PRU LearnR - Pedestrian to Novice Series

Prospect Padawans & RNinjas

revised version of 26 May 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.

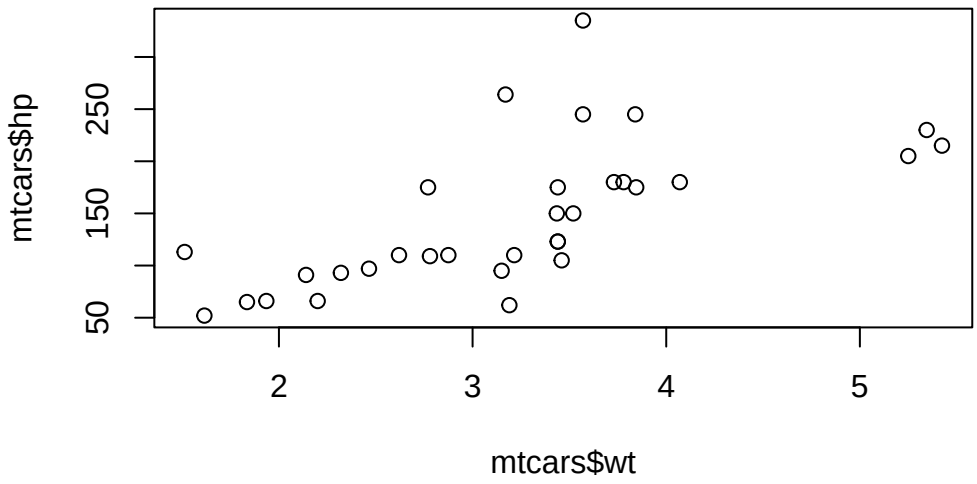
```
# Add you code here
```

Your result should look like the plot below

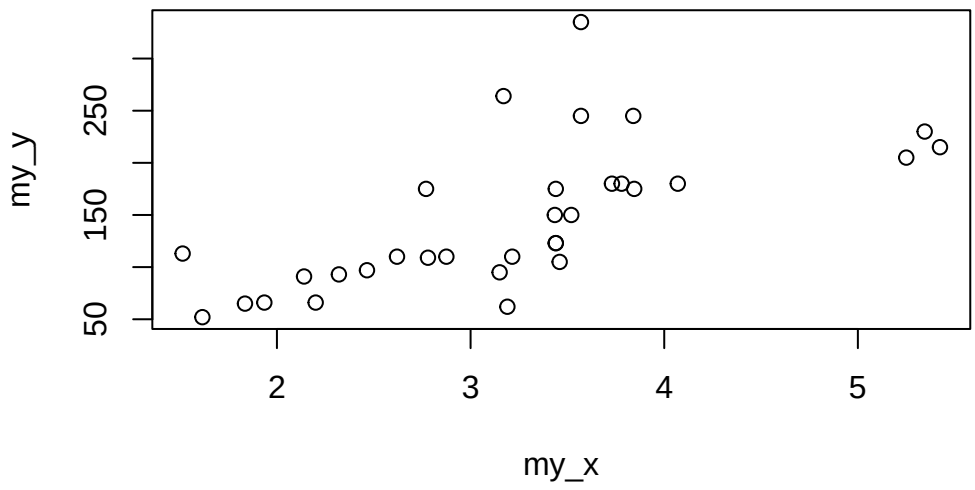
Expected Result

```
# 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

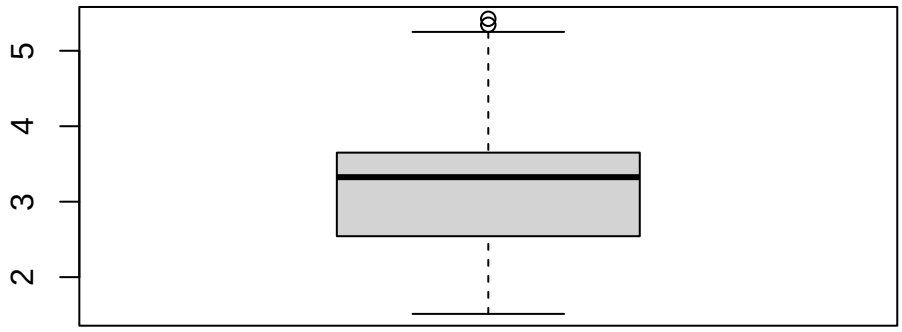
Another way of showing the distribution of values are so-called boxplots. The respective base-R call is `boxplot(x)`.
Using the built-in `mtcars` dataset, create a boxplot of horsepower `hp` and then a boxplot for the variable weight `wt`.

```
# 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)
```

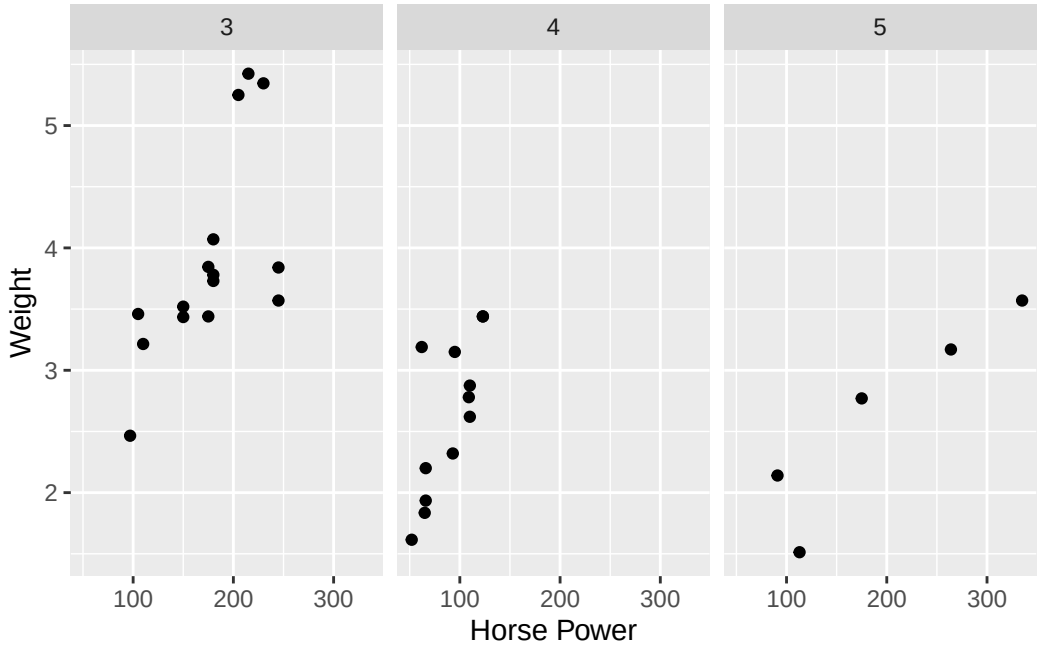
- Try changing this code to create a subplot for each value in `vs` with points of size 4

> **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

3.1 Overview

```
library(gt)
```

```
my_tbl <- mtcars[1:10,]

my_tbl
```

why is this not working