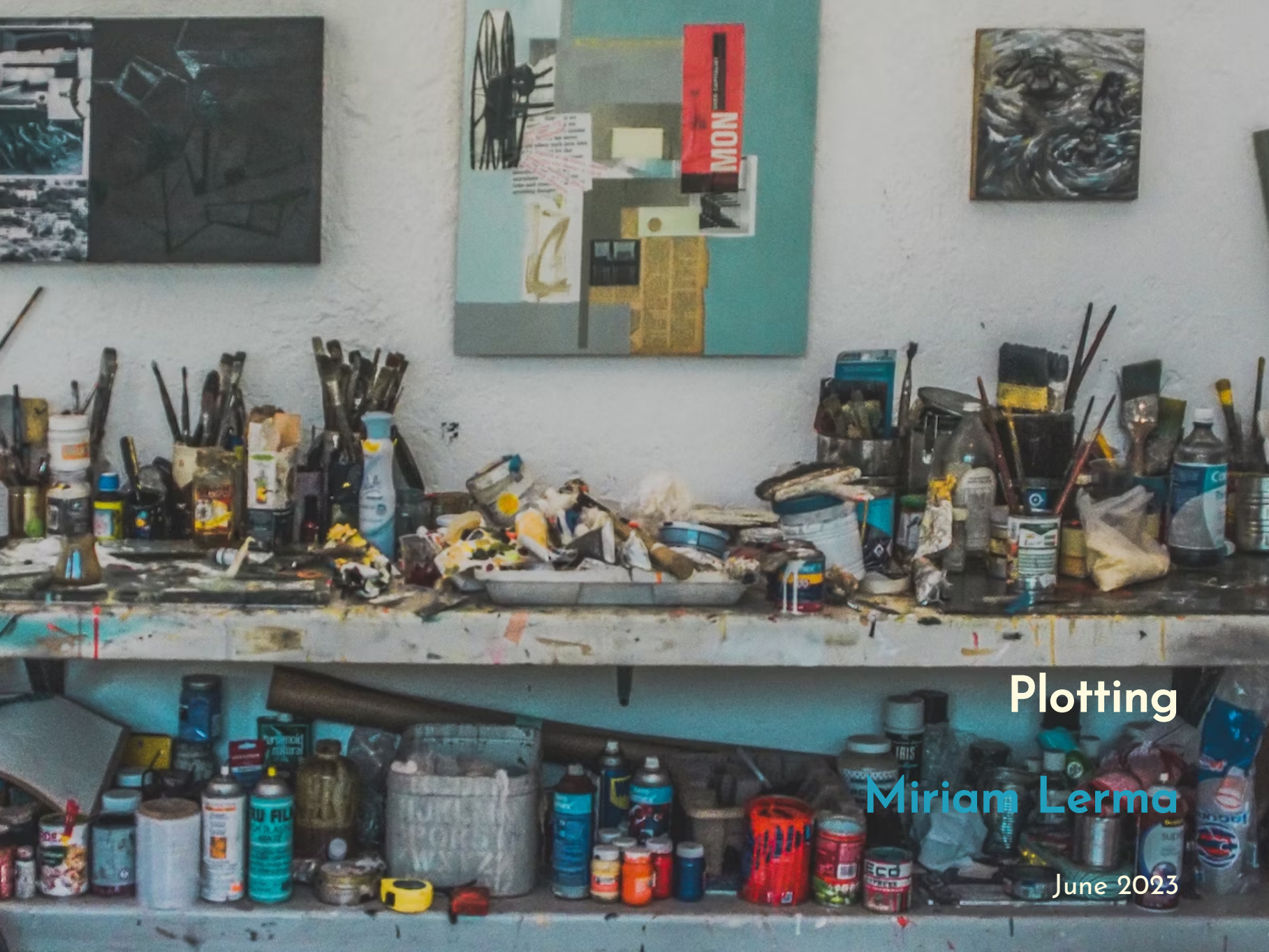


Plotting

Miriam Lerma

June 2023

Index

Today

Your profile

- You know how to use a Rmd document
- You know how to install packages

Goals of today

- Create plots and maps in R
- Change the apparence of the plots
- Export you plots

Pauses and questions

- Exercises and 10 minute pauses for catching up
- You can stop me to ask questions or use this link

We will use the packages

```
devtools::install_github("MiriamL
devtools::install_github("MiriamL
devtools::install_github("MiriamL
install.packages('sf')
install.packages('ggspatial')
install.packages('plotly')
```

References

Tutorials/Books ggplot

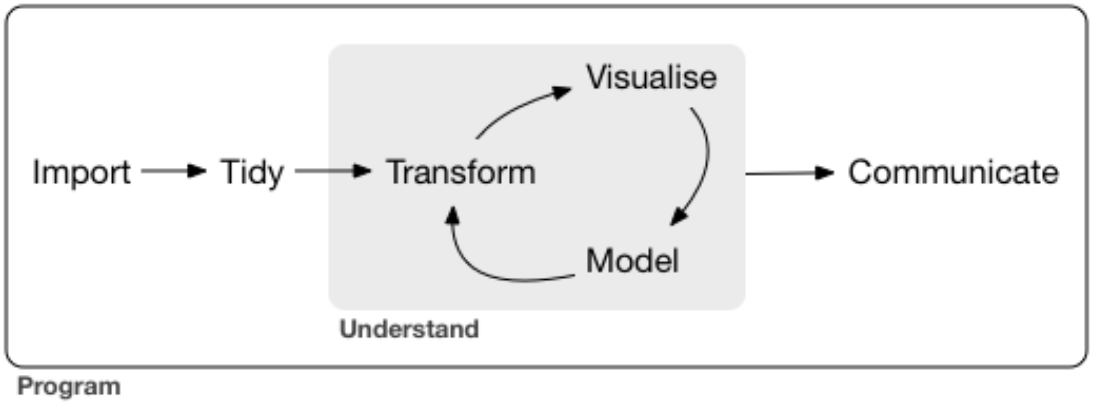
- gg is for Grammar of Graphics
- Introduccion to ggplot
- r4ds
- R Graphics Cookbook
- Allison Horst tutorial
- Coolors
- Palettes
- Size, Shape, Lines
- Geocomputation with R
- Earth Data Science
- r-spatial

Gallery/Portafolios

- Data-Viz-Bookmarks
- RGraphGallery
- DataToViz
- Georgios-Mastodon

1. Theory

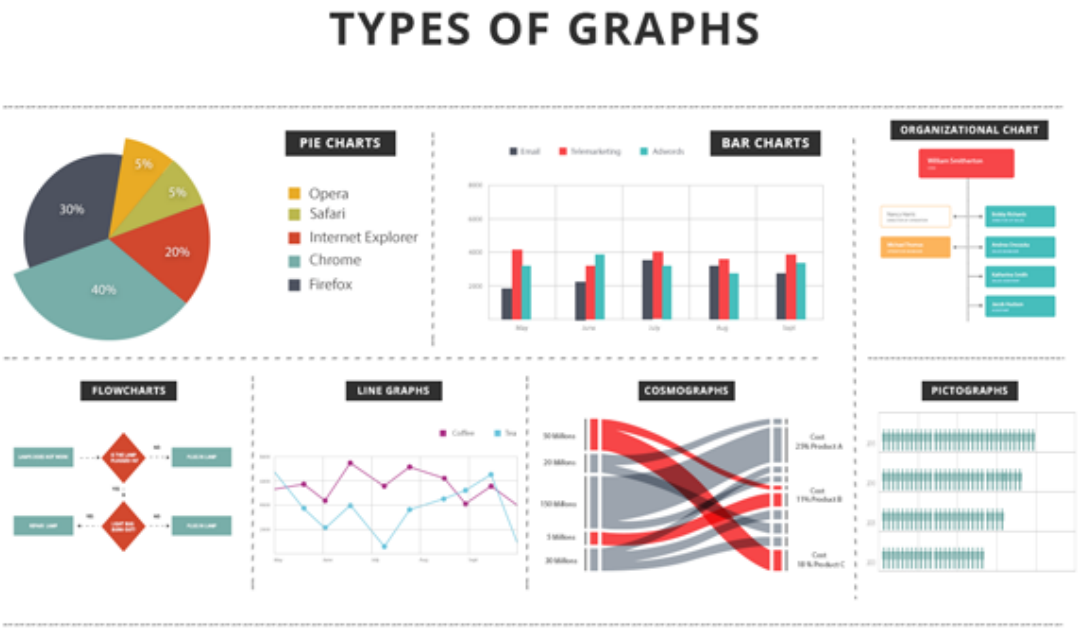
A typical R project looks like this:



Source: R4DS

1.1. Theory

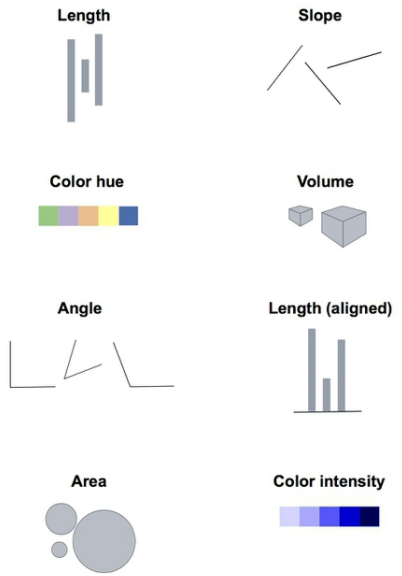
The objective of a graph/plot/map/chart is to communicate.
In science, the idea is to show information in a clear and understandable way.



Fuente: VizThinker

1.1. Theory

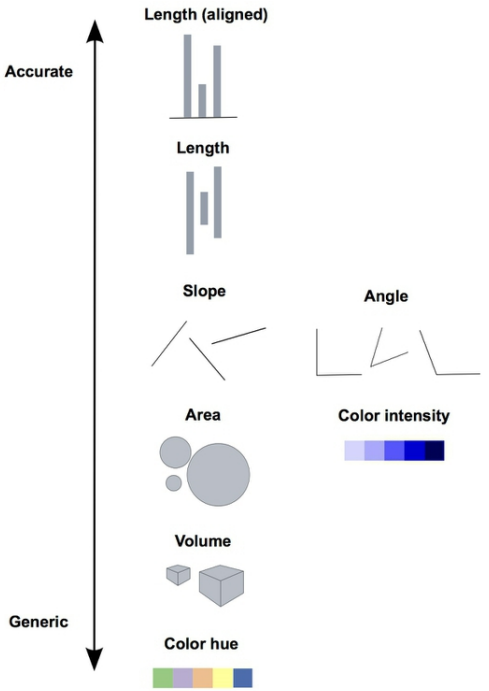
Plotting is transforming data to visualizations that vary in shape, size and colors.



Fuente: VizThinker

1.1. Theory

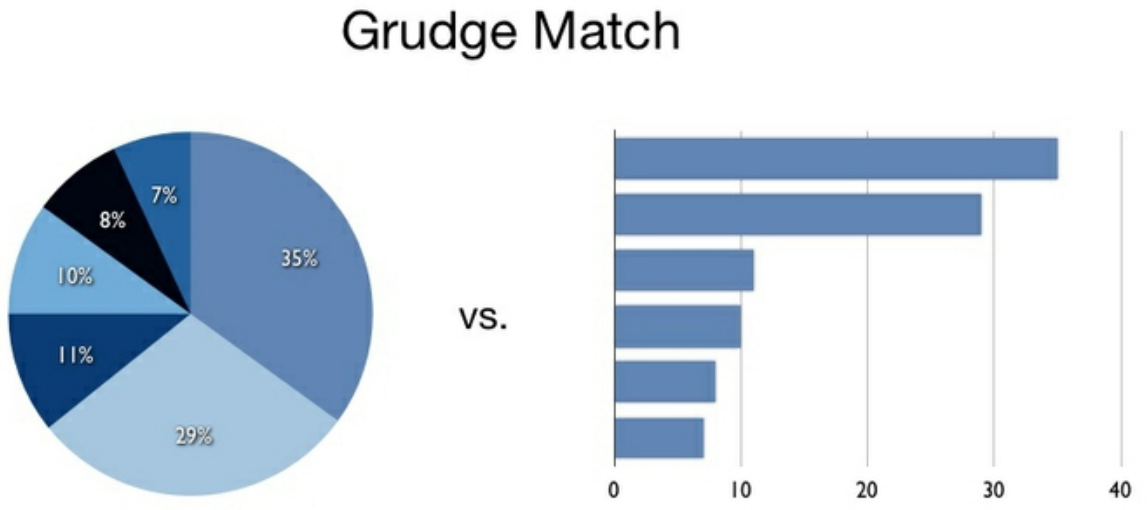
There are many ways to plot data, and different ways to interpretate the graphs.



Fuente: VizThinker

1.1. Theory

Which one is more intuitive?



