PRU LearnR - Pedestrian to Novice Series

Prospect Padawans & RNinjas

revised version of 1 Jun 2024

Table of contents

Preface

This quarto-book was developed with the help of many colleagues. It builds on the material presented and discussed during the *LearnR* sessions at EUROCONTROL.

This online book/resource is *work-in-progress* and represents a trial-run of the use of `{webr}` to allow for interactive content in online books.

This skeleton was setup for the recap on `{ggplot2}` and a follow-up session on recapping more `{dplyr}` and `{ggplot2}` stuff, and then venturing into `{gt}`. It is planned to add the material from the first sessions as we go (and time permits).

While this might be disappointing for a first interaction. This could be the basis for transferring all our sessions into this format and support future "pedestrians" to get achieve the "novice" level … before embarking to become a "R/RStudio ecosystem ninja/jedi"!

May the forRce be with you!

1 Test chapter

When accessing the (book) page, you will see a `WEBR STATUS` loading symbol at the top of the page. This loads some of the required (underlying packages for this page). It may take a few seconds when you access this for the first time (and your browser has not yet cashed and downloaded the required package).

But it should be reasonable fast.

Conceptually, once you see the *"green" Ready!* label. All prep work is done and the ensuing code windows work.

Give it a try … and run the code!

```
# set a random seed and generate data
set.seed(123)
x <- rnorm(100)

# calculate mean
mean(x)
```

2 Visualisation - Plotting in R

2.1 Overview

Visualisations form a key component of our deliverables.
There are several plotting "systems" in R. This session will focus on plotting with `{ggplot2}` after introducing some *`base-R graphic` commands*. The latter might come handy for a quick orientation or initial data exploration. However, `{ggplot2}` offers more flexibility and extensibility when we want to produce publication-ready plots.

2.2 base-R plotting

Base-R uses the generic `plot()` function (and many objects have implemented a plot-function). So let's *explore* associations in the built-in `iris` dataset.
To be frank, `plot()` works here as it is implemented as a method for a R-object (i.e. the iris dataset).

```
plot(iris)
```

More generically, `plot(x,y)` accepts *vectors*. A simple scatter-plot of numerical values can then be called with the following commands.

```
# define the vectors
x <- -5:5
y <- x^2
# now plot
plot(x,y)
```

> **Exercise**
>
> The `mtcars` dataset is a built-in dataset in BaseR. It provides information on a series of car models.
> Make a scatter plot with (horsepower) `hp` on the x axis and (weight of the car) `wt` on the y axis. Do not worry about axis labels, etc.

