point()
```

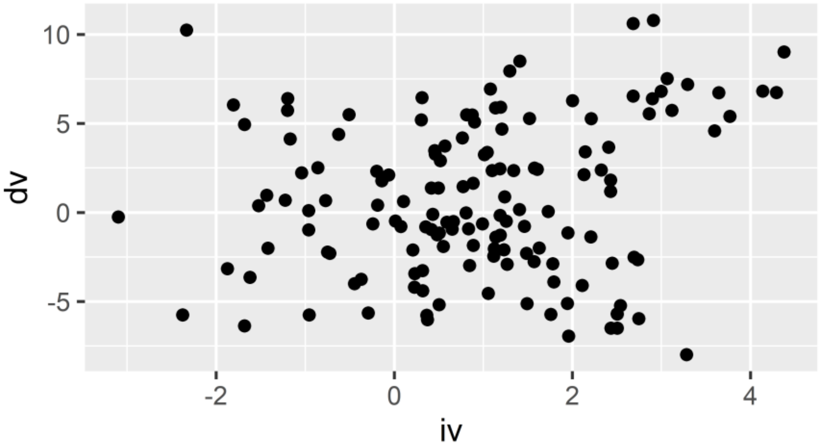


Figure 1: A simple scatterplot

- Each of the graph elements is an R function.
- Functions in `ggplot2` are added up with a `+` symbol.
- Variables names are NOT quoted (`""`) in `ggplot2`: they are written directly with their full names (here: `iv`, `dv`)

1.1.3.1 Colors and shapes

- Graphs can be extensively customized using additional arguments inside of elements, or with additional functions. For example, you may want to color the element by group:

```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group) +
  geom_point()
```

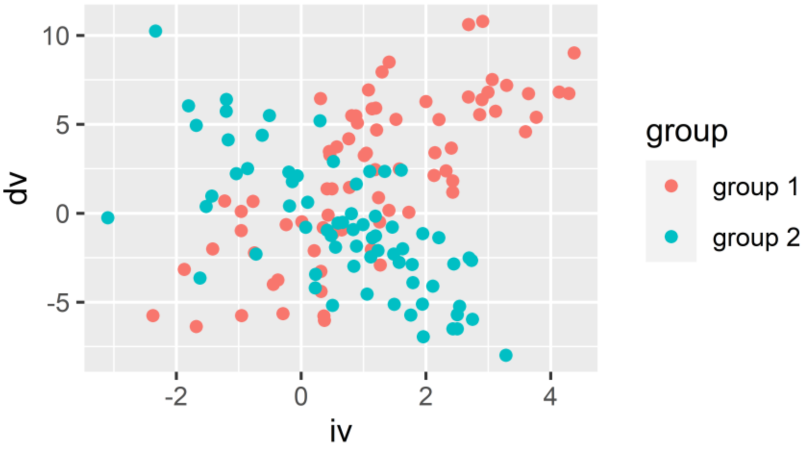


Figure 2: Adding colors

- Or you want to combine it with different shapes, in case you print in black and white:

```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group,
                 shape = group) +
  geom_point()
```

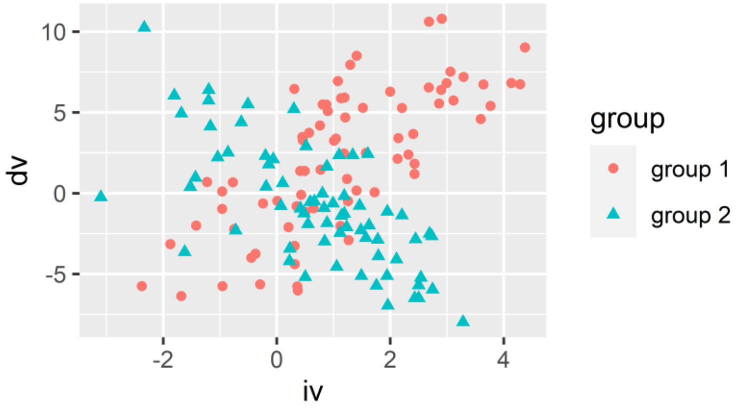


Figure 3: Adding shapes

Now, if you are unhappy with the choice of color or shapes, you can customize them:

```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group,
                 shape = group) +
  geom_point() +
  scale_color_manual(values = c("green", "blue")) +
  scale_shape_manual(values = c(3, 4))
```

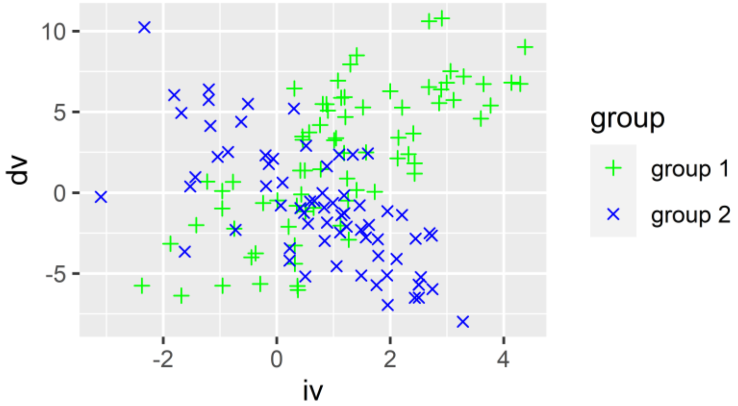


Figure 4: Manual colors and shapes

- The color can be written as color names in English, `blue`, `green`, `red`, `black`, etc. or as hexadecimal values, *e.g.* `#FF5733`. You can find a color picker here: https://htmlcolorcodes.com/
- The number of shapes are limited to 25 (numbered from 1 to 25). You can test them as you like, or you can find the list here: http://www.sthda.com/english/wiki/ggplot2-point-shapes
- In both cases, you have to make sure that your color or shape list contains as many values as the number of groups you want to plot (or, it can contain more, but not less).

1.1.3.2 Adding gradient colors

If you have a continuous scale to color, you may want to add gradient.