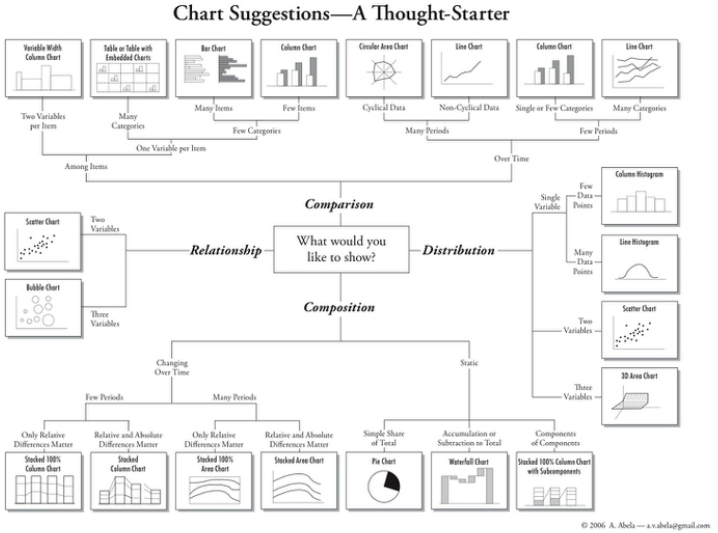
Fuente: VizThinker

1.1. Theory

Which one should I use?

You can try different plots and choose the one that betters tells the story.

Tableau Public has a suggestion on which graph to use according to the type of data you are using.

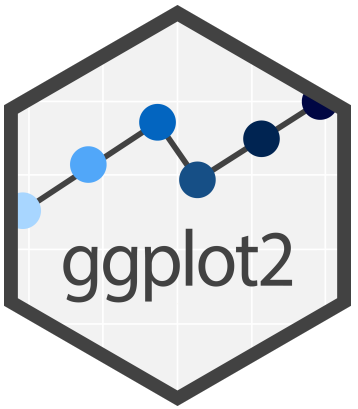


1.2. Intro

There are many packages to create plots in R.

- Here we will use **ggplot2**.

ggplot is the most versatile one, and from which you can find more help and inspiration.



1.2. Intro

It can be divided in three main components:

- **Data** (data): which are the variables that we plot.
- **Geometry** (geom): a geometric object such as dots, lines, bars.
- **Aesthetics** (aes): the aesthetic attributes to a geometric object, such as position, color, shape and size.

Each component is displayed as **layers**.

Source ggplot2 book

1.3. Data

The package ggplot is already included in tidyverse.
If you have installed tidyverse you just need to call your package.

```
library(tidyverse)
```

If not you can also install it separately.

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

1.3. Data

For the exercises, we can use data from packages.
Lets add our object to our **environment**.

Package with data

```
library(palmerpenguins)
penguins<-penguins
```

Call library **ggplot2**.

```
library(ggplot2)
```

1.3. Data

First step: add data to a plot.

```
ggplot(data=penguins)
```

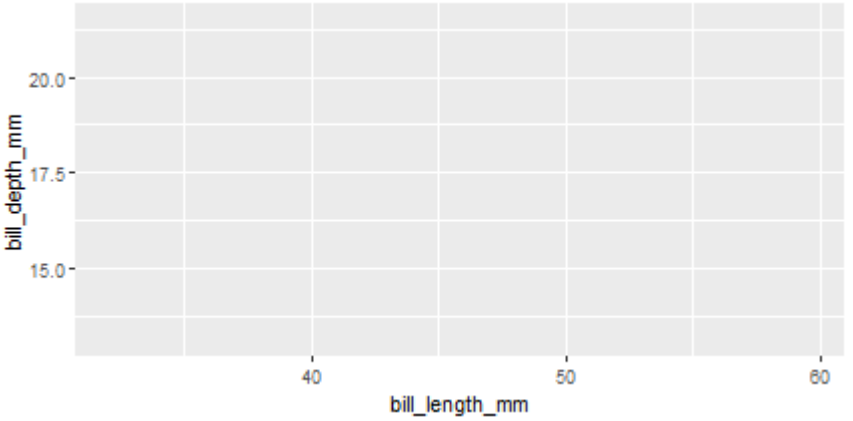

We told ggplot what data frame we want to use, but need to give instruction on how to plot.

1.3. Add axis

Using **mapping** we tell ggplot what to plot in the x and y axis.

Note that the axis are columns from your data frame. This is when having tidydata is important.

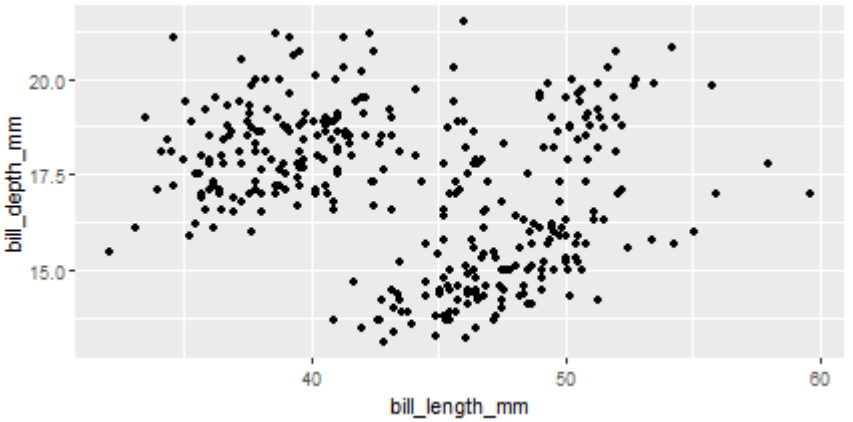
```
ggplot(data=penguins,
       mapping=aes(x=bill_length_mm,
                   y=bill_depth_mm))
```



1.3. Geometry

The next step is to give it a geometry, in this case we say plot points with the argument **geom_point**

```
ggplot(data=penguins,
       mapping=aes(x=bill_length_mm,y=bill_depth_mm))+
       geom_point()
```



It would likely show a **warning**.
This is because NAs are not going to be plotted and R is trying to warn you.

1.3. Geometry

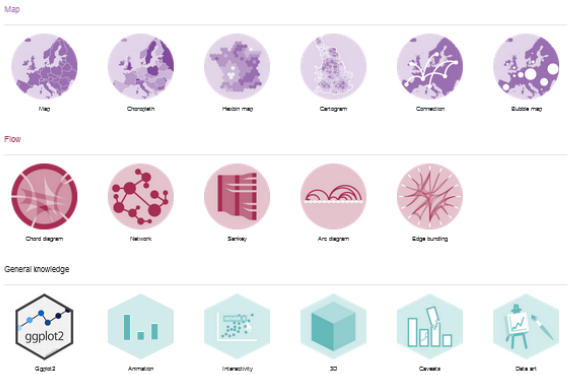
There are many **geometries**.

Most of them start with **geom_***

Type	Function
Point	`geom_point()`
Line	`geom_line()`
Bar	`geom_bar()`, `geom_col()`
Histogram	`geom_histogram()`
Regression	`geom_smooth()`
Boxplot	`geom_boxplot()`
Text	`geom_text()`
Vert./Horiz. Line	`geom_{vh}line()`
Count	`geom_count()`
Density	`geom_density()`

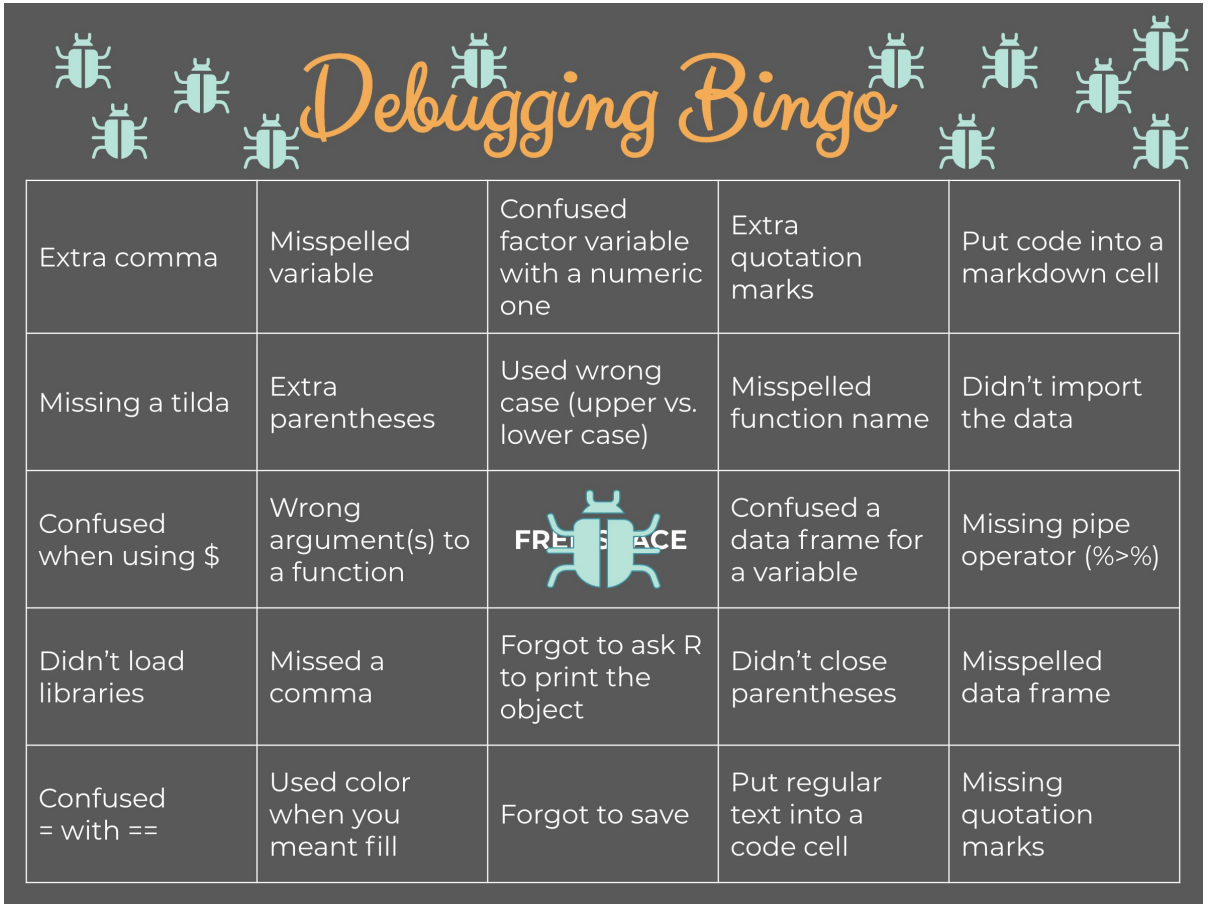
1.3. Geometry

Examples of most common geometries are to be found on R-graph-gallery.



1.4. Common mistakes

The most common mistakes with plotting are the same as while writing code.



1.4. Common mistakes

Other common mistakes specific to ggplot are:

- Forgetting a plus symbol (+)

```
ggplot(data=penguins,
       mapping=aes(x=bill_length_mm,y=flipper_length_mm))
  geom_point()
```

- Adding the + on the wrong order

```
ggplot(data=penguins,
       mapping=aes(x=bill_length_mm,y=flipper_length_mm))
 +geom_point()
```

- Forgetting a parenthesis

```
ggplot(penguins,
       aes(x=bill_length_mm,y=flipper_length_mm)+
       geom_point()
```

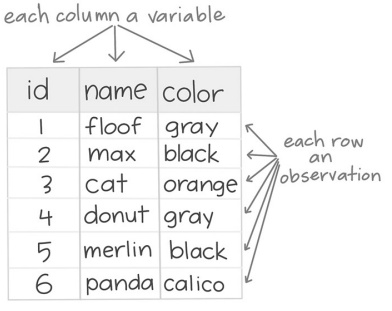
1.4. Common mistakes

Not having **tidydata** is also a very common problem.

> "TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."
>
> –HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

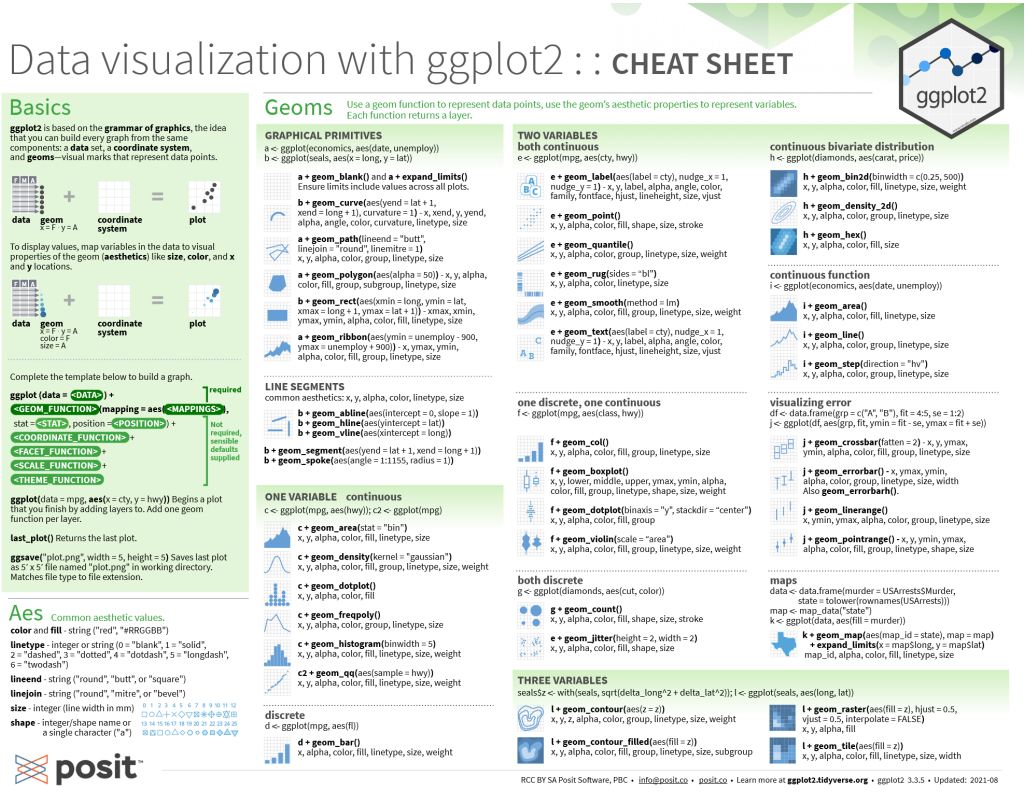


Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

Source Allison Horst

1.5. Cheat sheet

Help>CheatSheets>Data Visualization with ggplot



Data visualization with ggplot2 : : **CHEAT SHEET**

ggplot2

Basics

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

data | geom (x = F, y = A) | coordinate system | plot

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

data | geom (x = F, y = A, color = F, size = A) | coordinate system | plot

Complete the template below to build a graph.