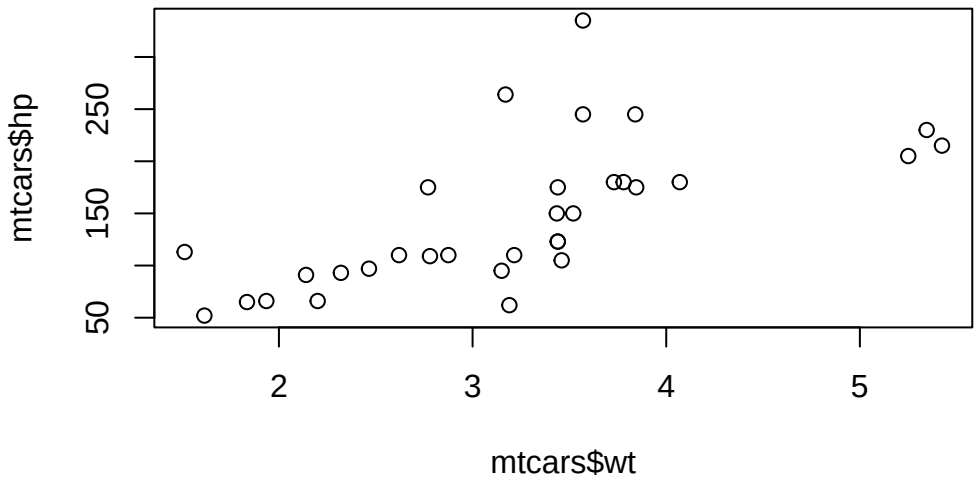
```
# Add you code here
```

Your result should look like the plot below

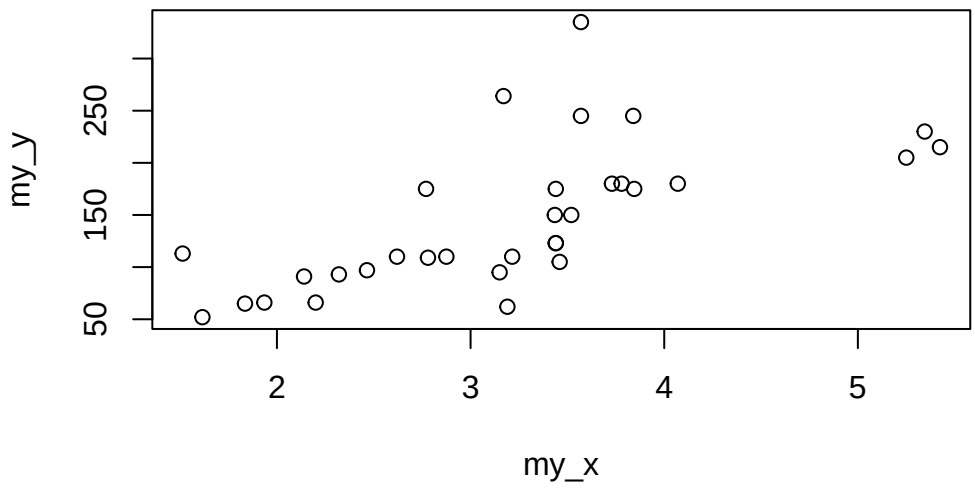
Expected Result

```
# plot wt versus hp of the mtcars dataset in BaseR
# the plot function uses vectors
# Note 1: you recall how to access vectors with the $ notation, don't you?
# Note 2: x = ... and y = ... is verbose.
# Recall however, position matters in function calls.
# Test this by swapping around x any y.

plot(x = mtcars$wt, y = mtcars$hp)
```




```
# you may also assign the vectors outside the plot() call, e.g.
my_x <- mtcars$wt
my_y <- mtcars$hp
plot(x = my_x, y = my_y)
```



2.3 Univariate plots - Get a feel for the data

2.3.1 Base-R histogram

It is often useful to get a feel for the distribution of variables/data values. Here a *histogram* or *boxplot* are useful exploratory helpers.
Before constructing a publication ready (gg)plot (c.f. next section), the base-R variants may help.

Please recall: base-R plotting works with vectors!

```
# let's check the distribution of the horsepower
# with the breaks argument, you can control the number of (breakdown)bins
hp_vec <- mtcars$hp
hist(hp_vec)
hist(hp_vec, breaks = 30)
```

2.3.2 Base-R boxplot

Exercise

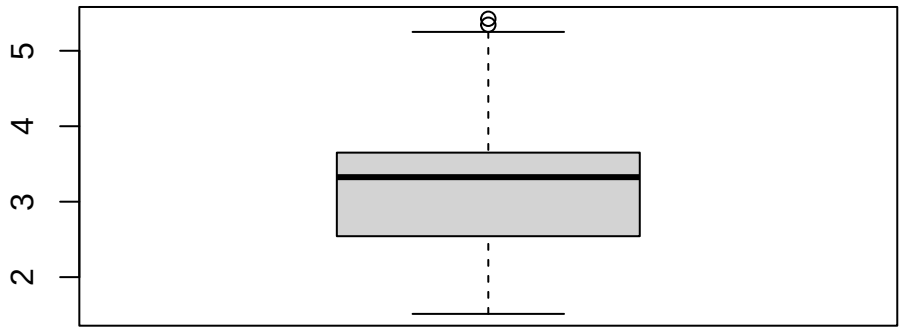
Another way of showing the distribution of values are so-called boxplots. The respective base-R call is `boxplot(x)`.
Using the built-in `mtcars` dataset, create a boxplot of horsepower `hp` and then a boxplot for the variable weight `wt`.

```
# Add you code here
```

Your result should look like the plot below

Expected Result

```
# boxplot of weight variable
boxplot(x = mtcars$wt)
```



2.4 Plotting with {ggplot2}

{ggplot2} is a popular plotting package of the {tidyverse}-family. The package implements the *grammar of graphics* which offers a conceptual framework to "construct" plots.
This allows for the generation of good plots quickly … that can then be iteratively *"beautified"* and made publication ready.