```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = iv) + geom_point() +
  scale_color_gradient(low = "green", high = "red",
                       name = "colored IV")
```

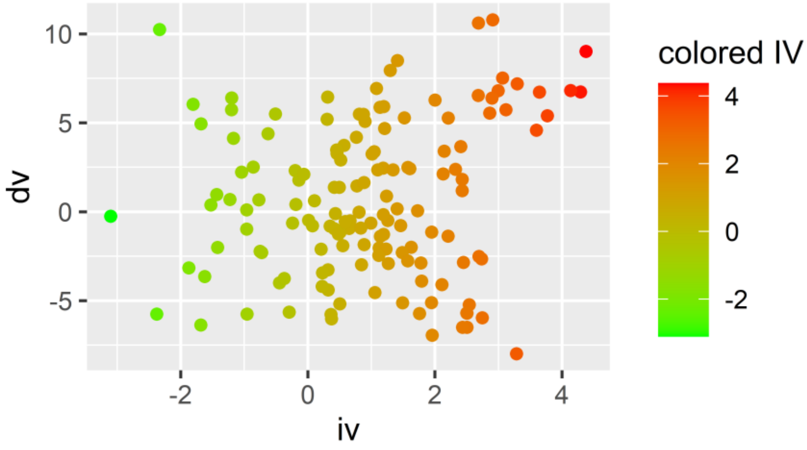
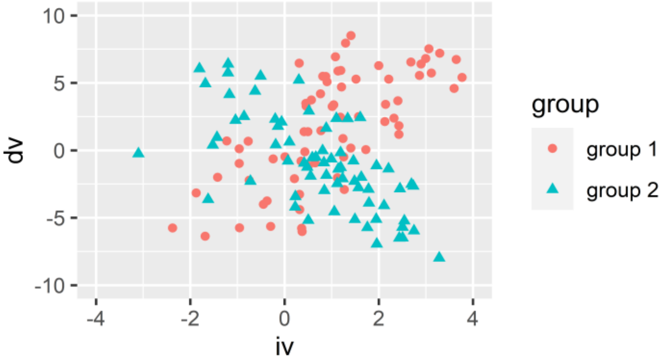


Figure 5: Gradients of colors

1.1.4 Axis and titles

Sometimes, you would like to use a customize scale for the axis:

```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group,
                 shape = group) +
  geom_point() +
  xlim(-4, 4) + ylim(-10,10)
Warning: Removed 6 rows containing missing values (geom_point).
```

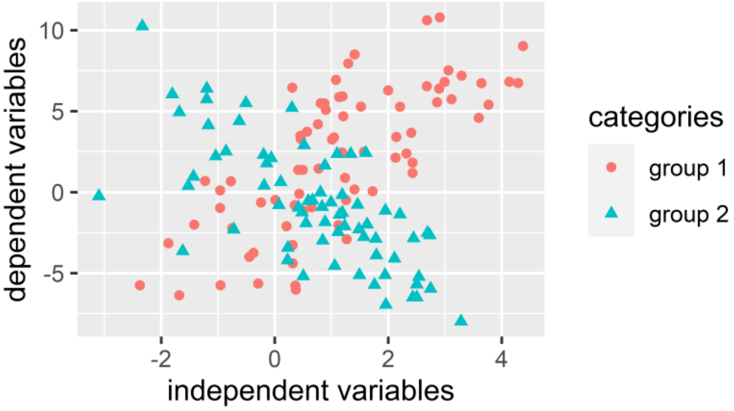


Custom scales

- Note the warning message you get: this is because some data points are out of bounds, hence, not plotted on the graph.

You can also change the name of axis, for example, using full labels:

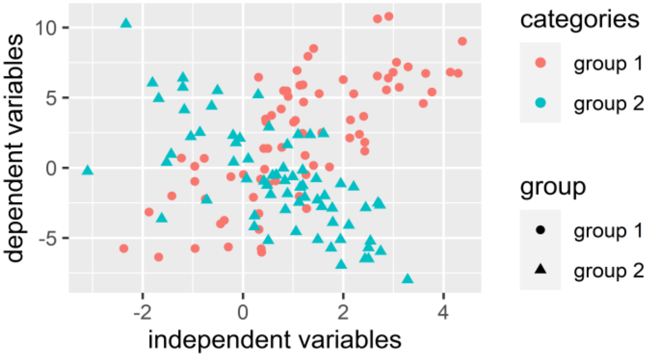
```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group,
                 shape = group) +
  geom_point() +
  labs(x = "independent variables",
       y = "dependent variables",
       color = "categories",
       shape = "categories")
```



Custom labels

- Note that you have to indicate both `color` and `shape` in your graph to change the title of your legend, because you graph include colors and shapes. If you change only one, the legen will be repeated twice: once with the new label you wrote, one with the variable name. Example:

```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group,
                 shape = group) +
  geom_point() +
  labs(x = "independent variables",
       y = "dependent variables",
       color = "categories")
```

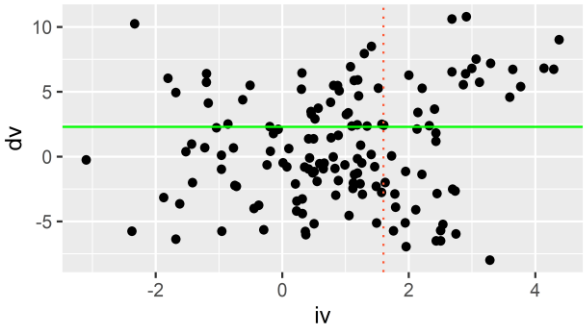


Wrongly assigned shape legend

1.1.4.1 Lines

To plot a straight line somewhere on your graph, you can use `geom_vline()` (vertical) or `hline` (horizontal):

```
ggplot(df) + aes(x = iv,
                 y = dv) +
  geom_point() +
  geom_vline(xintercept = 1.6,
             color = "#FF5733",
             linetype = "dotted") +
  geom_hline(yintercept = 2.3,
             color = "green",
             linetype = "solid")
```



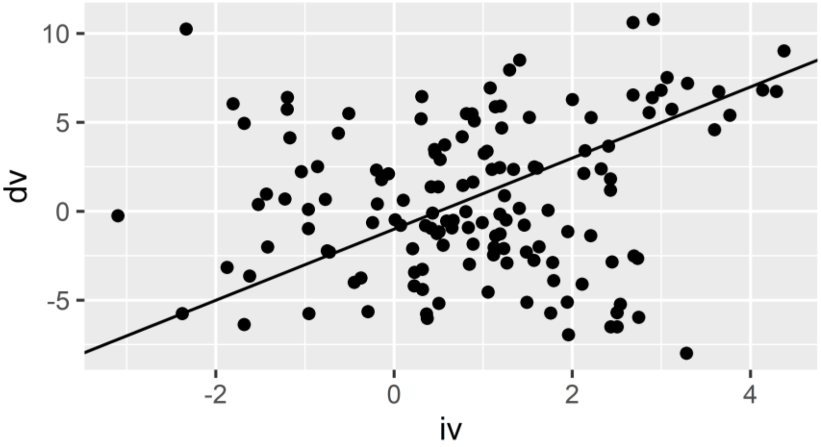
Vertical and horizontal lines

- Here again, you can customize the position, color, or linetype. The seven styles of linetypes are available here.

1.1.4.2 ablines and regression lines

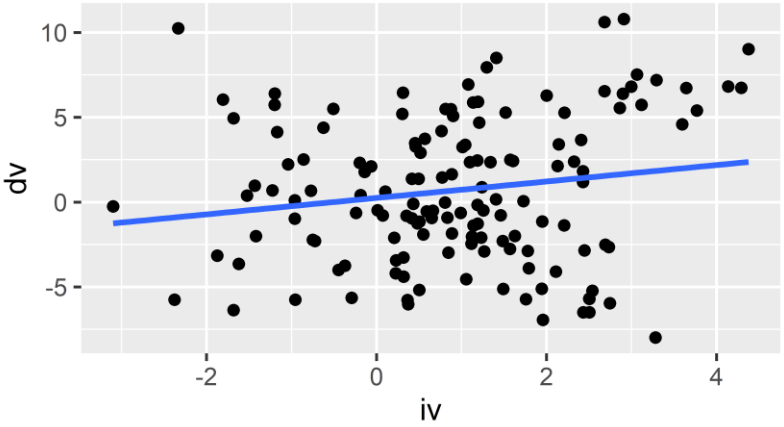
You can also plot a line across the graph, indicated slope and intercept, using `abline()`, or you can plot a regression lines fitted to your data, using `geom_smooth()`:

```
# With abline (not fitted line)
ggplot(df) +
  aes(x = iv, y = dv) +
  geom_point() +
  geom_abline(intercept = -1,
              slope = 2)
```



Abline and regression lines