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

required | Not required, sensible defaults supplied

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

last_plot() Returns the last plot.

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

Aes

Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

Geoms

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

GRAPHICAL PRIMITIVES

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

a + geom_blank() and **a + expand_limits()** Ensure limits include values across all plots.

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

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

a + geom_polygon(aes(alpha = 50)) - x, y, alpha, color, fill, group, subgroup, linetype, size

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

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

LINE SEGMENTS

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

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

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

ONE VARIABLE continuous

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

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

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

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

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

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

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

discrete

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

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

TWO VARIABLES both continuous

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

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

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

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

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

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

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

one discrete, one continuous

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

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

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

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

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

both discrete

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

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

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

THREE VARIABLES

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

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

l + geom_contour_filled(aes(fill = z)) x, y, alpha, color, fill, group, linetype, size, subgroup

continuous bivariate distribution

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

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

h + geom_density_2d() x, y, alpha, color, group, linetype, size

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

continuous function

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

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

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

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

visualizing error

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

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

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

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

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

maps

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

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

posit

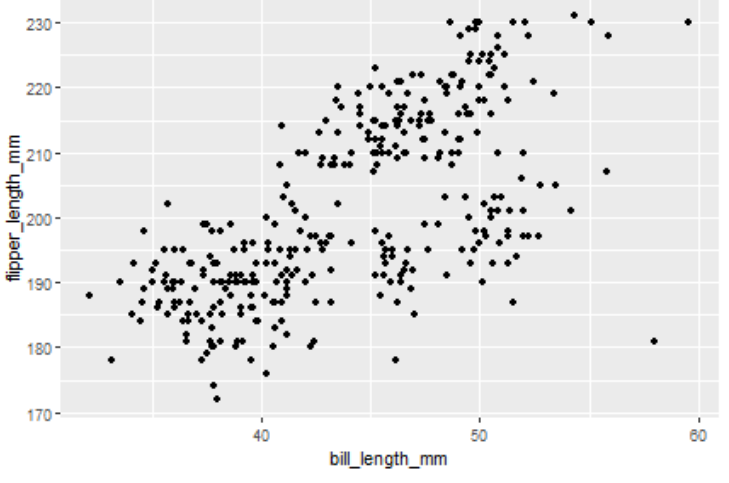
1.6. Simplify

We can skip writing **data=** and **mapping=** by just following the order of the elements.

Both codes give the same output.

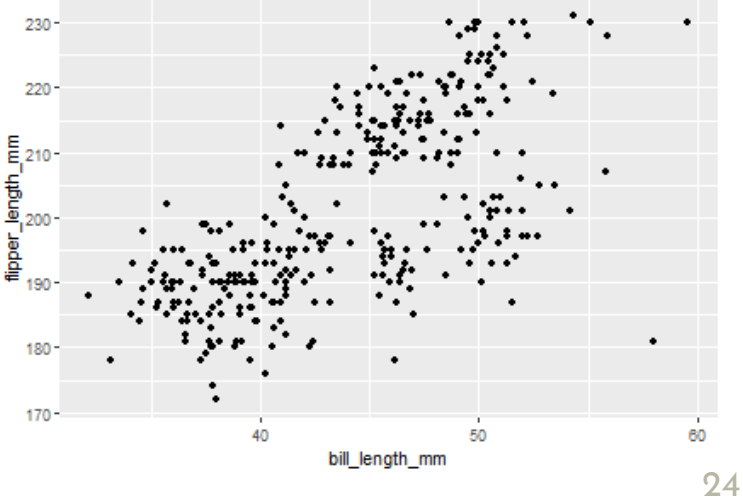
Including the elements.

```
ggplot(data=penguins,
       mapping=aes(x=bill_length_
                   y=flipper_leng
  geom_point()
```



Simplified example.

```
ggplot(penguins,
       aes(x=bill_length_mm,
           y=flipper_length_mm,))
  geom_point()
```

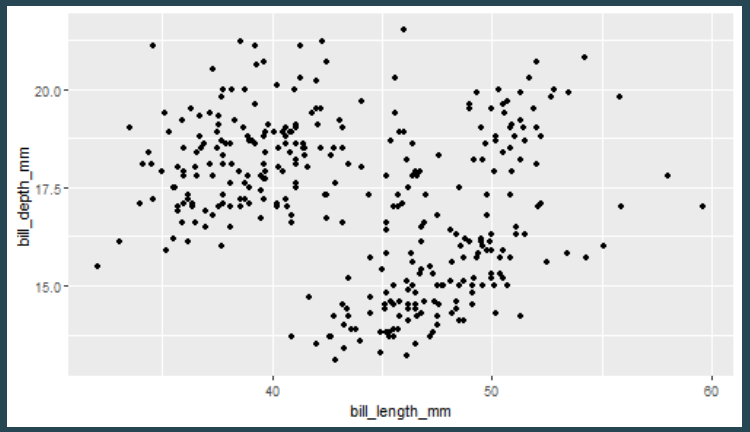


Pause

Practice
Open the file 03ExercisesPlotting.Rmd

Create your first plot in R.

Outline

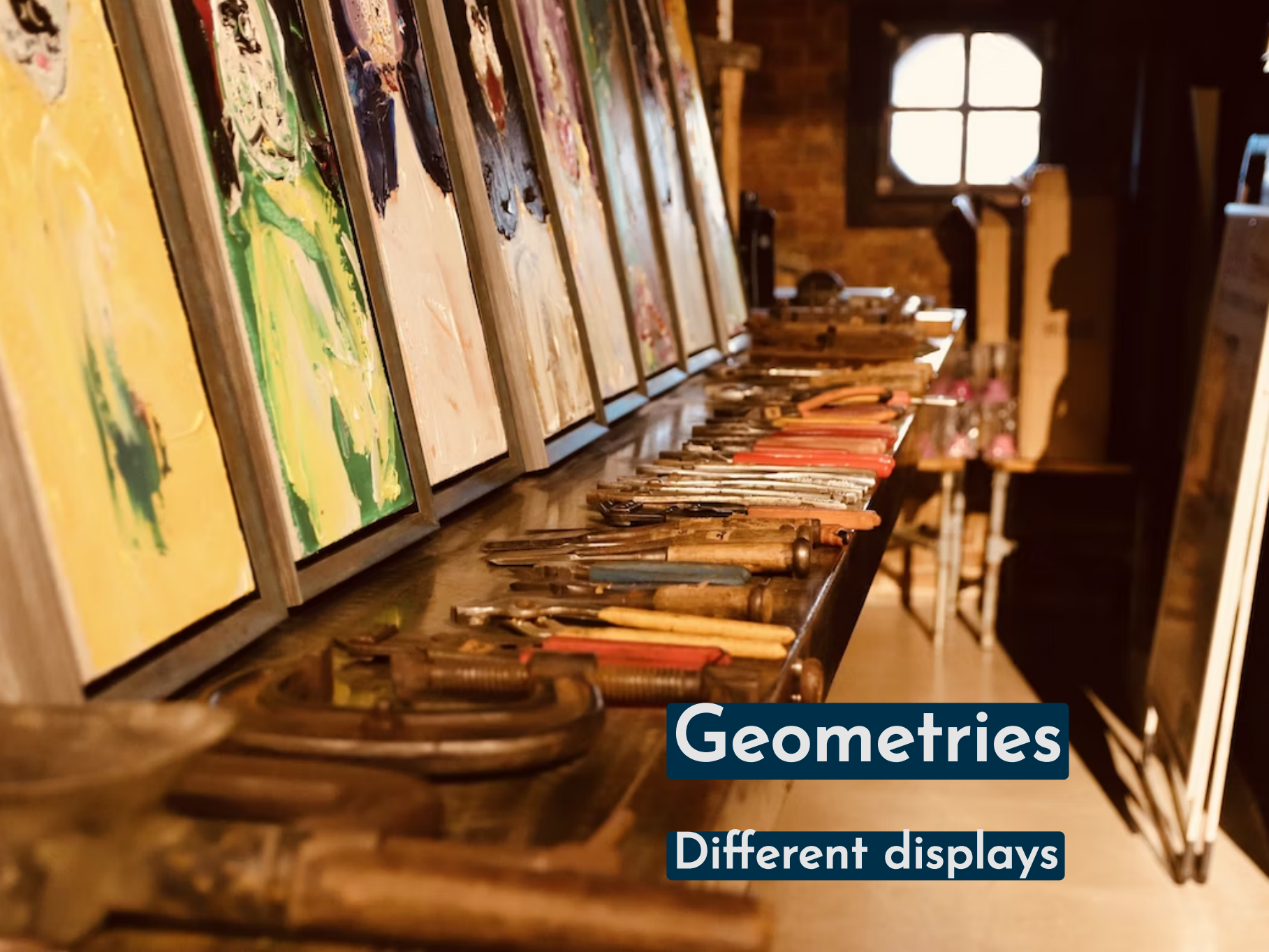


Until here:

- index
- theory

Next part:

- geometries
- maps
- aesthetics
- layers
- export
- contact

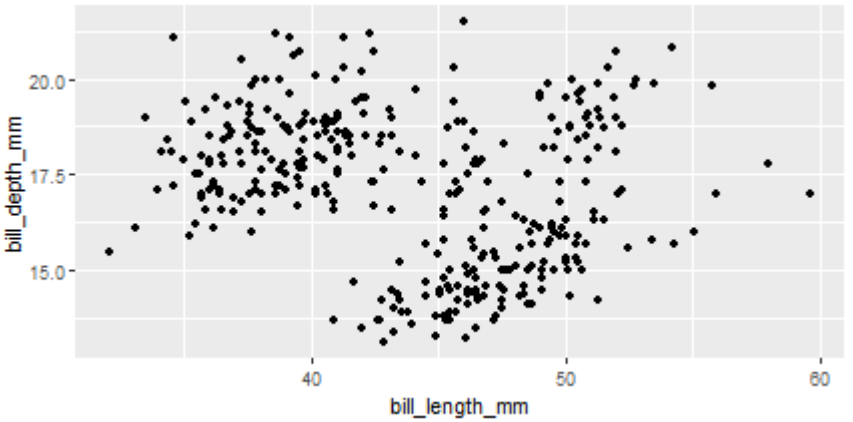

Geometries
Different displays

2.1. Scatterplots

The scatterplot is most useful for displaying the relationship between two continuous variables. Each dot represents an observation. Their position on the x (horizontal) and y (vertical) axis represents the values of the 2 variables.

The argument to create scatterplots is **geom_point**

```
ggplot(data=penguins,
       mapping=aes(x=bill_length_mm,y=bill_depth_mm))+
  geom_point()
```



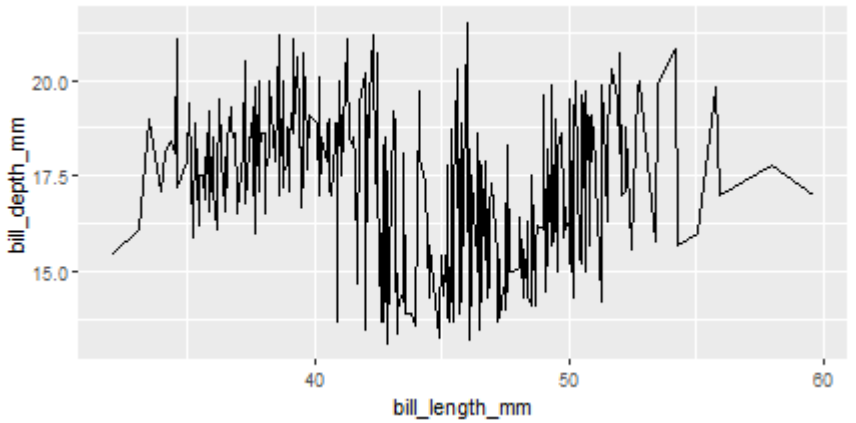
See more scatterplots

2.2. Line chart

A line chart or line graph displays the evolution of one or several numeric variables. Data points are connected by straight line segments.