> Things to know/recall
>
> The underlying framework (i.e. *grammar of graphics*) considers constructing a visualisation as a series of composeable elements.
> This allows for the creation of "thousands of visualisation" instead of being restricted to a defined set of graphs in other software packages/apps.

Note: As you will learn in the following {ggplot2} works extremely well with a `tibble/dataframe` in the *long* format (recall: tidy data).

2.4.1 {ggplot2} Basics

- as usual, before working with a package, one has to load the package (and some data)
- please recall: if your setup has not yet the library available, you can download it with the command `install.packages("ggplot2")`.

```
library(ggplot2)
```

- To build a plot using {ggplot2} we start with the `ggplot()` function

```
# basic call to create the "canvas"
ggplot()
```

- `ggplot()` creates a base ggplot object, think about a painter's *blank canvas*. On this canvas we can then add *layer by layer* to establish our *picture*
- `ggplot()` accepts a series of optional arguments for information to be shared across different components of the plot
- The two main arguments we typically use here are
  - `data` - which is the name of the data frame we are working with, so `mtcars`
  - `mapping` - which describes which columns of the data are used for different aspects of the plot
- We create a `mapping` by using the *aesthetic* function `aes()` linking columns (~ vectors within our dataframe) to pieces of the plot

  - We typically start with telling `ggplot()` what values should be on the x and y axes

Let's plot the relationship betwen the horsepower `hp` of the *mtcars* dataset and the fuel consumption, mile-per-gallon `mpg`.

```
ggplot(data = mtcars, mapping = aes(x = mpg, y = hp))
```

- This doesn't create a final figure. This time the blank *canvas* from above is augmented with some additional (aesthetic) information on default values for the data and mapped columns (e.g. a range of the x-axis is established that covers the data points in `mpg`, labels for the x- and y-axis are added)
- To show (aka add) data to the plot, we add a so-called *`geom`*etric layer (or `geometry`)
- ggplot2 uses the `+` operator to add a new layer
- as a reasonable starting point for any bivariate (two variable) plot, we want to establish a scatter plot. The ggplot2 geometry for this is `geom_point()`

```
ggplot(data = mtcars, mapping = aes(x = mpg, y = hp)) +
  geom_point()
```

- To change things about the presentation of the layer we can pass additional arguments to the `geom` calls.
  - For example, we can set/change
    * the `size` argument of the points, let's set it to `3`
    * the `color` argument of the points, we'll set it to `"blue"`
    * the transparency of the points, i.e. `alpha` value, let's pick something semi-transparent between 0-100%, i.e. let's set it to 0.5

```
ggplot(data = mtcars, mapping = aes(x = mpg, y = hp)) +
  geom_point(size = 3, color = "blue", alpha = 0.5)
```

Exercise

Try changing these values to make the graph look like you want it to. Explore how the graph changes when setting different values for `size`, `color`, or `alpha`.
For example * set size to 1.5 or color to "lightgreen". * Remove one/some of these arguments (e.g. alpha).
What happens, when you remove `size = 1.5`?
P.S. Do not forget to hit `Run Code` after your changes ;)

```
# Adapt as appropriate
ggplot(data = mtcars, mapping = aes(x = mpg, y = hp)) +
  geom_point(size = 1.5, color = "lightgreen")
```

2.4.2 First improvements - towards production ready graphs

While `ggplot2` implements the *grammar of graphics* and speaks about *aesthetics* ... the default color-settings, including the "characteristic" background are – at least – cringeworthy (in the humble opinion of the editor).