```
# With geom_smooth (fitted)
ggplot(df) + aes(x = iv,
                 y = dv) +
  geom_point() +
  geom_smooth(method = "lm",
              formula = "y ~ x", se = FALSE)
```



Abline and regression lines

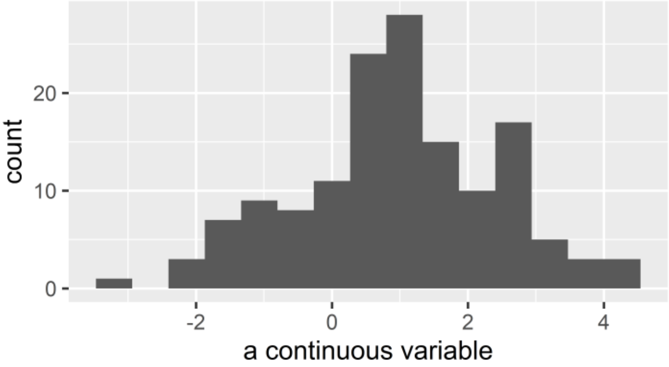
The parameter of `geom_smooth`:

- `method`: the model to use to fit the data (here, OLS model)
- `formula`: the formula used to fit the model (here, y on x, but more complex formula could be written, as we will see)
- `se`: a logical whether standard errors should be plotted or not.

1.1.5 Bars and histograms

For continuous variables, you may want to plot an histogram with `geom_histogram()`.

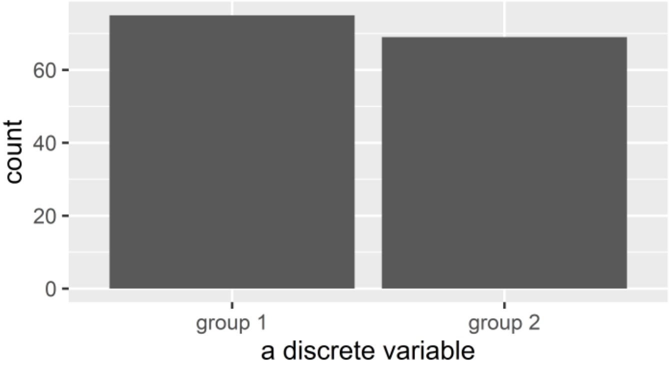
```
ggplot(df) +
  aes(x = iv) +
  geom_histogram(bins = 15) +
  labs(x = "a continuous variable")
```



Drawing histograms

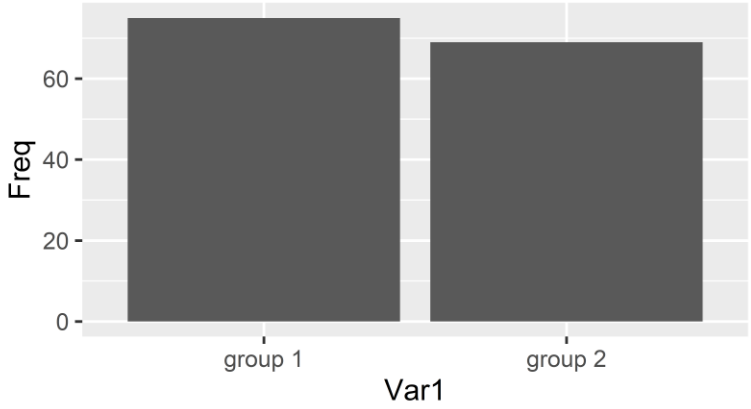
`bar_plot()` is the same, but for discrete variables. By default, a bar plot uses frequencies for its values, but you can use values from a column by specifying `stat = "identity"` inside `geom_bar()`.

```
# Bar plot with frequency
ggplot(df) +
  aes(x = group) +
  geom_bar() +
  labs(x = "a discrete variable")
```



Drawing barplots

```
tmp <- as.data.frame(table(df$group))
ggplot(tmp) +
  aes(x = Var1, y = Freq) +
  geom_bar(stat = "identity")
```



Drawing barplots

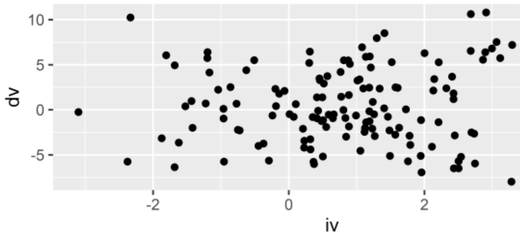
1.1.6 Saving and reusing graph

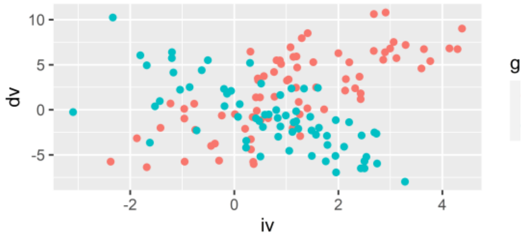
Sometimes, you may want to save a graph for later use, or for update. With R, it is possible: a graph is an object that can be manipulated as any other object. For example, in the first example above, instead of repeating twice, the same code, we could have assigned the scatterplot to an object and added colors to that object. Example:

```
# Assigning ggplot object in the environment
graph_1 <- ggplot(df) +
  aes(x = iv, y = dv) +
  geom_point()