The argument to create line charts is **geom_line()**

```
ggplot(data=penguins,
       mapping=aes(x=bill_length_mm,y=bill_depth_mm))+
  geom_line()
```



See more line chart

2.3. Barplot

A barplot is used to display the relationship between a numeric and a categorical variable.

The argument to create barplots is **geom_bar()**

```
ggplot(data=penguins,
       mapping=aes(x=species,y=body_mass_g))+
  geom_bar()
```

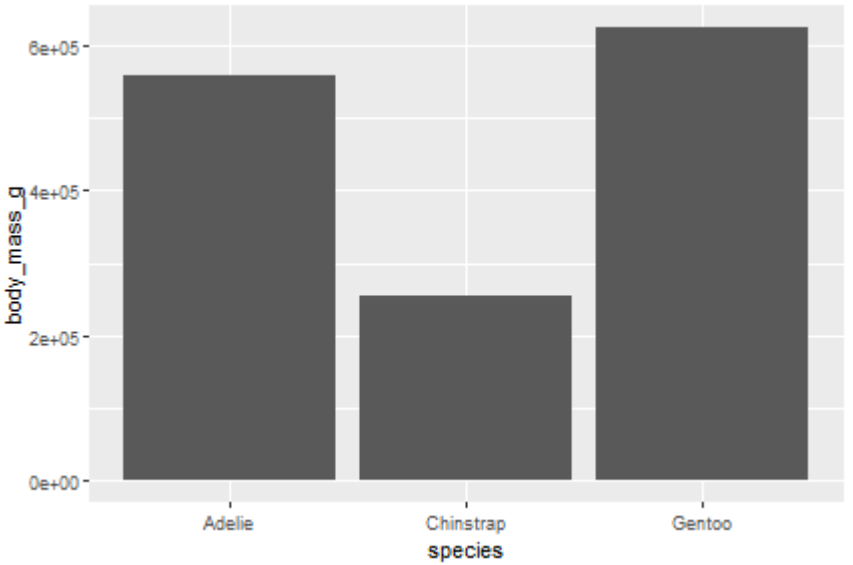
Error: stat_count() can only have an x or y aesthetic.

Different from the logic of the previous plots, we need to tell what do you want inside the bars by adding **stats=**.

2.3. Barplot

The argument **stats=** is to be included inside the parethesis of **geom_bar**

```
ggplot(data=penguins,
       mapping=aes(x=species,y=body_mass_g))+
  geom_bar(stat='identity')
```

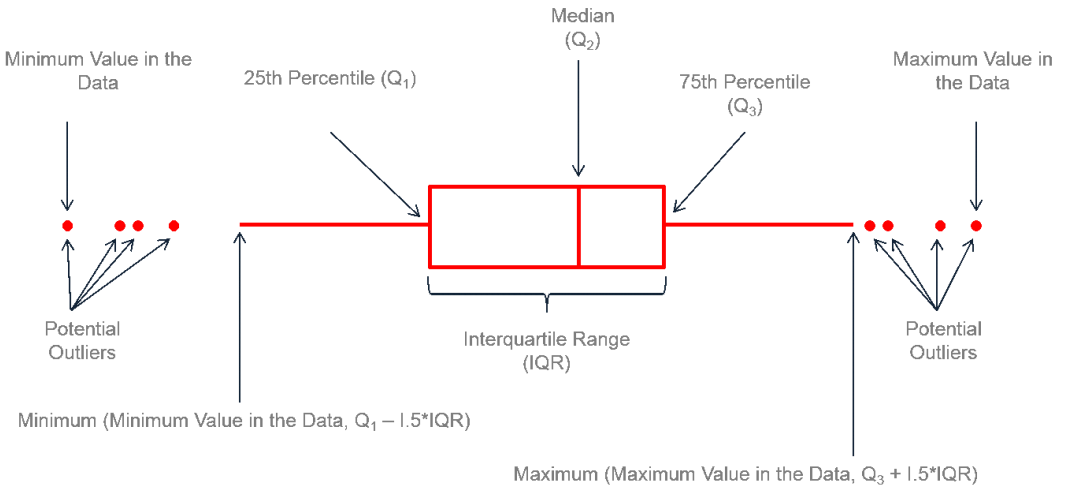


See more barplots

2.4. Boxplot

Boxplot is probably the most commonly used chart type to compare distribution of several groups.

It visualises five summary statistics (the median, two percentiles and two whiskers).

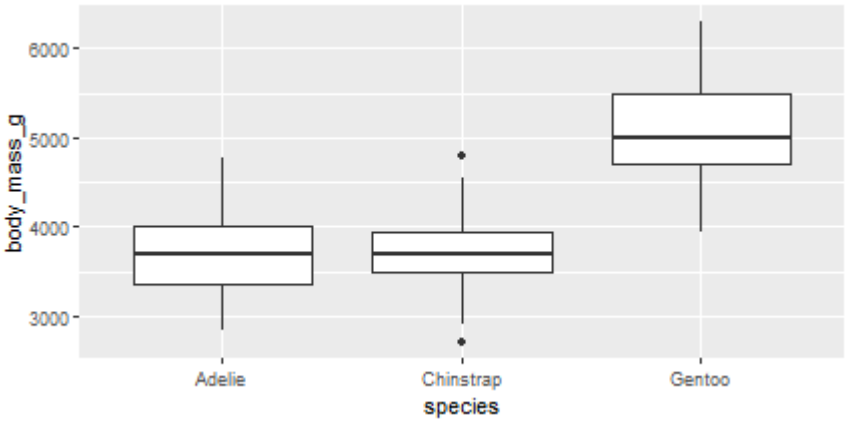


Source boxplot explanation

2.4. Boxplot

The argument to create boxplots is **geom_boxplot()**

```
ggplot(data=penguins,
       mapping=aes(x=species,y=body_mass_g))+
  geom_boxplot()
```



See more boxplots

2.5. Trajectory

For plotting moving objects, connecting points with a line is the most common visualization used.

For today exercises, install **sula**.

```
devtools::install_github("MiriamLL/sula")
```

The package **sula** contains tracking data from Masked boobies on Easter Island.

```
library(sula)
```

Load data from the package **sula** into the environment.

```
tracking_data<-sula::GPS_raw
```

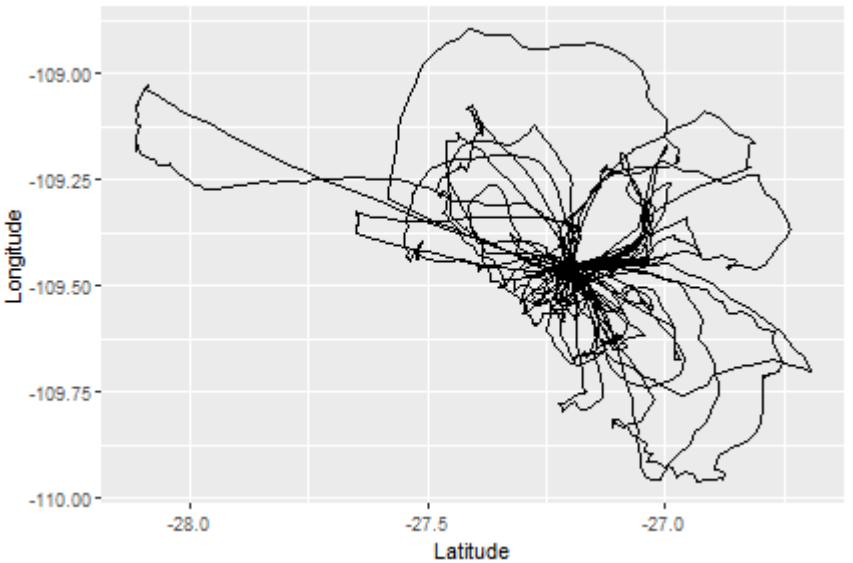
The data frame contains the columns **Latitude** and **Longitude**.

2.5. Trajectory

The argument to create trajectories is **geom_path()**.

By definition geom_path() connects the observations in the order in which they appear in the data.

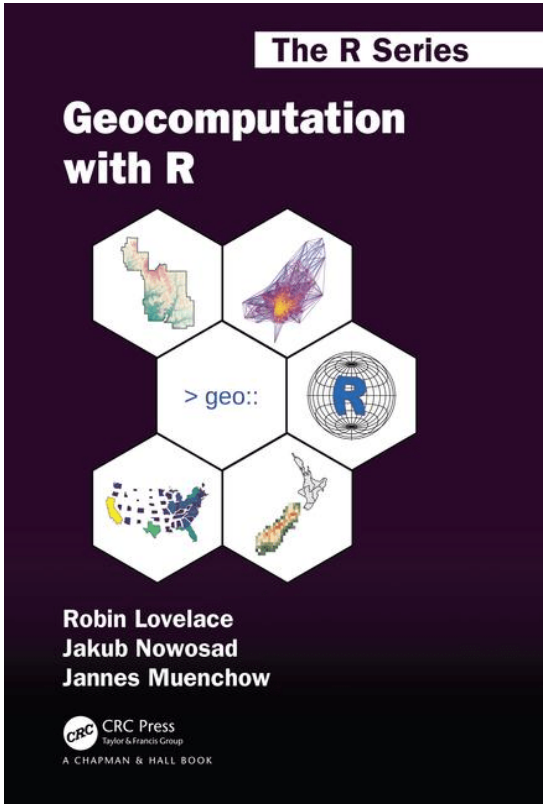
```
ggplot(data=tracking_data,
       mapping=aes(x=Latitude, y=Longitude)) +
  geom_path()
```



2.6. Maps

R is an great tool for geospatial data analysis.

The book geocomputation with R is a recommended guide to create maps in R (is open access and constantly updated).



2.6. Maps

For today exercises, install **GermanNorthSea**.

```
devtools::install_github("MiriamLL/GermanNorthSea")
```

The package **GermanNorthSea** compiles shapefiles from the North Sea.

```
library(GermanNorthSea)
```

To see the containing shapefiles go to my github.

To add shapefile to environment.

```
Europe<-German_land
```

Note that shapefiles can be an **sf** object. An sf object is different from a data frame, if we click on the object is not clear what it contains.

```
class(Europe)
```

```
## [1] "sf"         "data.frame"
```

2.6. Maps

The argument to create a map is **geom_sf()**

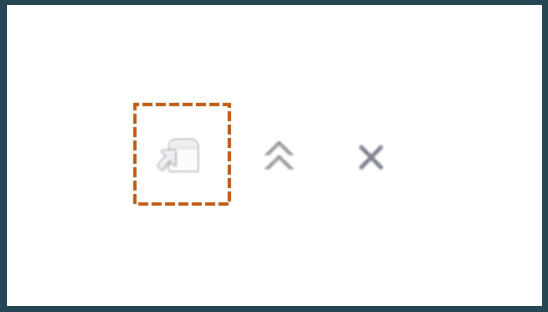
```
ggplot(data=Europe)+
  geom_sf()
```

Pause

Practice
Open the file 03ExercisesPlotting.Rmd

Run the code for plotting different geometries.

Note to see the plot in a separated window, search for the following icon.



Until here:

- index
- theory
- geometries

Next part:

- maps
- aesthetics
- layers
- export
- contact


Maps
Elements of a map

3.Maps

What makes a map a map?

A map is a symbolic representation of selected characteristics of a place, usually drawn on a flat surface.

Source Paula Scher

3.1. Shapefile

A shapefile is a format for geospatial vector data.

For the exercises download the shapefile from Europe.

Give the directory where you save your shapefile, either by calling the package **here**...

```
library(here)
My_directory<-here('Downloads')
```

... or manually by adding the directory where your shapefile is.

```
My_directory<-"..."
```

Check that the folder contains your shapefile.

```
list.files(My_directory)
```

```
My_shapefile<-paste0(My_directory,"/Europe.shp")
```

3.1. Shapefile

The package **sf** contains arguments to load shapefiles in R.

```
library(sf)
```

Visit the sf page.

3.1. Shapefile

The argument **st_read** loads the shapefile into the environment.

```
Europe_shp<-st_read(My_shapefile)
```

```
## Reading layer `Europe' from data source
##   `C:\Users\lerma\OneDrive\Documents\2MonTrack\2023\Teaching\R_intro\R_intr
##   using driver `ESRI Shapefile'
## Simple feature collection with 54 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -31.26575 ymin: 32.39748 xmax: 69.07032 ymax: 81.85737
## Geodetic CRS:  WGS 84
```

Note that it gives us the Geodetic CRS: WGS 84

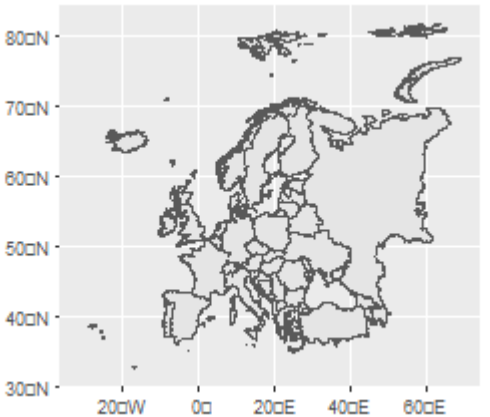
3.2. Create a basic map

Load the package ggplot2

```
library(ggplot2)
```

Plot your shapefile using **geom_sf**

```
ggplot(data = Europe_shp)+
  geom_sf()
```

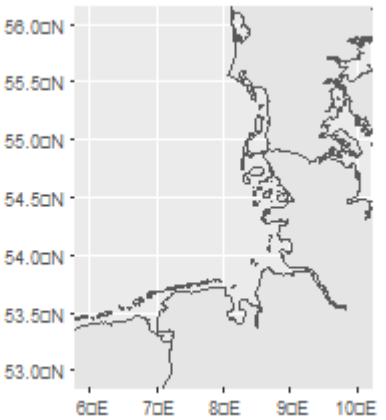


3.3. Add limits

Using the argument **coord_sf** to focus in the area of interest.