Above serves as a good basic plot. To use the graph in a report of presentation the following might help:

- make the axis-labels "human-readable", and possibly add a title
- choose another background

Labels serve as a documentation for your graph and can be added with the `labs()` function (layer). `labs()` accepts arguments for * `title`, `subtitle`, and `caption` * the axes, i.e. `x` and `y`. * you provide the desired `string`, e.g. `"Miles per Gallon"` to the argument

```
ggplot(data = mtcars, mapping = aes(x = mpg, y = hp)) +
  geom_point(size = 3, color = "blue", alpha = 0.5) +
  labs(x = "Miles per Gallon", y = "Horse Power",
       title = "Car stuff")
```

2.4.3 Grouping

- Group on a single graph
- Look at influence of experimental treatment

```
ggplot(data = mtcars, mapping = aes(x = mpg, y = hp, color = cyl)) +
  geom_point(size = 3, alpha = 0.5)
```

- Try changing the above code to color based on the `gear`
- We can also split each group into different subplots (known as facets)

```
ggplot(data = mtcars, mapping = aes(x = mpg, y = hp)) +
  geom_point(size = 3, alpha = 0.5) +
  facet_wrap(~cyl)
```

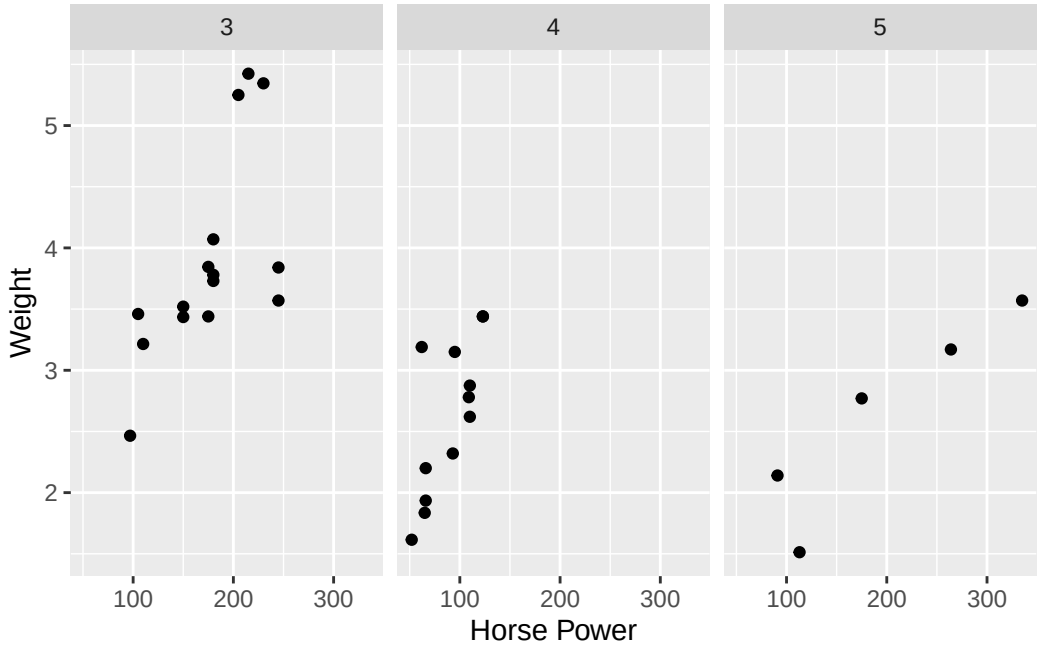
- Try changing this code to create a subplot for each value in `vs` with points of size 4

> **Exercise**
>
> Make a scatter plot with `hp` on the x axis and `wt` on the y axis. Label the x axis "Horse Power" and the y axis "Weight". Make one subplot for each value in `gear`.

```
# Add you code here
```

Your result should look like the plot below

Expected Result



Solution Code

```
library(ggplot2)
ggplot(mtcars, aes(x = hp, y = wt)) +
  geom_point() +
  labs(x = "Horse Power", y = "Weight") +
  facet_wrap(~gear)
```

3 Publication-Ready Tables in R

> **Note**
>
> This section is a demo/show only.
> {gt} appears to be not supported under {webr} … the latter being required for the interactive code-chunks.
> Thus, practice by hacking this into posit.cloud or your local RStudio installation.

