Data visualisation workshop

Introduction to ggplot

Roozbeh Valavi, Iliana Medina & David Wilkinson

26/04/2019

Contents

Creating high quality graphics

Once you've completed your data analysis you're going to need to summarise it in some really nice figures for publication and/or presentations. This is where `R` can really shine in comparison to something like Excel.

While you can create plots through various ways, including base `R`, the most popular method of producing fancy figures is with the `ggplot2` package. First things first, if you haven't done so yet, we need to install the `ggplot2` package:

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

And load the package:

```
library(ggplot2)
```

Importing data

The dataset we are going to use is public and available on Plos Biology (Bestion et al, 2015). It is a simple dataset, where every row represents a lizard in one of the 18 populations. There is information on the temperature treatment (Present or Warm), the weight of the lizard at birth and many other variables. We are interested specifically in in the total annual growth (TotLength_growth_annual) and the temperature treatment (Temperature_Treatment_Intro).

```
sample_data <- read.csv('~/Dropbox/Data Viz/data_1.csv')

head(sample_data)
```

```
##   Temperature_Treatment_Intro W_birth Sex W_birth.1 SVL_sept
## 1                        Warm   0.178   M     0.178       NA
## 2                        Warm   0.225   F     0.225       43
## 3                        Warm   0.217   F     0.217       41
## 4                        Warm   0.198   M     0.198       NA
## 5                        Warm   0.184   F     0.184       41
## 6                        Warm   0.181   F     0.181       NA
##   Summer_Survival Annual_Survival TotLength_growth_annual W_growth_annual
## 1               0               0                      NA              NA
```

```
## 2               1               0                     NA              NA
## 3               1               0                     NA              NA
## 4               0               0                     NA              NA
## 5               1               0                     NA              NA
## 6               0               0                     NA              NA
##   W_may W_birth.2
## 1    NA     0.178
## 2    NA     0.225
## 3    NA     0.217
## 4    NA     0.198
## 5    NA     0.184
## 6    NA     0.181
```

Grammar of Graphics

`ggplot2` is built on the grammar of graphics concept: the idea that any plot can be expressed from the same set of components:

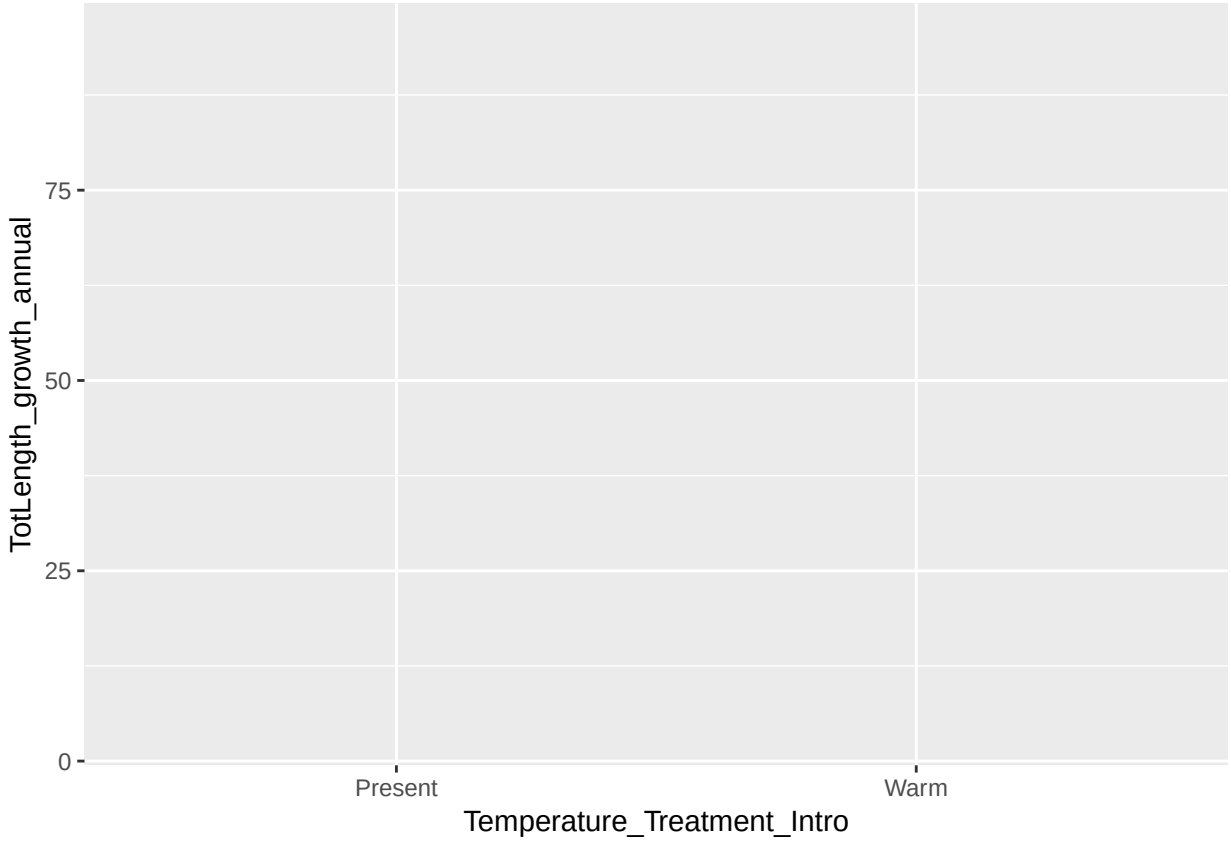
- A **data** set
- A **coordinate** system
- A set of **geoms** - the visual representation of data points