- Add x axis coordinates in **xlim**
- Add y axis coordinates in **ylim** .

```
ggplot(data = Europe_shp)+
  geom_sf()+
  coord_sf(xlim = c(6, 10),ylim = c(53, 56))
```



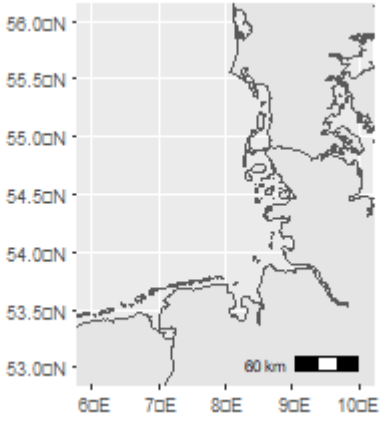
3.4. Add scale

To add more elements on a map, the package ggspatial contains scale and north arrow.

```
library(ggspatial)
```

The argument **annotation_scale** adds a scale.
By including **br** the scale is set to the **b**ottom **r**ight.

```
ggplot(data = Europe_shp)+
  geom_sf()+
  coord_sf(xlim = c(6, 10),ylim = c(53, 56))+
  annotation_scale(location = "br")
```

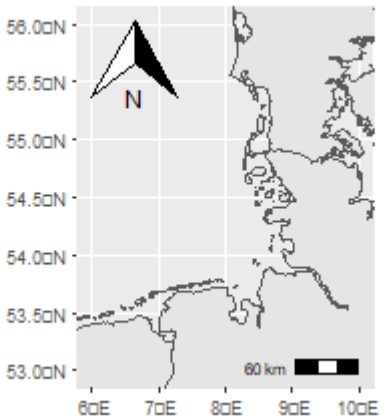


3.5. Add arrow

For including an arrow add the argument **annotation_north_arrow**.

Include **tl** to set the scale on the **top left**

```
ggplot(data = Europe_shp)+
  geom_sf()+
  coord_sf(xlim = c(6, 10),ylim = c(53, 56))+
    annotation_scale(location = "br")+
    annotation_north_arrow(location = "tl")
```

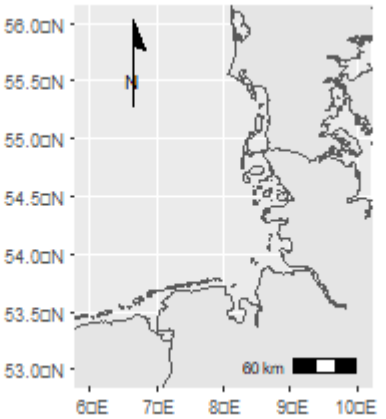


3.5. Add arrow

There are many options for **annotation_north_arrow**.

Including **style = north_arrow_minimal()** makes a more minimalistic type of arrow.

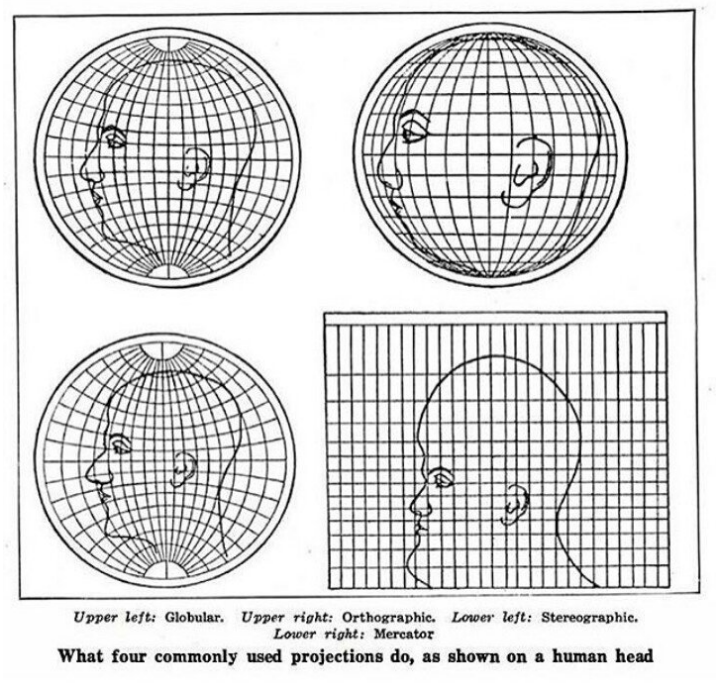
```
ggplot(data = Europe_shp)+
  geom_sf()+
  coord_sf(xlim = c(6, 10),ylim = c(53, 56))+
    annotation_scale(location = "br")+
    annotation_north_arrow(location = "tl",
                           style = north_arrow_minimal())
```



3.6. Important

A coordinate reference system (CRS) refers to the way in which spatial data that represent the earth's surface.

⚠️ It is important to understand the coordinate system that your data uses - particularly if you are working with different data stored in different coordinate systems.



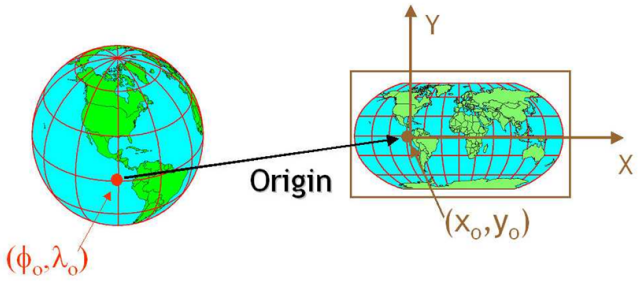
Upper left: Globular. *Upper right:* Orthographic. *Lower left:* Stereographic. *Lower right:* Mercator

What four commonly used projections do, as shown on a human head

3.6. CRS

All maps are distorted and all projections have some error to represent the earth perfectly.

The error is also depending on the origin.



Not using the correct projection might derive in calculation mistakes (over or under calculating distances for example).

I will not go into detaill in CRS but give you some tips to transform to the corresponding system.

3.6. CRS

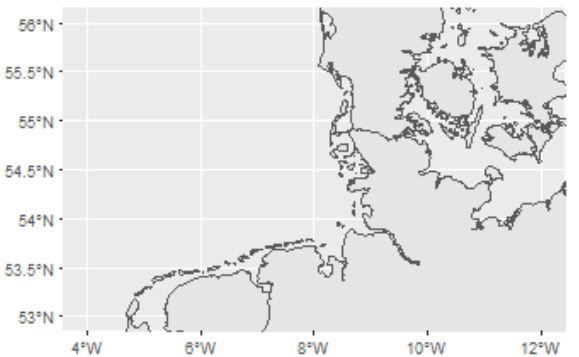
The argument **st_transform** from the package **sf** allows you to change the CRS of your shapefile.

First give the name of your shapefile and later the CRS in 4 digits numbers.

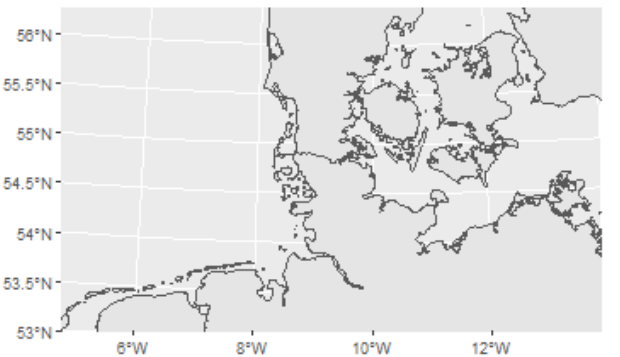
Note that there are many options available in R to transform your CRS.

```
Europe_3035<-sf::st_transform(Europe_shp, 3035)
```

CRS **4326**



CRS **3035**

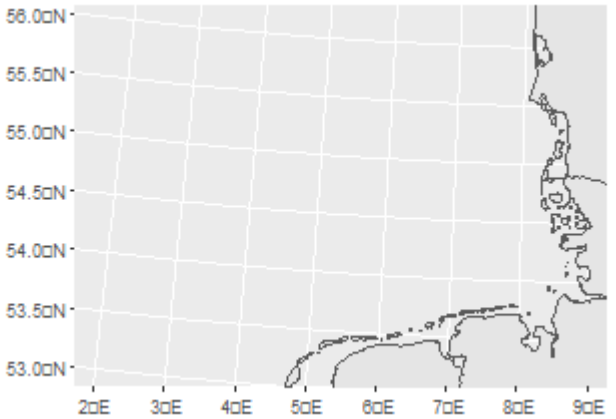


Do you notice the **differences?**

3.6. CRS

Consider that when using the map with the **CRS 3035**, the units of **xlim** and my **ylim** need to be changed accordingly.

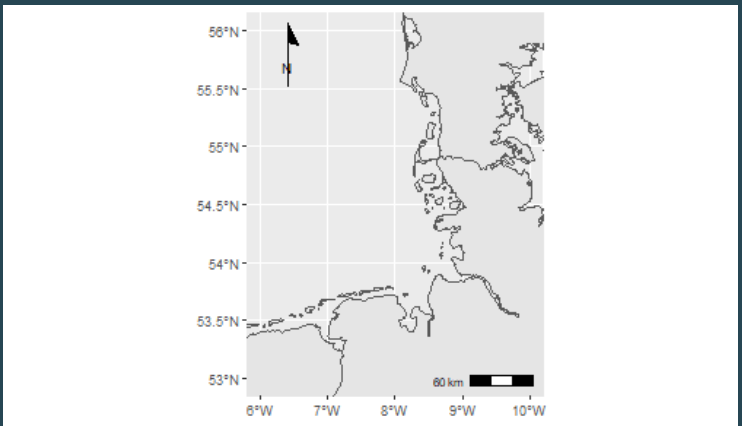
```
ggplot(data=Europe_3035)+
  geom_sf()+
  coord_sf(xlim = c(3790000,4250000), ylim = c(3350000,3680000))
```



Pause

Practice
Open the file 03ExercisesPlotting.Rmd

Create a map.



Until here:

- index
- theory
- geometries
- maps

Next part:

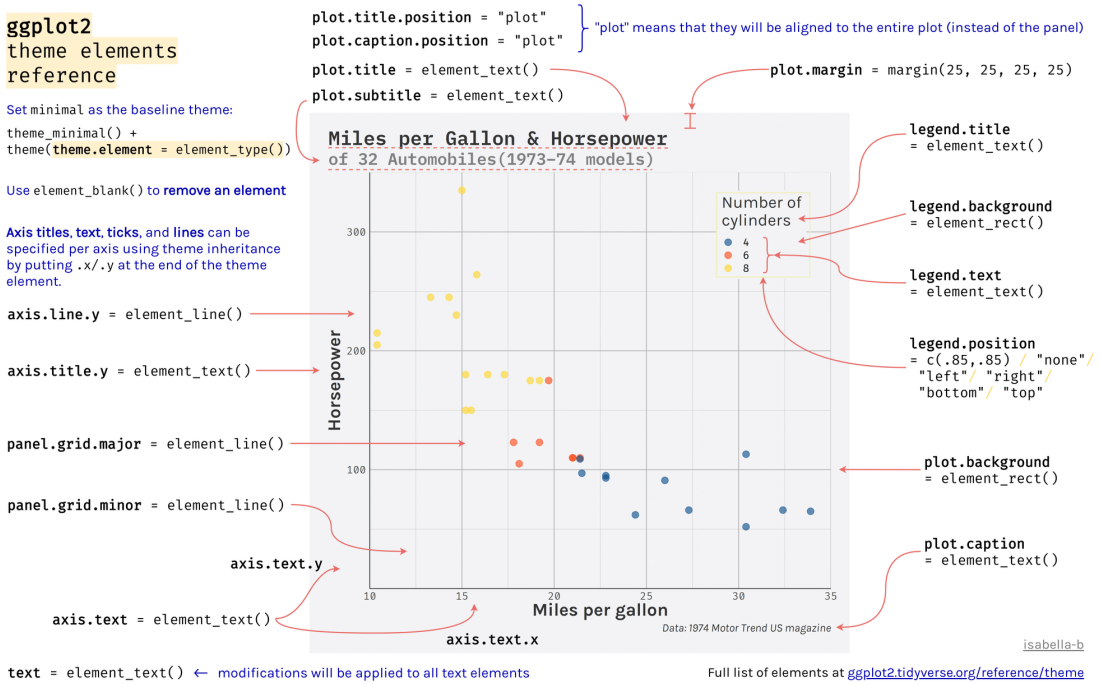
- aesthetics
- layers
- export
- contact


Aesthetics
Bring some colors

4. Aesthetics

Aesthetics are for controlling the overall appearance of graphs.