3.1 Overview

RMarkdown has a primitive support for tables. This works fine during the development. I save showcasing the native RMarkdown syntax - just cumbersome for tables of more than a few rows.
However, publications deserve better.

```
# show the first 10 rows of the mtcars dataset
mtcars[1:10,]
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

There exists a variety of packages that help with the presentation of tables. Some of these table-packages are

- `{kable}` and its supporting package `{kableExtra}` - for pdf output
- `{flextable}` - offered good output for html, pdf, and docx
- `{gt}` a new(er)kid on the block and fitting nicely into the {tidyverse} ecosystem

The `{gt}` package conceptualises *tables* composed of a cohesive set of table parts. Similar to `{ggplot}`'s *grammar-of-graphics*, `{gt}` allows for the construction of a wide variety of useful tables based on these parts.
These include the table header, the stub, the column labels and spanner column labels, the table body, and the table footer.

```
knitr::include_graphics("./figures/gt_parts_of_a_table.png")
```

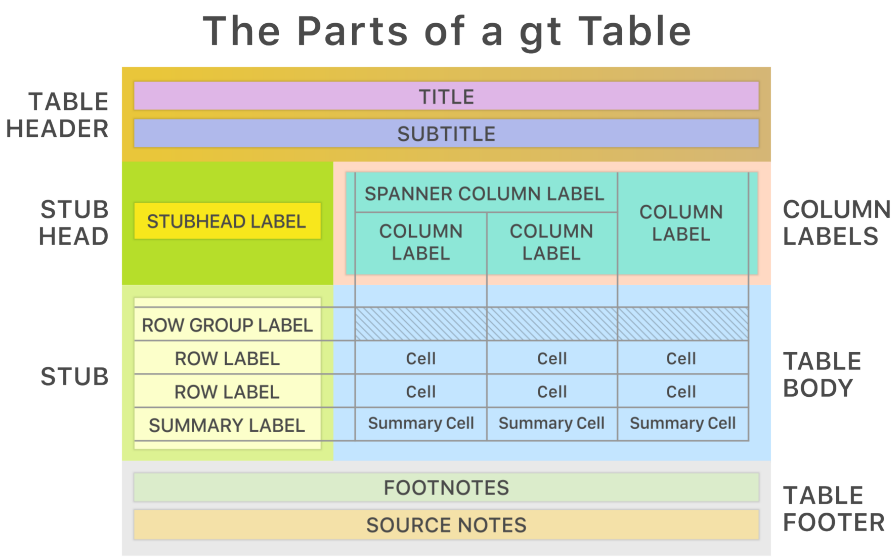


Figure 3.1: {gt} parts of a table

3.2 Getting goint with `{gt}`

- to work with `{gt}`, we first load the library
- the key table function is `gt()` which allows to set a couple of arguments (if needed)

Note: `mtcars` stores the model names in a *special* manner. To make this work, we first create a separate `model` column.

```r
library(dplyr)
library(gt)

# prepare the dataset - move the model names into separate column
# force this new column to the beginning (before the 1st column)
my_cars <- mtcars[1:10,] |>
  mutate(models = rownames(mtcars[1:10,]), .before = 1)

# work with {gt}
my_cars |>
  gt()
```

models	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

3.2.1 Organising the table content

Occassionally, there is a need to organise subsets of the columns.
This could be done as arguments of the `gt()` call.

Assuming our data has respective grouping variables, one can set these with `gt(rowname_col = ..., groupname_col = ...)`

For example we can group our data set according to the number of `gear`s. This inserts the `grouping` into the table presentation, i.e. there are grouping rows separating the different groups.

```r
my_cars |>
  gt(
    groupname_col = "gear"
  )
```

models	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	carb
4										
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	1
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4
3										
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	4

3.2.2 Labelling Groups of columns – `spanner`

Dependent on the data, one can introduce additional column (group) labels. `tab_spanner()` allows to combine columns under a separate (group) label. It accepts the `label` and `columns` argument. We can use standard dplyr-indexing to group columns. Note that when using a named vector, the sequence of columns might be influenced. Below the `carb` column is shifted.

Note: use a separate `tab_spanner()` per desired grouping.