The key to understanding ggplot2 is thinking about a figure in layers.

To start with we need to create the base layer for our plot. This is done with the `ggplot()` function, and it is here we define are **data** and **coordinate system** components. We set our **data** component using the data argument, and then use the aesthetic function `aes()` as a second argument to define our **coordinate system**. The `aes()` function can potentially be used multiple times when constructing our figure, so it is important to know that anything defined inside the base `ggplot()` function is a *global option* inherited by all subsequent layers.

Time for an example:

```
ggplot(data = sample_data, aes(x = Temperature_Treatment_Intro, y = TotLength_growth_annual))
```

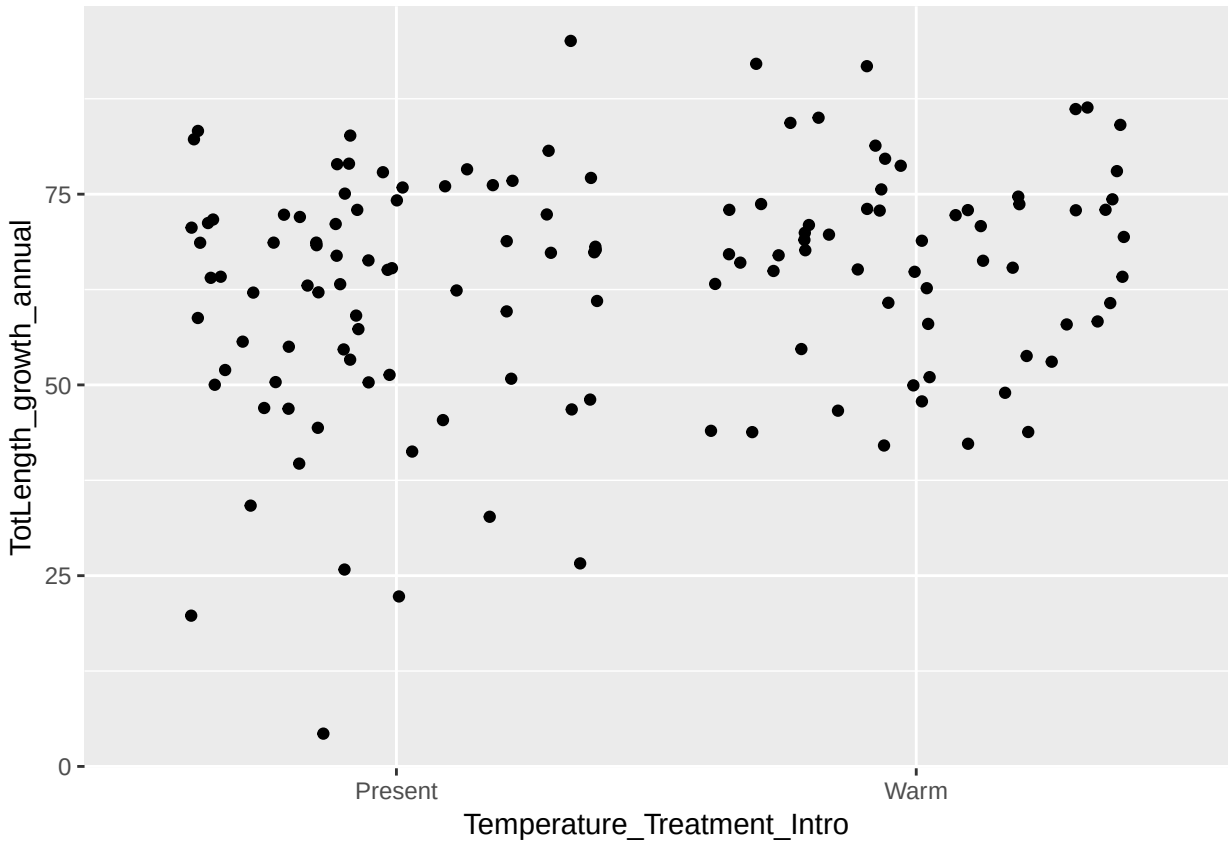


Here we have said that `sample_data` is our **data** component, and to use `Temperature_Treatment_Intro` on the x-axis and `TotLength_growth_annual` on the y-axis for our **coordinate system**.