You can change almost every tiny detail to create the plot that you want.

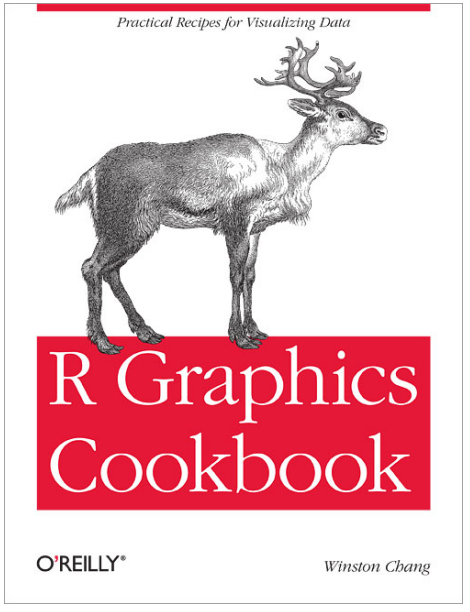


Source ggplot2 elements
Download pdf

4. Aesthetics

There are many elements, but I will not go into detail to most of them.

For going into more detail the book R Graphics cookbook is a great guide.



4.1. Color by name

To go beyond the default colors, the package ggplot2 allows you to change colors on the elements of your plot.

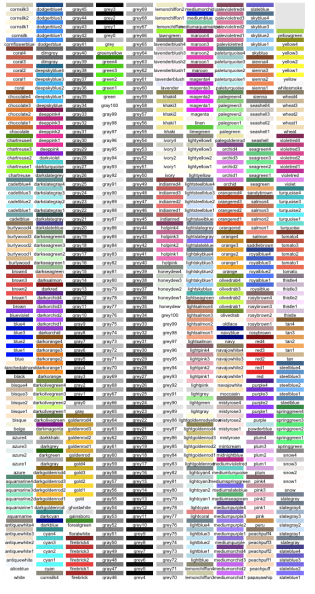
To select the colors there are several ways, I use by **name** or **hex codes**.

Using by **name**, the default names are used.

You can see a complete list of the 657 colors typing colors().

```
colors()
```

You can also see the color and the name by opening this link.



4.1. Color by hexcode

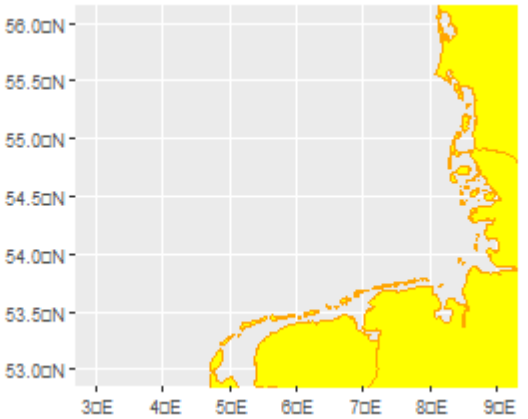
Colors can also be defined by their **hex code**. An hex code is the code of the color in 6 digits.

To find the hex code there are tons of webpages. I personally like coolors.

4.1. Fill color

For a map, we can change the color of our shapefile by adding the argument **colour** and **fill** inside the brackets of **geom_sf()**

```
ggplot(data = Europe_shp)+
  geom_sf(color = "orange",
          fill = "yellow")+
  coord_sf(xlim = c(3, 9),ylim = c(53, 56))
```

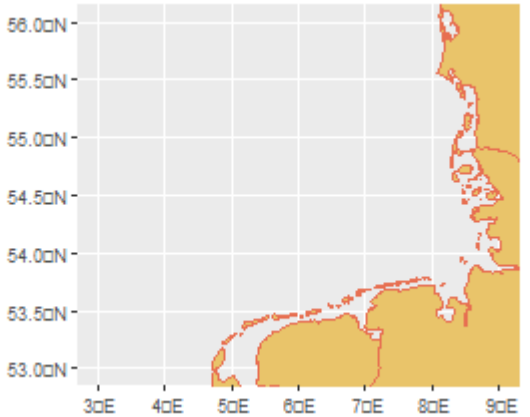


4.1. Fill color

To use the color by **hex code** you need to add a **#**.

In coolors go to explore>put your cursor on top of the color you like>click>paste on your code

```
ggplot(data = Europe_shp)+
    geom_sf(color = "#e76f51",
            fill = "#e9c46a")+
      coord_sf(xlim = c(3,9), ylim = c(53,56))
```



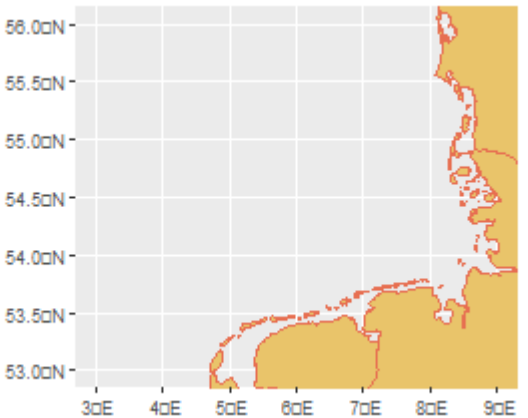
4.2. Object

To simplify the process, you can save your plot as an object in the environment.

```
My_plot<-ggplot(data = Europe_shp)+
    geom_sf(color = "#e76f51",
            fill = "#e9c46a")+
      coord_sf(xlim = c(3,9), ylim = c(53,56))
```

The map is now contained in an object.

```
My_plot
```



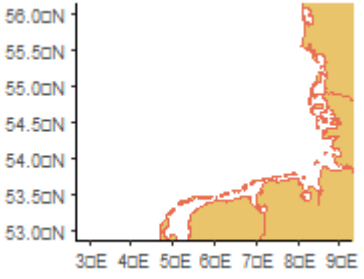
4.3. Theme

Themes are a powerful way to customize the non-data components of your plots: i.e. titles, labels, fonts, background, gridlines, and legends.

Themes can be used to give plots a consistent customized look.

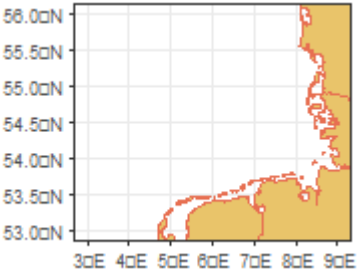
The theme can be changed using **theme_classic()**

```
My_plot+
  theme_classic()
```



The theme can be changed using **theme_bw()**

```
My_plot+
  theme_bw()
```

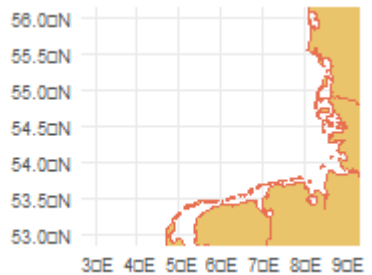


4.3. Theme

There are many different themes. It would depend on the type of graph and taste to choose.

The theme can be changed using **theme_minimal()**

```
My_plot+
  theme_minimal()
```



T he theme can be changed using **theme_void()**

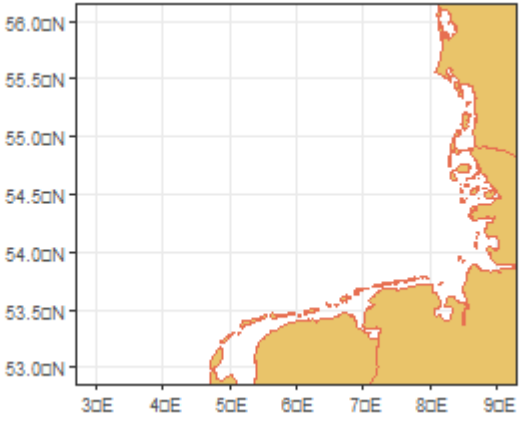
```
My_plot+
  theme_void()
```

4.3. Theme

For maps, in my opinion **theme_bw** is the one that makes most sense.

So lets added to our object using the **+** symbol.

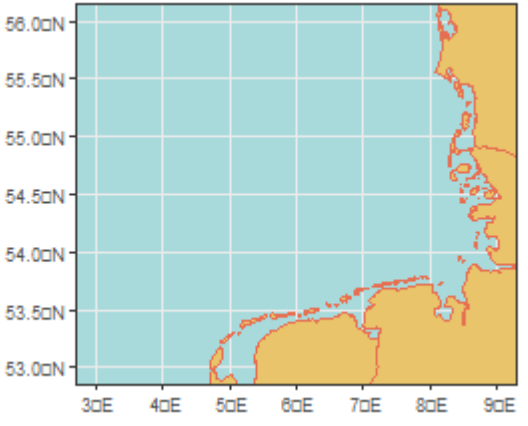
```
My_plot<-My_plot+
  theme_bw()
My_plot
```



4.4. Background colors

Adding the argument **theme** and then **panel.background** we can change the color of the background.

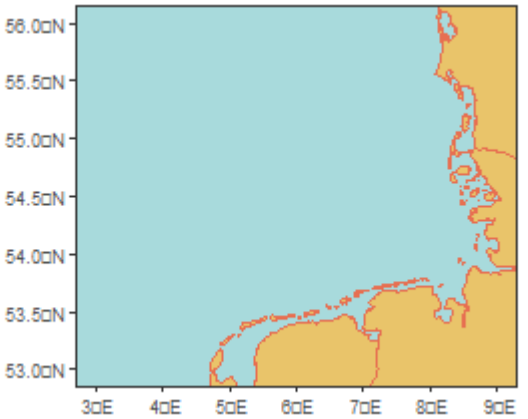
```
My_plot<-My_plot +
  theme(panel.background = element_rect(fill = '#a8dadc'))
My_plot
```



4.4. Background colors

In theme, the grid can be removed by adding **panel.grid.major** and **panel.grid.minor** and **element_blank** to make it blank.

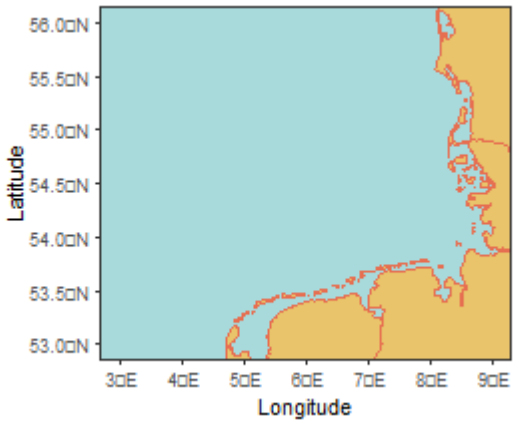
```
My_plot<-My_plot +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
My_plot
```



4.5. Axis legends

To change the legends on the axis, we can add the arguments **xlab** and **ylab**

```
My_plot<-My_plot +
  xlab('Longitude')+ylab('Latitude')
My_plot
```

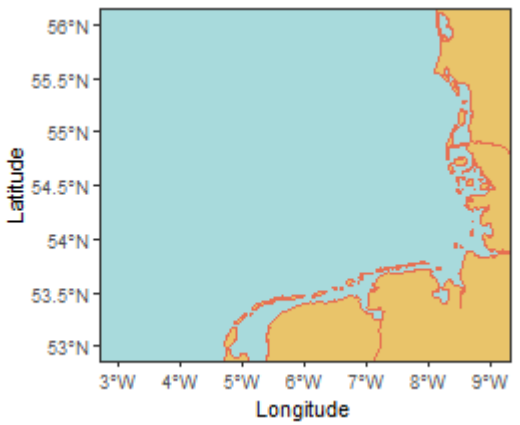


4.6. Add degrees

To add the degrees on the x and y axis legends, use a function on **scale_x_continuous** and **scale_y_continuous**.

This is specific from when plotting maps

```
My_plot<-My_plot +
    scale_x_continuous(labels = function(x) paste0(x, '\u00B0', "W")) +
    scale_y_continuous(labels = function(x) paste0(x, '\u00B0', "N"))
My_plot
```



4.7. Esquisse

There is an app to personalize all the parts, and gives you the code afterwards.

To install the add-inn go to Esquisse.

Pause

Practice 

Open the file 03ExercisesPlotting.Rmd

Create a map with your prefered colors.

Until here:

- index
- theory
- geometries
- maps
- aesthetics

Next part:

- layers
- export
- contact


Layers
Adding geometries

5. Geometries

To add observations on a map, we can add geometries.

To practice, the package **seamonas** contains data.

```
library(seamonas)
```

To add the data to your environment.

```
survey_4326<-survey_4326
```

The data frame contains the columns **latitude** and **longitude**.

```
head(survey_4326,5)
```

5.1. Add points

Because ggplot functions as layers, we can add geometries on top of each other.

5.1. Add points

For the exercise, lets create a base map.