```
my_cars |>
  gt() |>
  tab_spanner(label = "first block", columns = mpg:drat) |>
  tab_spanner(label = "other block", columns = c("qsec","vs","carb"))
```

	first block						other block				
models	mpg	cyl	disp	hp	drat	wt	qsec	vs	carb	am	gear
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	4	1	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	4	1	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	1	4
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	1	0	3
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	2	0	3
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	1	0	3
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	4	0	3

Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	2	0	4
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	2	0	4
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	4	0	4

3.2.3 Adding title, subtitle, and footnotes

```
my_cars |>
  gt() |>
  tab_header(
     title = "Title of my_cars table"
    ,subtitle = "Any useful subtitle information can go here"
    )
```

Title of my_cars table

Any useful subtitle information can go here

models	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

Next, we want to set a footnote. Here we can differentiate between

- *lazy* table footnotes (that should be sourcenotes ;))
- {gt} sourcenotes (optional) to account for references
- {gt} footnotes to cross-reference table cells

```
my_cars |>
  gt() |>
  tab_footnote(
    footnote = "any footnote text"  # ignoring the location argument
  ) |>
```

```
  tab_source_note(
    source_note = "a source note does not expect a reference to any cell body"
  )
```

models	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

any footnote text
a source note does not expect a reference to any cell body

The power of `footnotes()` is that it can be combined with a reference to any part of the table.
There exists various helper for the referencing. In most cases, we want to reference one (or several) table cells with `cells_body()`.

`cells_body()`

- expects a `columns` and/or `rows` argument
- one could use dplyr-indexing
- in particular, `rows` accepts logical conditions

Let's try this (we truncate the table to the 1st, 9th, and 10th row):

```
my_cars[c(1,9,10),] |>
  gt() |>
  tab_footnote(
      footnote  = "One of the coolest car models ever built!"
    , locations = cells_body(
                        columns = models               # ref to column
                      , rows    = models == "Merc 280" # a logical condition
                      )
    ) |>
  tab_footnote(
```

```
      footnote  = "Six cylinders are two more than four"
    , locations = cells_body(columns = cyl, rows = cyl == 6)
  )
```

models	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	[1] 6	160.0	110	3.90	2.62	16.46	0	1	4	4
Merc 230	22.8	4	140.8	95	3.92	3.15	22.90	1	0	4	2
Merc 280[2]	19.2	[1] 6	167.6	123	3.92	3.44	18.30	1	0	4	4

[1]Six cylinders are two more than four
[2]One of the coolest car models ever built!

Note: similar to the layering of a ggplot, the order of the footnotes follows the `last-on-top` principle.

The aforementioned referencing allows to also highlight specific cells by formatting its presentation.
For the following we use `tab_style()` and the `cell_fill()` helper.

```
my_cars[c(1,9,10),] |>
  gt() |>
  tab_style(
      locations = cells_body(columns = cyl, rows = cyl == 6)
    , style     = cell_fill(color = "green")
  )
```

models	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.62	16.46	0	1	4	4
Merc 230	22.8	4	140.8	95	3.92	3.15	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.44	18.30	1	0	4	4

3.2.4 Theming `{gt}` tables - opt_stylize

`{gt}` provides a set of pre-defined table style options.
These styles allow for (smaller) adaptations:

- `style` pick one of the pre-defined styles, currently 1 through 6, default is 1.
- `color` allows to select from a limited set of colors. There are six color variations: “blue”, “cyan”, “pink”, “green”, “red”, and “gray”.