# Adding colors
graph_2 <- graph_1 + aes(color = group)

# Print graph_1 and graph_2
graph_1
graph_2
```





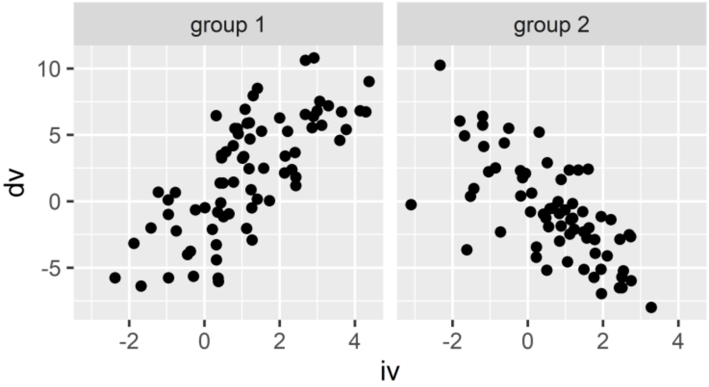
without colors

colors added

1.1.7 Facet grid

I placed the two figures above side by side, but they are two different figures. Sometimes, you may want to show a grid of figures, for example, n scatterplots of the same variables, per group (instead of using colors or shapes). You can do this with `facet_wrap`. For example, reusing `graph_1` from above:

```
graph_1 + facet_wrap(~group, ncol = 2)
```



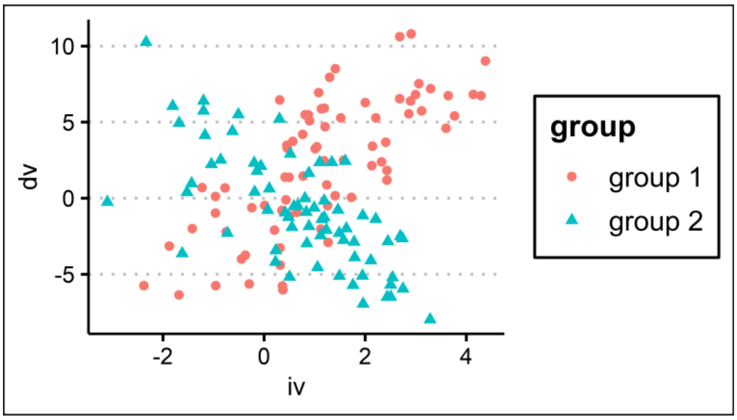
Facet wrap

1.1.8 Adding themes

Another option to affect the appearance of the graph is to use themes, which affect a number of general aspects concerning how graphs are displayed.

- Some default themes come installed with ggplot2/tidyverse, but some of the best in my opinion come from the package ggthemes. You can see the gallery of themes here.
- To apply a theme, just add `+ themename()` to your ggplot graphic, or `+ ggthemes::themename()`.

```
ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group,
                 shape = group) +
  geom_point() + ggthemes::theme_clean()
```



Changing theme

1.1.9 High quality graph

Once your plot is ready, if you want to save it in high resolution, you can use:

```
png(filename = "figure_1.png",
    unit  = "cm", width = 12, height = 12,
    res = 800)

ggplot(df) + aes(x = iv,
                 y = dv,
                 color = group,
                 shape = group) +
  geom_point() + ggthemes::theme_clean()

dev.off()
```

- where `width` and `height` are the size of your plot, saved in `figure_1.png`, and `res` is the resolution (*i.e.* the quality) of the image. By default, the resolution of plots is 150 dpi. A higher value is preferable.

1.1.10 More with ggplot2

This has been just a small overview of things you can do with ggplot2. To learn more about it, you can:

- read the book in the syllabus about `ggplot2`: https://ggplot2-book.org/.
- You may also want to read the STHDA Guide to ggplot2, a good general guide to ggplot2 that is still pretty thorough.
- Finally, the RStudio cheat sheet may help you move further.

1.2 Descriptive statistics

1.2.1 Summarize data

When exploring your data, or writing your result section in your thesis, you will need descriptive statistics. R includes all the required functions for usual descriptive statistics: `mean()`, `median()`, `sd()`, `min()`, `max()`, `cor()`, `quantile()`, `IQR()` , *etc.*

To get the mean of wages (example from module 1), a continuous , simply enter:

```
# Loading data
wages <- Ecdat::Wages1

# Calculating the mean
mean(wages$wage)
```

```
[1] 5.757585
```

- You can also save the multiple descriptive statistics in a dataframe. Example:

```
summary_stat <- data.frame(avg_wage    = mean(wages$wage),
                           median_wage = median(wages$wage),
                           min_wage     = min(wages$wage),
                           max_wage     = max(wages$wage))
```

- or you may want to calculate the mean or other measures per group:

```
aggregate(wages$wage, list(wages$sex), FUN = mean)

  Group.1        x
1  female 5.146924
2    male 6.313021
```

Warning: If your data contains missing values, the function will send `NA`. To avoid it, use the `na.rm` argument:

```
test_var <- c(2, 5, 5, 4, 8, 9, NA, 12)

# No na.rm
mean(test_var)

[1] NA

# With na.rm
mean(test_var, na.rm = TRUE)

[1] 6.428571
```

- `length()` is a count function. It returns the length of a vector.
  - `length(unique())` will then give you the number of unique values in a vector.
  - You can also extract frequencies of all unique values into a `table()`:

```
table(wages$school)


   3    4    5    6    7    8    9   10   11   12   13   14   15   16 
   1    2    5   14   24   86  161  399  661 1188  339  264  134   16 
```

Question: How would you extract this table into Excel?

Solution:

```
tmp <- as.data.frame(table(wages$school), stringsAsFactors = FALSE)

rio::export(tmp, "data/CNT.xlsx")
```

1.2.2 Summary statistics in a nice table

Often, in your report, you simply want to insert a table with the mean and standard deviation on all the variables in your model. You can do this with the command `st()` from the package `vtable`:

```
vtable::st(wages, out = "kable")
```

Table 1: Using vtable without options

Variable	N	Mean	Std. Dev.	Min	Pctl. 25	Pctl. 75	Max
exper	3294	8.043	2.291	1	7	9	18
sex	3294						
... female	1569	47.6%					
... male	1725	52.4%					
school	3294	11.631	1.658	3	11	12	16
wage	3294	5.758	3.269	0.077	3.622	7.305	39.809

- This function is very handy. You can use it to generate a table compatible with word, that you can use directly in your thesis.
- `out = "kable"` is here to create a table that you can export in a MS Word document. You can also export tables in html format (for a webbrowser), csv, or in latex.
- You can also adjust other options of the table, as bellow:

```
# Example:
vtable::st(
  # The dataset to describe
  wages,

  # Vector of variables to display
  vars = c("school", "wage"),

  # How to name the variables in the output table
  labels = c("Years of schooling", "Wage per hours"),

  # Statistics to display
  summ = c("notNA(x)", "mean(x)", "sd(x)", "min(x)", "max(x)"),

  # How to name the columns in the output table
  summ.names = c("N", "Mean", "S.D.", "Min", "Max"),

  # Number of decimals to display
  digits = 3,

  # Table to export for MS Word
  out = "kable"
)
```

Table 2: Summary Statistics

Variable	N	Mean	S.D.	Min	Max
Years of schooling	3294	11.631	1.658	3	16
Wage per hours	3294	5.758	3.269	0.077	39.809

- By default, it returns the descriptive statistics of all your variables in your dataframe.
  - If you want to select return, for example, only `school` and `wage` variables in your table, use: `vars = c("school", "wage")`.
  - You can assign the labels of each columns with `labels =` followed by a vector of labels. Warning: labels MUST be ordered in the same order as the `vars` parameter.
  - summ and summ.names to choose what statistics to display and how to name the columns. For example, to display N, mean, standard deviation, min, and max, you would enter the parameter: `summ = c("notNA(x)", "mean(x)", "sd(x)", "min(x)", "max(x)")`. The help page provides more information on what statistics can be displayed.
  - `digits =` lets you choose how many decimals to display

1.2.3 Correlation matrices

You can return a correlation between two variables with `cor(x, y)`.

Unfortunately, correlation matrices, which you need in most quantitative works, have always been a pain to display easily in R. The following blog posts give you relatively easy solution to display a correlation matrix with an acceptable design for an academic work:

- Correlation matrices according to sthda.
- Correlation matrices on datadream.org.

The simplest option to get a correlation matrix is with the `corrr` package from the tidyverse.

```
# Create a correlation table
x <- corrr::correlate(wages)
```

```
Non-numeric variables removed from input: `sex`
Correlation computed with
• Method: 'pearson'
• Missing treated using: 'pairwise.complete.obs'
```

```
# Save only the lower triangle
x <- corrr::shave(x)