Now let's build onto this base layer by adding **geoms** to represent our data! `ggplot2` employs a really nice syntax where you can add subsequent layers using the `+` operator. This lets `R` knows that the plot isn't finished yet and to add the new layer on top. In this case, we want to represent our data as *jittered points* so we use the `geom_jitter()` function. This is essentially a scatterplot for discrete values. It adds a small amount of *random variation* to the location of each point, and is a useful way of handling overplotting caused by discreteness in smaller datasets.

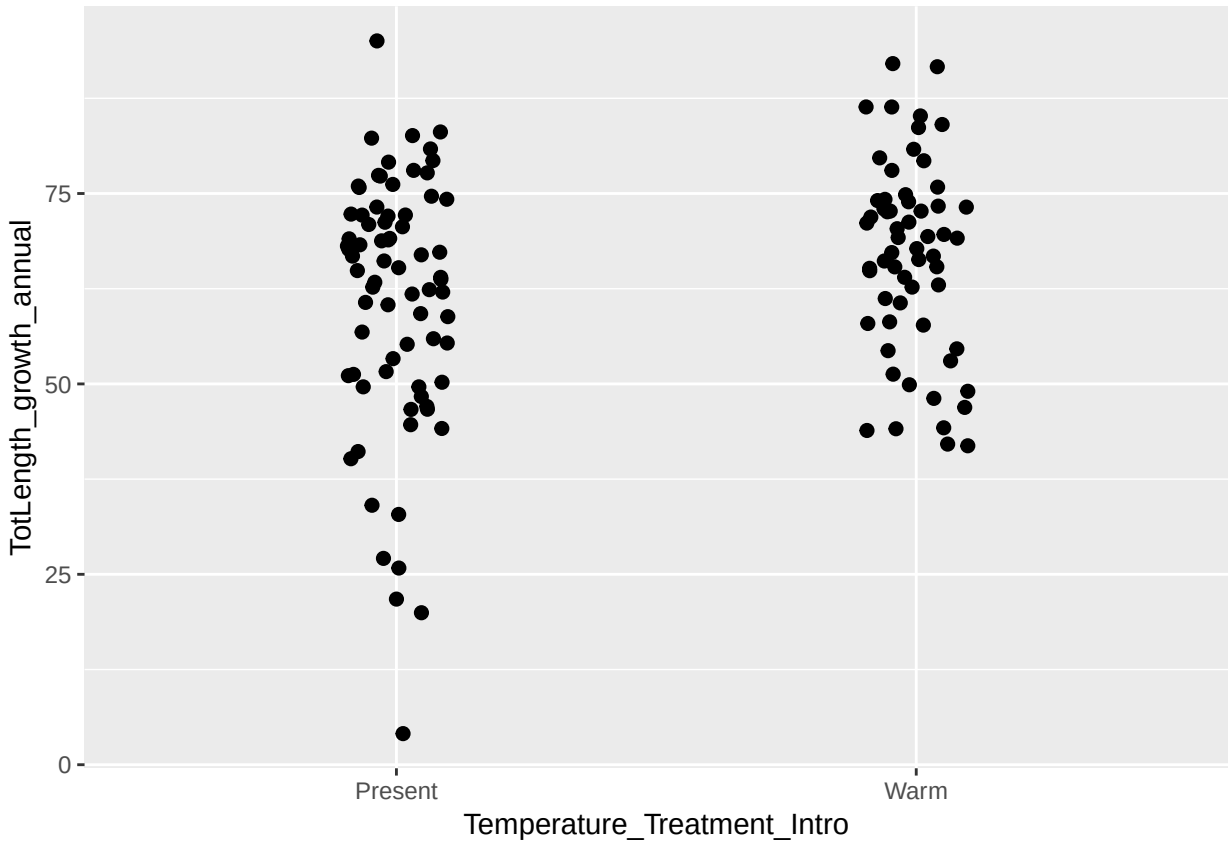
```
ggplot(data = sample_data, aes(x = Temperature_Treatment_Intro,
                               y = TotLength_growth_annual)) +
  geom_jitter()
```

```
## Warning: Removed 469 rows containing missing values (geom_point).
```