```r
my_cars[c(1,9,10),] |>
  gt() |>
  opt_stylize(style = 2, color = "cyan")
```

models	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.62	16.46	0	1	4	4
Merc 230	22.8	4	140.8	95	3.92	3.15	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.44	18.30	1	0	4	4

4 Communicate - Output

4.1 Overview

Throughout the past sessions we have implicitly *rendered* output to communicate our results. `Quarto` supports the production of a variety of documents in a reproducible manner.

Authoring `Quarto`-documents is done with `(R)Markdown` - a *light-weight* syntax around plein text [1].

From a processing perspective, the input (a combination of *text* + *code* + *visual elements*) is "converted" to the different outputs formats. This process takes the input, interprets the R (or other code) to (pure) Markdown. Pandoc takes the markdown and converts it into the desired output format [2].

Quarto supports the creation of *static* and *interactive* content dependent on the chosen output format. This offers the opportunity to expand content beyond the classical categories of 'report'/'technical paper' and can serve to establish interactive version of static reports, presentations, and dashboards.

This section provides a quick-step guide to/through

- (R)Markdown
- Quarto document structure and its YAML

4.2 A FL300 look at a *"document"*

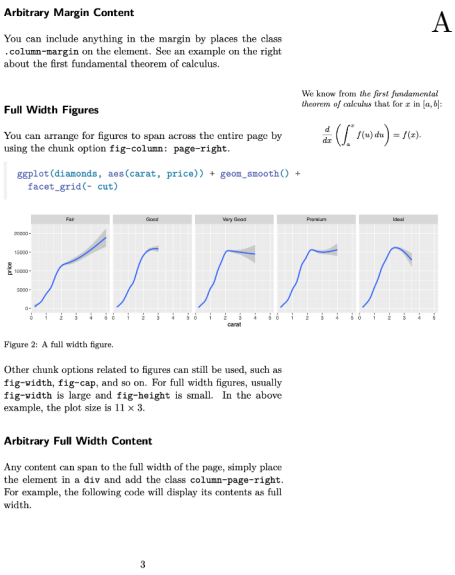
4.3 (R)Markdown

The philosphy behind (R)Markdown is to use plain text format that is easy to write and even easy to read. This reduces the cognitive load and allows to focus on the writing/editing.

[1] Thus, conceptually, we can edit/author a Qmd-document in any text editor - which ensures long-term survivability. Rendering a Qmd-document, however, requires an environment in which the *marked* text is interpreted.

[2] If you are interested in tinkering with *`pandoc`*, have fun at: https://pandoc.org/try/.

Figure 4.1: Artwork by @allison_horst



Arbitrary Margin Content

You can include anything in the margin by places the class `.column-margin` on the element. See an example on the right about the first fundamental theorem of calculus.

We know from *the first fundamental theorem of calculus* that for x in $[a, b]$:

$$\frac{d}{dx}\left(\int_a^x f(u)\,du\right) = f(x).$$

Full Width Figures

You can arrange for figures to span across the entire page by using the chunk option `fig-column: page-right`.

```
ggplot(diamonds, aes(carat, price)) + geom_smooth() +
  facet_grid(~ cut)
```


Figure 2: A full width figure.

Other chunk options related to figures can still be used, such as `fig-width`, `fig-cap`, and so on. For full width figures, usually `fig-width` is large and `fig-height` is small. In the above example, the plot size is 11×3.

Arbitrary Full Width Content

Any content can span to the full width of the page, simply place the element in a `div` and add the class `column-page-right`. For example, the following code will display its contents as full width.

Figure 4.2: Example document

A document consists of

- content
  - text, code, and visualisations
- structure
  - layout
  - paragraphs or other blocks of content
- appearance
  - fonts, colour, etc.
- format
  - overall output and functionality (i.e. static, interactive)

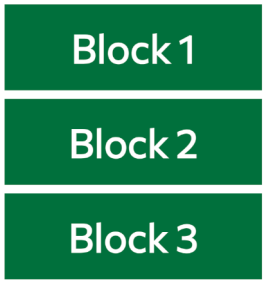


Figure 4.3: Document as a sequence of blocks

Thus, a document is a sequence (or list) of *blocks* that contain *inline elements* or other blocks. Such blocks and elements can be associated with *metadata*.A block element starts on a new line and is followed by an empty line. It forms an identifiable block.

4.4 Text Formatting

Markdown Syntax	Output
`This is some creative text ... just type`	… well … I hope you are not expecting something here …. that would be different from what
`*italics*, **bold**, ***bold italics***`	*italics*, **bold**, ***bold italics***
`superscript^2^ / subscript~2~`	superscript^{2} / subscript$_{2}$
`~~strikethrough~~`	~~strikethrough~~
`[This text is underlined]{.underline}`	This text is underlined

(R)Markdown	Output
highlight text (italic, bold): `*italic text* and **bold text**`	*italic text* and **bold text**
super- and subscript: `superscript^up^ and subscript~down~`	superscript^{up} and subscript$_{down}$
underline (using Pandoc): `my [text to underline]{.underline}`	my text to underline
strikethrough: `~~text stricken through~~`	~~text stricken through~~