# Format "nicely", and get ready for MS Word
knitr::kable(corrr::fashion(x))
```

term	exper	school	wage
exper			
school	-.19		
wage	.05	.28	

That way, you get a correlation matrix without the significance level. You can still get them through the procedures of the blog posts above, but that will be longer code.

1.3 Getting nice output documents with Quarto

1.3.1 Reproducible R Reports

So far, we have been working purely with basic `.R` script files, which are very similar to Stata `.do` files. But R is a programming language that allows you to do MUCH more, including dynamic website (to share your data outputs and data online), or reports (the course you are reading was written within R).

To do so, R is able to combine R code with a markup language: Markdown. Markdown is a markup language, *i.e.* a language made to describe documents (Latex or html are two other famous markup language). Markdown is easy and intuitive: you simply write your text as is, and complete with some markups to indicate where are the headers, the text, the bullet points, the numbered lists, etc.

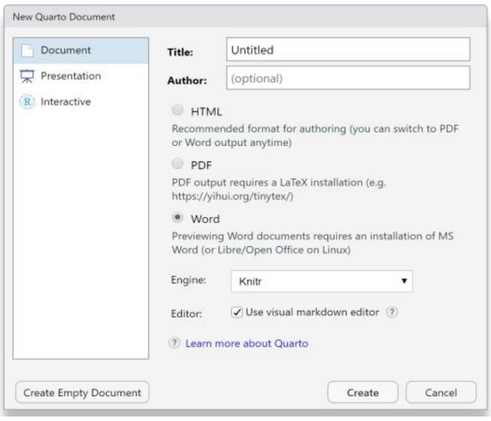
Once you have your nicely structured document, combining R code and Markdown text, you will need to *knit* the chunks of codes and markups into the format that people use for report: pdf, Word document, html page, PowerPoint, etc. In R, the knitting can be done through Quarto.

Quarto is a new module with R. Previously, documents were created with RMarkdown. Quarto works exactly the same way, but is more versatile, and compatible with other statistical solutions, such as Python and Julia. Also, quarto offers a visual editor: you can then edit your reports, articles, or thesis, directly in R, just as you would edit a Word document. Hence, I recommend the use of Quarto. You can find a detailed documentation here: https://quarto.org/docs/guide/.

This course was written with Quarto. You can find the corresponding `.qmd` files on the github repository of the course (see Module 1).

1.3.2 Creating an Quarto document

- To create an quarto document (file .qmd), go to File > New File > Quarto document…
- A popup shows up to ask enter the document Title and Author, as well as what type of document you want to create.



Title: Untitled

Author: (optional)

○ HTML
Recommended format for authoring (you can switch to PDF or Word output anytime)

○ PDF
PDF output requires a LaTeX installation (e.g. https://yihui.org/tinytex/)

● Word
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux)

Engine: Knitr

Editor: ☑ Use visual markdown editor

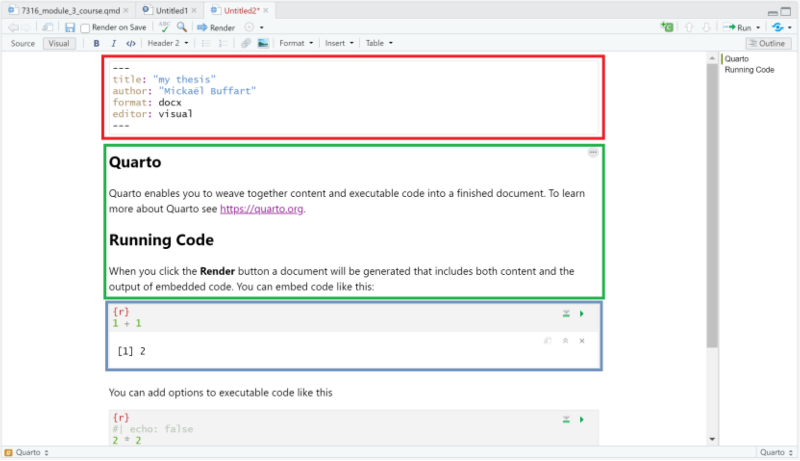
Learn more about Quarto

New RMarkdown file

- Note that TeX is required to generate pdf document. Also, generating pdf directly will often require combining Markdown with LaTex. Unless you already know and use LaTeX, this is overly complicated for no reason: I would rather advice to generate Word document, and save them as pdf from Word if you want pdf.
- In most cases, your best choice is likely Word document, or Powerpoint presentation, but you may choose as you see fit. Some options of Quarto are only available in specific formats.

1.3.2.1 Writing and Code in Quarto

Once you chose *Word*, and clicked OK, a visual editor appears, with an example code, as on the figure bellow:



```
---
title: "my thesis"
author: "Mickaël Buffart"
format: docx
editor: visual
---
```

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
{r}
1 + 1

 [1] 2
```

You can add options to executable code like this

```
{r}
#| echo: false
2 * 2
```

New Quarto file

The quarto visual editors contains 3 different types of blocks

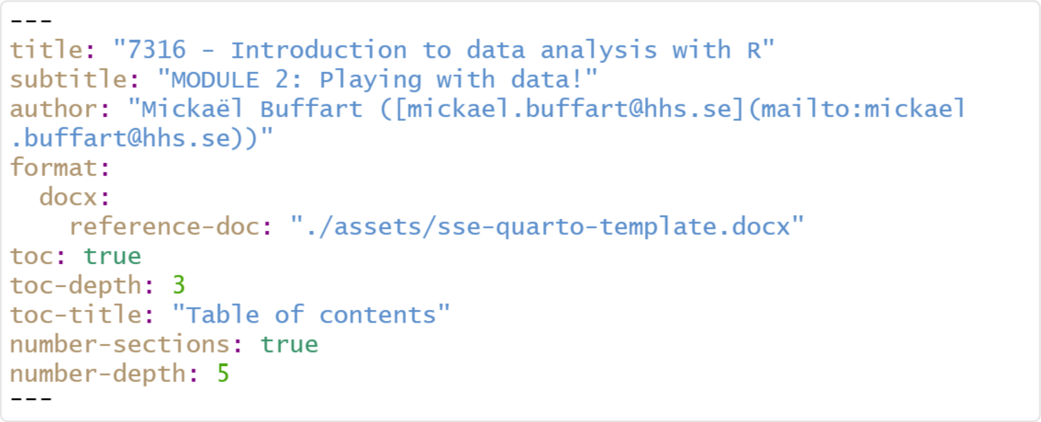
1. On top, (in the red square on the image), you can see the YAML metadata, where you set the properties of your document: title, author, type of output, etc. You may also set multiple types of output here.
2. The part in the green square, below, are just simple text, that you can edit with the buttons on top (Bold, Italic, code, choose a style, add hyperlink, insert images or tables, footnotes, references, etc.) just as a normal text editor.
3. Then, the part in the blue square, starting with `{r}` is a code chunk (*i.e.* a portion of R code). *A code chunk* is a portion of R code that will can be interpreted by Quarto, and the output displayed below it, in the output document. For example, the code chunk in blue in the image above contains the code `1+1` (in gray) and you can see bellow the output `[1] 2`. You generate the output of the chunk by clicking on the little green play button (top right corner of the code chunk), or when you `Render` the output document. Options allow you to choose which chunk have to be interpreted, or not, when rendering a document.