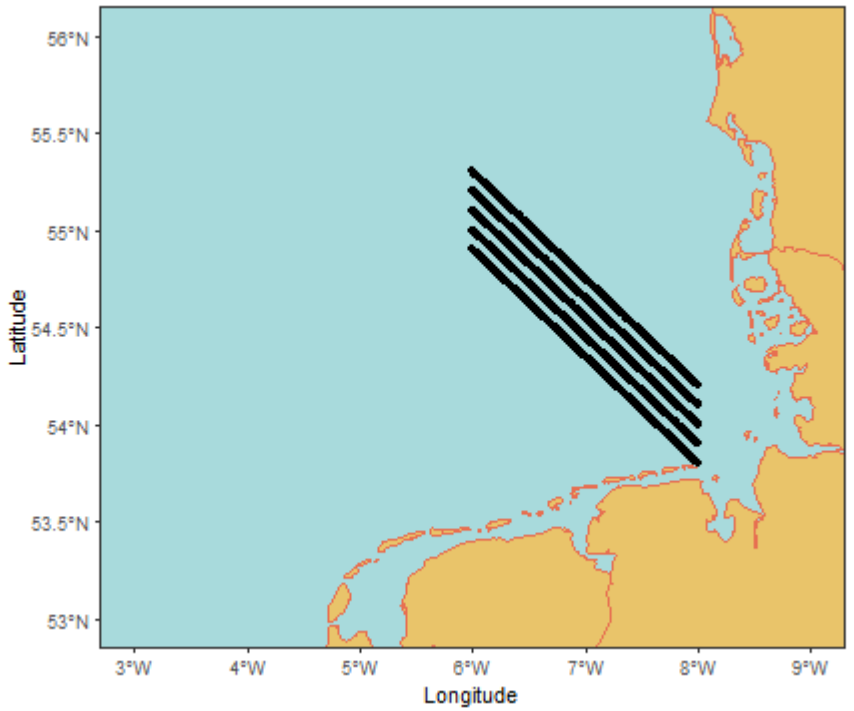
```
Base_map<-ggplot(data = Europe_shp)+
    geom_sf(color = "#e76f51",
            fill = "#e9c46a")+
  coord_sf(xlim = c(3,9), ylim = c(53,56))+
  theme_bw()+
  theme(panel.background = element_rect(fill = '#a8dadc'),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())+
  xlab('Longitude')+ylab('Latitude')+
  scale_x_continuous(labels = function(x) paste0(x, '\u00B0', "W"))+
  scale_y_continuous(labels = function(x) paste0(x, '\u00B0', "N"))
```

5.1. Add points

The order needs to be considered, the geometry to be on the top should be the last on the list.

Using the argument **geom_point** we can add points.

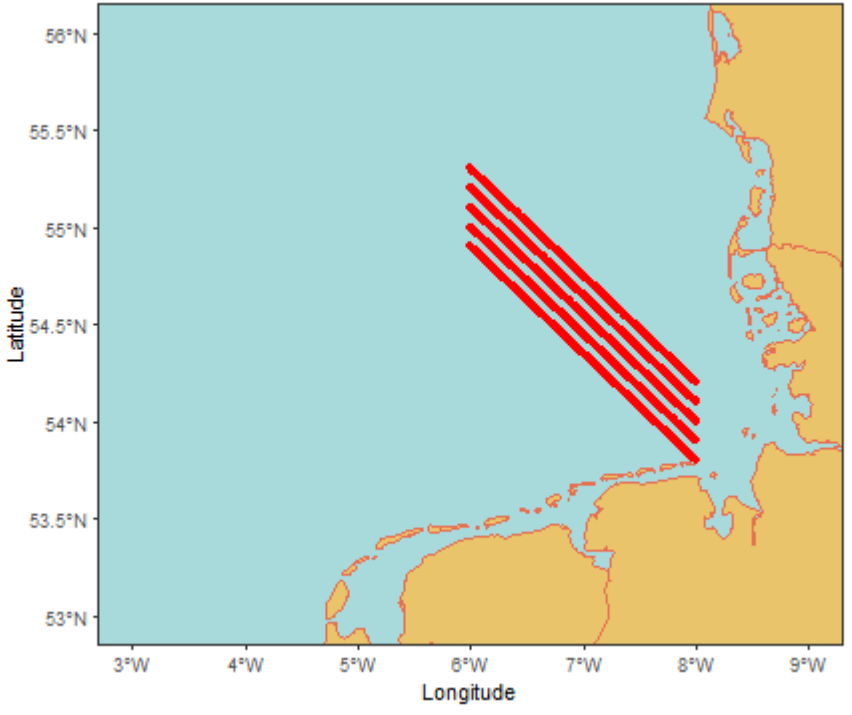
```
Base_map+
  geom_point(data=survey_4326,aes(x = longitude,y= latitude))
```



5.2. Change color

Adding the argument **color** inside the parethesis allows to change the color of the points.

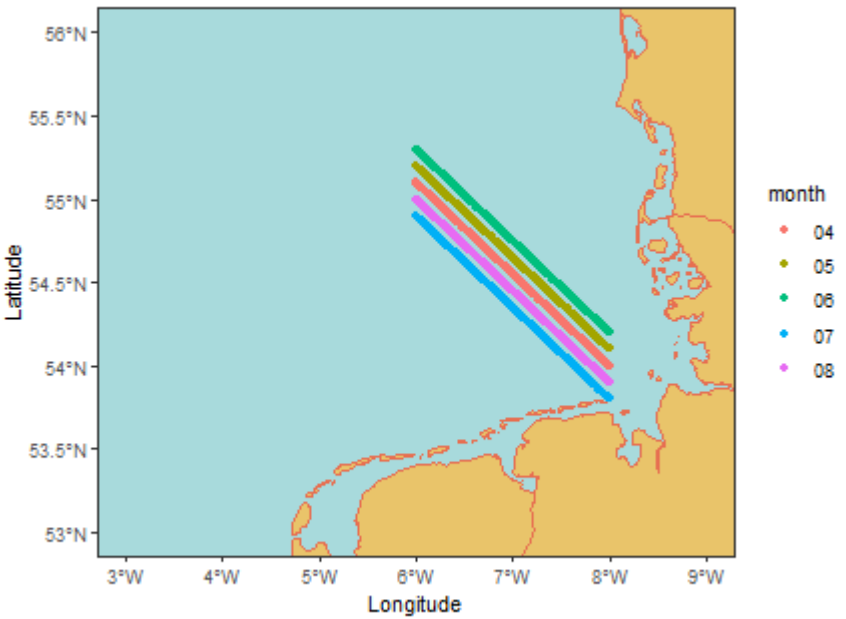
```
Base_map+
  geom_point(data=survey_4326,aes(x = longitude,y= latitude),
             color='red')
```



5.2. Change color

If you wish to color by a categorical variable, then add **colour =** within your aes brackets. This tells ggplot that this third variable will colour the points.

```
Base_map+
  geom_point(data=survey_4326,aes(x = longitude,y= latitude,
                                  color=month))
```

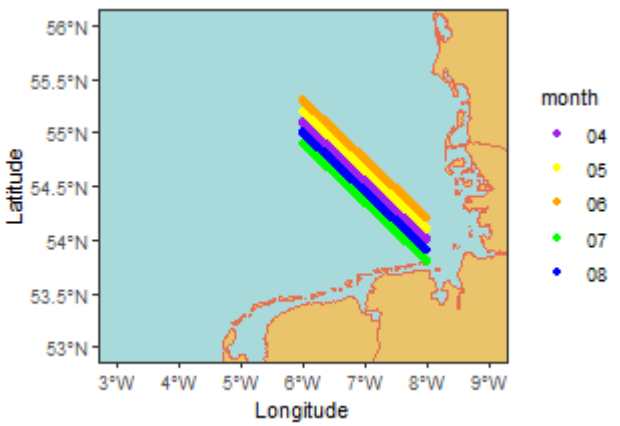


5.2. Select colors

The colors, when not declared, are selected by default.
To customize your colors you can add more arguments.
The argument **scale_colour_manual** allows you to customize those colors.
You can add colors by **name**.

Be careful and always add the same number of colors according to the category.

```
Base_map+
  geom_point(data=survey_4326,aes(x = longitude,y= latitude,
                                  color=month))+
  scale_colour_manual(values = c("purple", "yellow", "orange","green","b
```

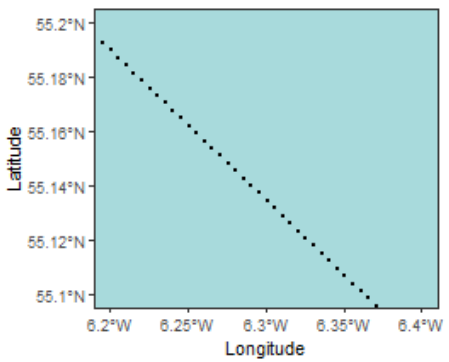


5.3. Change size

Adding the argument **size** inside **geom_point** let us change the size of the point.

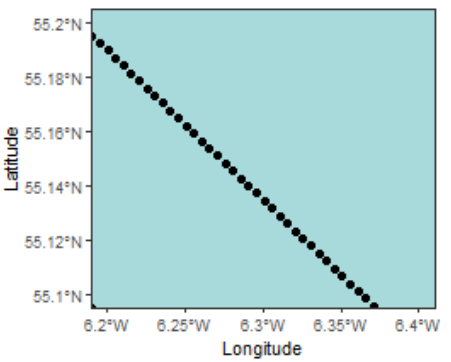
Smaller size

```
Base_map+
  coord_sf(xlim = c(6.2,6.4),
           ylim = c(55.1,55.2))+
  geom_point(data=survey_4326,
             aes(x = longitude,y=
             size=0.005)
```



Larger size

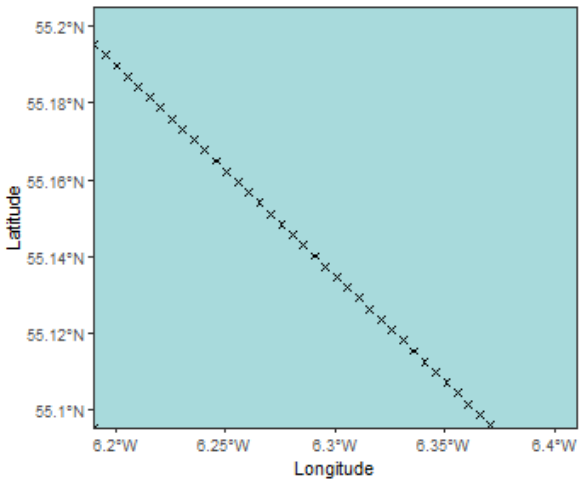
```
Base_map+
  coord_sf(xlim = c(6.2,6.4),
           ylim = c(55.1,55.2))+
  geom_point(data=survey_4326,
             aes(x = longitude,y=
             size=2)
```



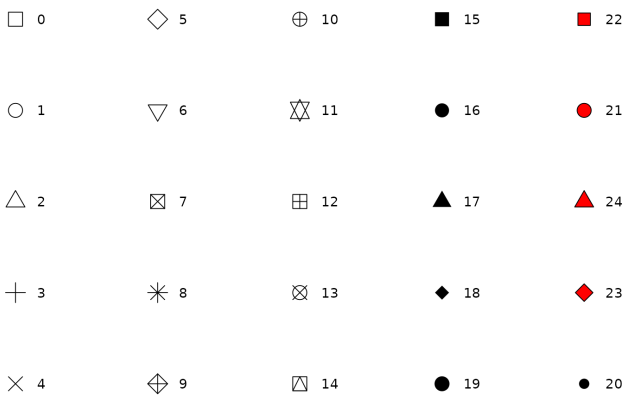
5.4. Change shape

Adding the argument **shape** inside **geom_point** let us change the shape of the point.

```
Base_map+
  coord_sf(xlim = c(6.2,6.4),
           ylim = c(55.1,55.2))+
  geom_point(data=survey_4326,
             aes(x = longitude,
                 y= latitude),
             shape=4)
```



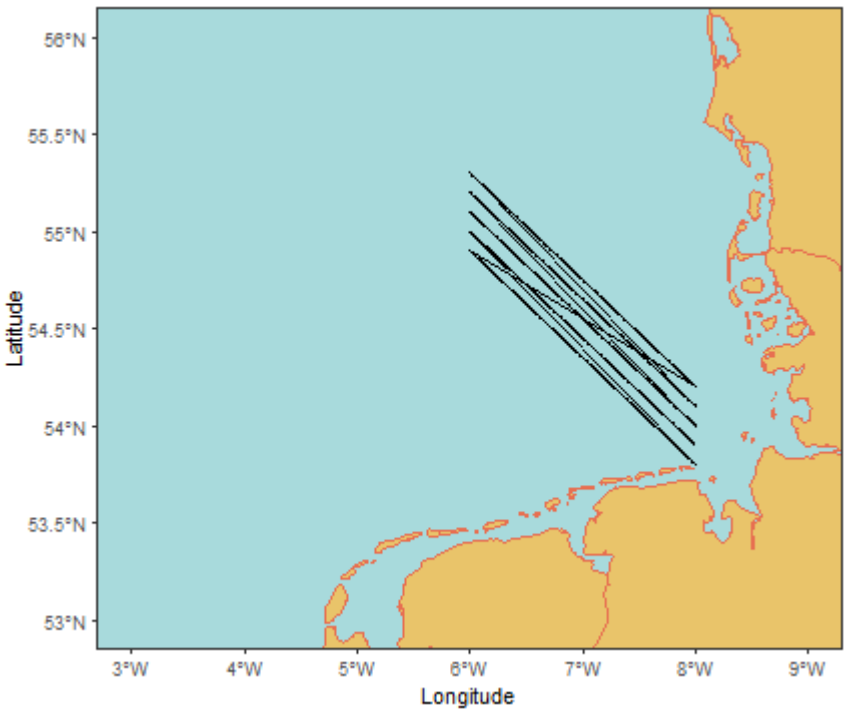
Other options are:



5.5. Add path

The argument **geom_path** can be used for trajectories.