You can see the points are *overly scattered*. But, every `geom` has several parameters that control the properties of the plotted geometry. Let's use `geom_jitter()` parameters to narrow the width of the points and change their size.

```
ggplot(data = sample_data, aes(x = Temperature_Treatment_Intro,
                               y = TotLength_growth_annual)) +
  geom_jitter(size = 2, width = 0.1)
```



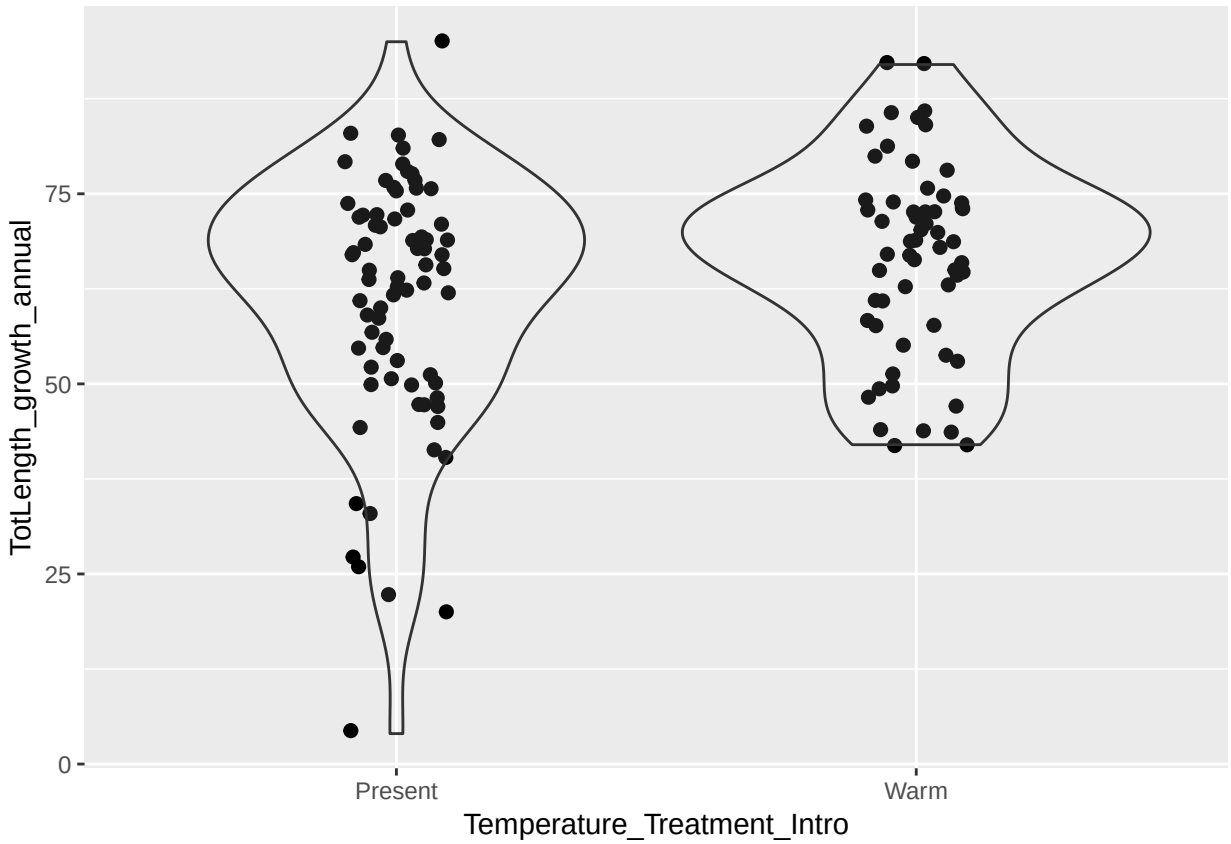
Challenge 01

Try to recreate the same graph.

- Use `geom_point()`and see the difference between `geom_points()` and `geom_jitter()`.

We can add more `geoms` on the previous plot by `+` operator.