On top of the document, if you press `Render`, a Word document will be generated (and Word will open automatically, if you have it installed on your computer).

The content of the .qmd file is in fact a combination or R and Markdown code (the markup language defining what is a header, what is a text body, where are the bullet lists, etc.) that is interpreted in the visual editor. If you prefer to work on the `.qmd` source, you can click on the `source` button (on the top left). However, in this course, since the visual editor is very handy and very intuitive, we will not cover the specifics of the Markdown language. If you want to learn it, you can read *R markdown: The definitive guide*, that you will find in the reading list of module 1. I will now present some of the YAML and code chunk options that will be useful for your own documents.

1.3.3 YAML metadata

The YAML metadata define the general properties of your document, including the appearance and the output format. Bellow, the figure shows the options I used to generate the module 2 course document. The blue parts (in quotes) are the values, the brown parts are possible options.



```
---
title: "7316 - Introduction to data analysis with R"
subtitle: "MODULE 2: Playing with data!"
author: "Mickaël Buffart ([mickael.buffart@hhs.se](mailto:mickael
.buffart@hhs.se))"
format:
  docx:
    reference-doc: "./assets/sse-quarto-template.docx"
toc: true
toc-depth: 3
toc-title: "Table of contents"
number-sections: true
number-depth: 5
---
```

Some YAML options in quarto

1.3.3.1 Common output options

Here are a few common options:

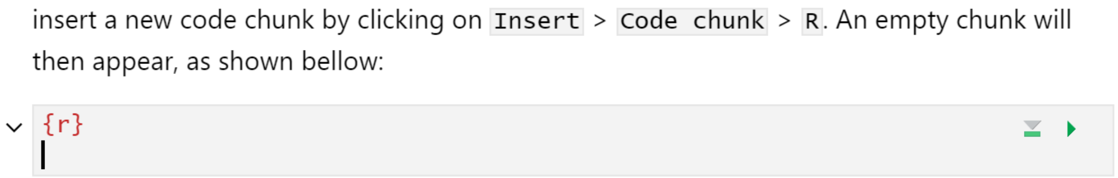
- `title:`, `subtitle:`, and `author:` are self-explanatory, written on the first page of the document you generate, or in the metadata.
- `format:` indicates the output format. Note that the value is indented on the next line because it is a suboptions. Here, I use a Word document (`docx`) output with a template (`reference-doc:`) placed in the `assets/` directory. The template indicates the style of my word file: margins, fonts, headers, etc. You can find the template of this course on canvas.
  - In general, I advise you to always generate word documents. This is usually what you want to get properly formatted tables and figures directly ready for your theses and articles.
  - If you wish to change the appearance of your Word document output, you will have to adjust the style in your template file (in the example: `sse-quarto-template.docx`)

- `toc: true` indicates that I want the output to include a table of content.
- With `toc-depth: 3`, the table of content should include 3 levels of headings
- `toc-title:` provides the header of the table of content.
- `number-sections:` true decides whether you want to number the headings in your output document.
- `number-depth:` indicates up to what level of heading you wish to number them

After this, you can simply type your output document. You can adjust the design and insert elements such as tables or figures using the buttons in the toolbar, just as in MS Word.

1.3.4 Code chunk

Now, if you want to write R code and display the output in your document, you need to insert a new code chunk by clicking on `Insert` > `Code chunk` > `R`. An empty chunk will then appear, as shown bellow:



insert a new code chunk by clicking on `Insert` > `Code chunk` > `R`. An empty chunk will then appear, as shown bellow:

```
{r}
```

Example of empty code chunk

In the code chunk, you can type and run code, just as in an R script.

1.3.4.1 Code chunk options

There are several output options you can specify for how R code and the code output are expressed in reports. These options are expressed bellow the red `{r}` declaration at the top of the chunk, on new lines (one line per option).

- `#| echo: FALSE`: the R code will not be displayed in the output document (but potentially the output of the code execution depending on other chunk options).
- `#| eval: FALSE`: the code inside the chunk will not be executed (*i.e.* there won't be an output), but the code is displayed in the output document.
- `#| results: "hide"`: the code will be executed, but the output of the execution will not be displayed in the output document.
- `#| include: FALSE`: the code will be executed, but nor the code, nor the output of the execution will be shown in the output document.
- `#| warning: FALSE` / `#| message: FALSE`: do not display warnings or messages associated with the R code.

- `#| fig-width:`, `#| fig-height:`, `#| fig-align: "center"` will be useful options in case the code chunk is generating a figure (with `ggplot2`, for example). `#| fig-cap:` is useful to define the figure title, and `#| dpi:` determines the resolution of the figure (a higher value means a better resolution, but also a more heavy file).
- You can also automatically number tables and figures, with 3 options:
  - For tables:
    - Add `#| label: tbl-tablename`, where `tablename` is the identifier of your table
    - `#| tbl-cap-location: top` indicates that you want the table title on top of the table
    - `#| tbl-cap: "the title to display"` will display the title in the format: Table 1. the title to display.
  - You can do the same with figures, by replacing `tbl` with `fig`.

You will find plenty of other useful options in the quarto documentation online: https://quarto.org/docs/guide/