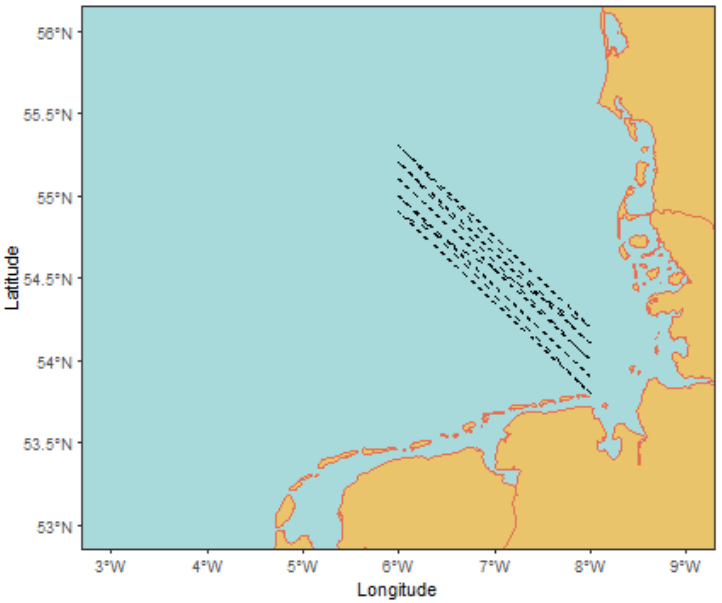
```
Base_map+
  geom_path(data=survey_4326,aes(x = longitude,y= latitude))
```



5.4. Add path

To change line type add the argument **linetype**.

```
Base_map+
  geom_path(data=survey_4326,
            aes(x = longitude,y=
            linetype = 'dashed')
```



Other line options are:

solid

dashed

dotted

dotdash

longdash

twodash

Pause

Practice 

Open the file 03ExercisesPlotting.Rmd

Create your own map with points or a path.

Until here:

- index
- theory
- geometries
- maps
- aesthetics
- layers

Next part:

- export
- contact

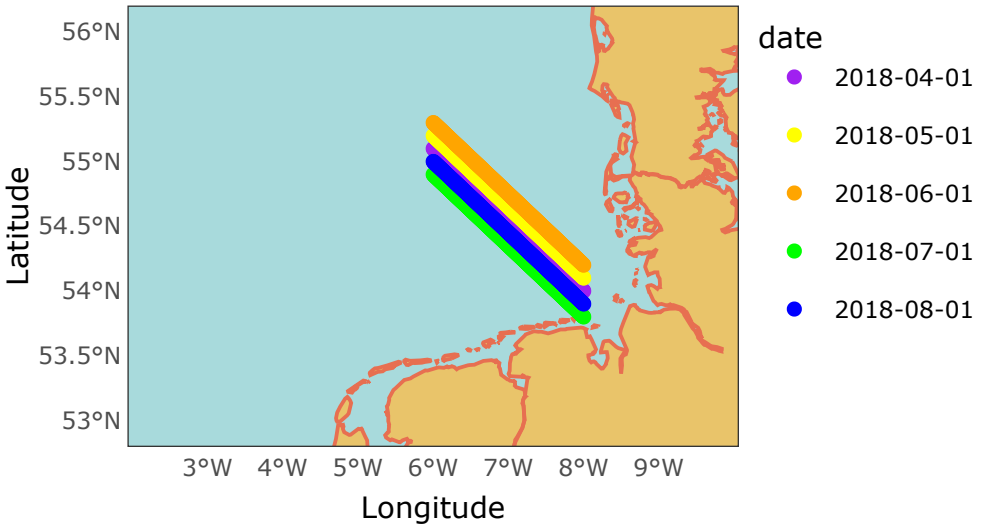

Export
Beyond static maps

6.1. Interactive

Graphs can be interactive.

-> The package plotly allows creating interactive maps.

-> The package tmap is also a recommende alternative.



Consider the elements inside the brackets of **aes** are the ones visible when putting the cursor on top.

6.1. Interactive

To install the package plotly.

```
install.packages('plotly')
```

To call the package.

```
library(plotly)
```

To use, include the argument **ggplotly** and add your ggplot inside the brackets.

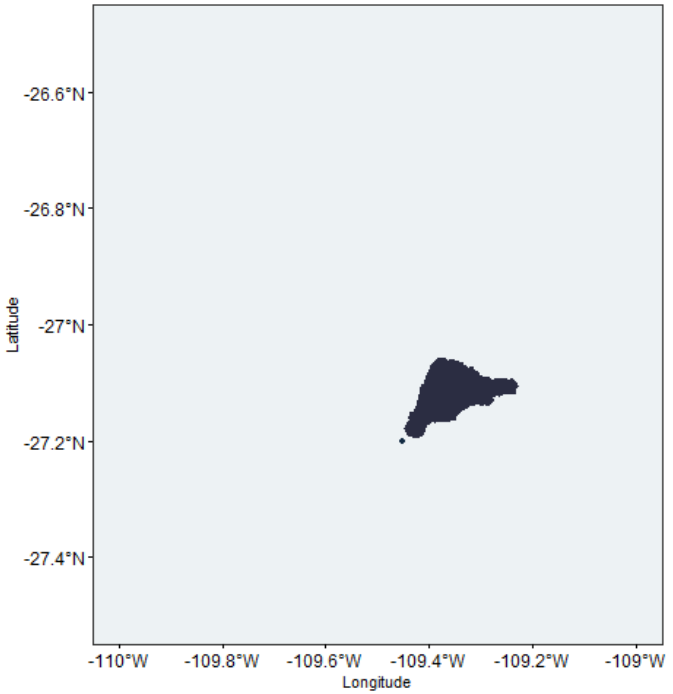
```
ggplotly(Base_map+
  geom_point(data=survey_4326,aes(x = longitude,y= latitude,
                                  color=date))+
  scale_colour_manual(values = c("purple", "yellow", "orange","green","b
)
```

6.2. Animate

Animated plots can also be created in R.

However, doing this during a R session would be very time consuming.

To recreate the animation, you can see my blog.



6.3. export

To export a plot as a figure there is **ggsave**.

Specify the folder where you want the plot to be exported to.

```
ResultsFolder<-here('Downloads')
```

Using the function **ggsave** it would be exported.

You can change the extension (.png, .jpg, etc), the width, length, units and dpi.

```
ggsave(My_plot,
       filename = paste0(ResultsFolder,"/My_plot.png"),
       width = 24,
       height = 24,
       units = "in",
       dpi = 500)
```

6.4. word

Another way to export maps is to use a Rmd file like the one on the link.

For exporting to word, consider:

- Not including the code, write **echo=FALSE** in your code chunk.
- Not including messages, write **message=FALSE** in your code chunk.
- Not including warnings, write **warning=FALSE** in your code chunk.

```
{r,echo=FALSE, warning=FALSE, message=FALSE}
```

To adjust the size of the maps you can specify the fig.height and fig.width in the code chunk

```
{r,fig.height=10, fig.width=10}
```

6.5. html

The best way to export interactive maps is to use Rmd to html like the one on the link.

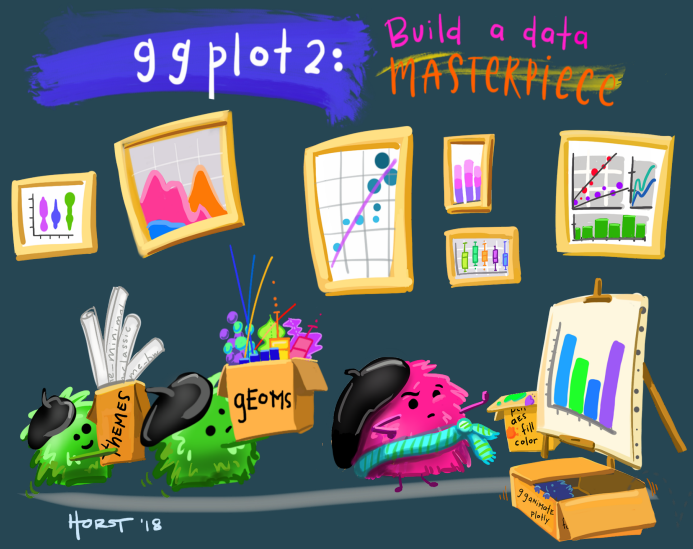
Just remember to change the output on the YAML to **html**.

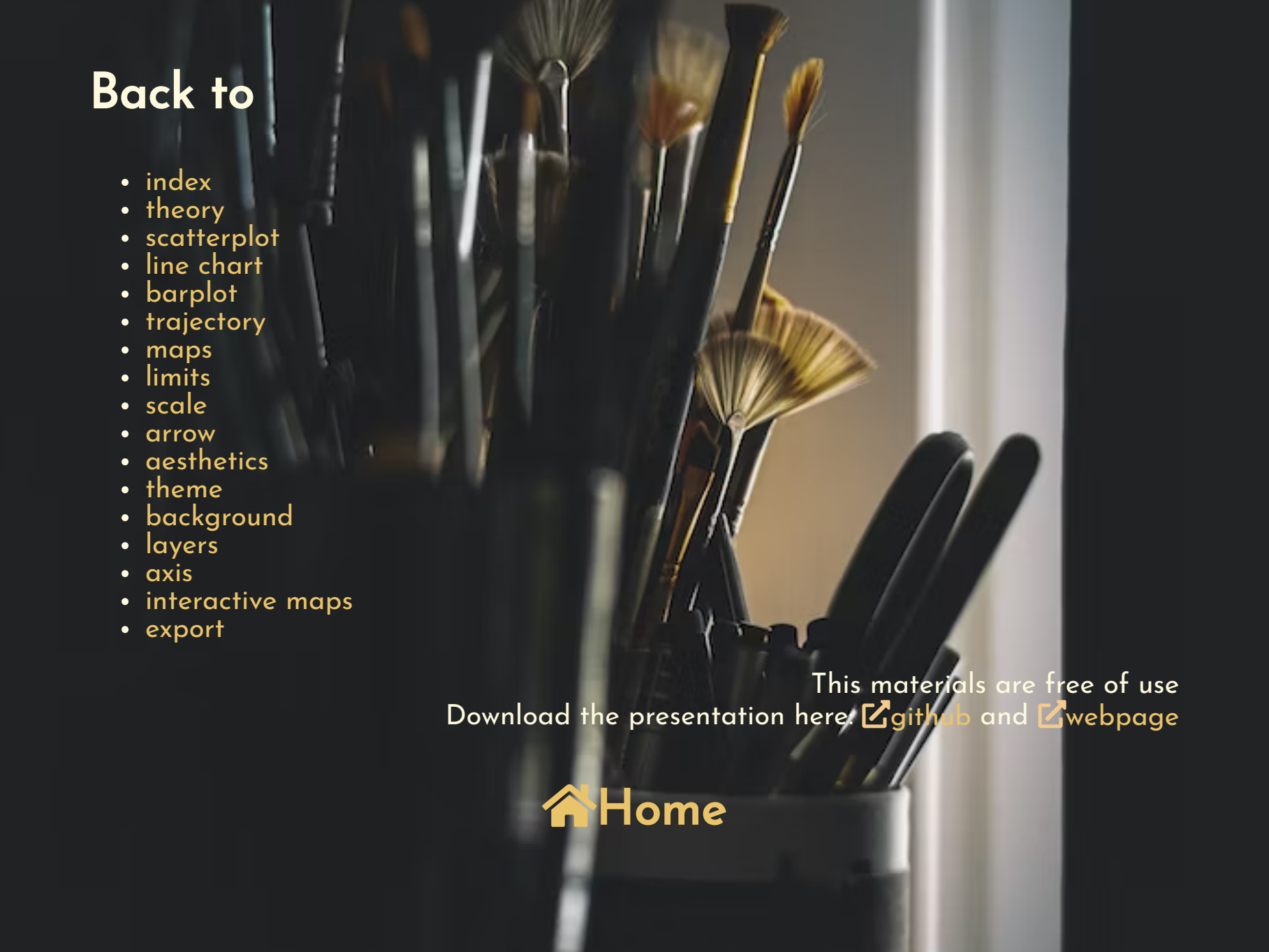
```
output: html_document
```

Pause

Practice ✎

- Export a plot in jpg.
- Export the plots in a word document.
- Export an interactive plot in html.





Back to

- index
- theory
- scatterplot
- line chart
- barplot
- trajectory
- maps
- limits
- scale
- arrow
- aesthetics
- theme
- background
- layers
- axis
- interactive maps
- export

This materials are free of use
Download the presentation here: github and webpage

Home