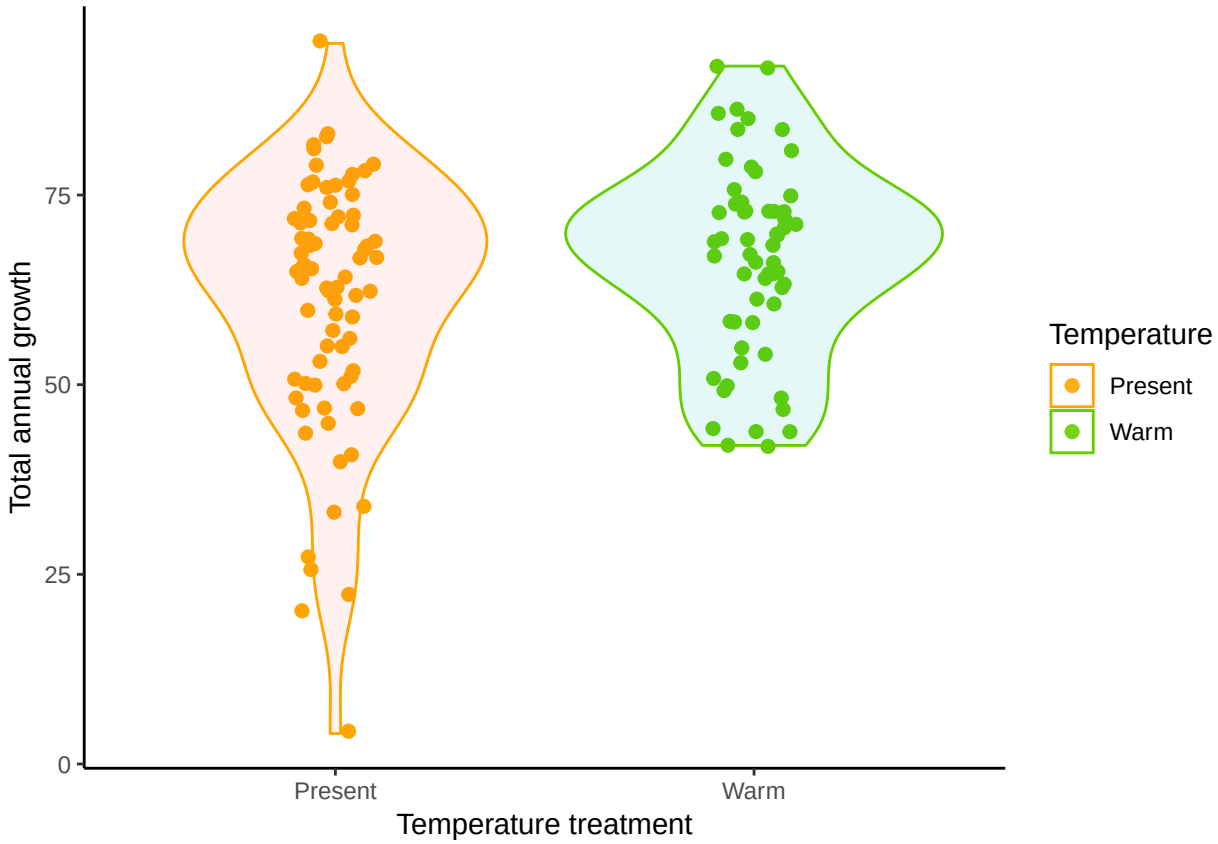
```
ggplot(data = sample_data, aes(x = Temperature_Treatment_Intro,
                               y = TotLength_growth_annual)) +
  geom_jitter(size = 2,  width = 0.1) +
  geom_violin(alpha = 0.1) ### alpha is the parameter to indicate how transparent the violins will be
```



Now, we can change the colour of the points and modify the axes titles.

```
plot01 <- ggplot(data = sample_data, aes(x = Temperature_Treatment_Intro,
                                         y = TotLength_growth_annual,
                                         color = Temperature_Treatment_Intro, # edge colour based on th
                                         fill = Temperature_Treatment_Intro)) + # fill colour based on
  geom_jitter(size = 2,  width = 0.1) +
  geom_violin(alpha = 0.1) + #### until here is the same code as before, just points and violin plot
  scale_color_manual(values = c(Present = "orange", Warm = "chartreuse3")) +  # specify the colours we
  labs(color = "Temperature", x = "Temperature treatment", y = "Total annual growth") + #define label n
  theme_classic() + #theme_classic means that the background is white
  guides(fill = FALSE) ## removes the redundancy in the legend

plot01
```



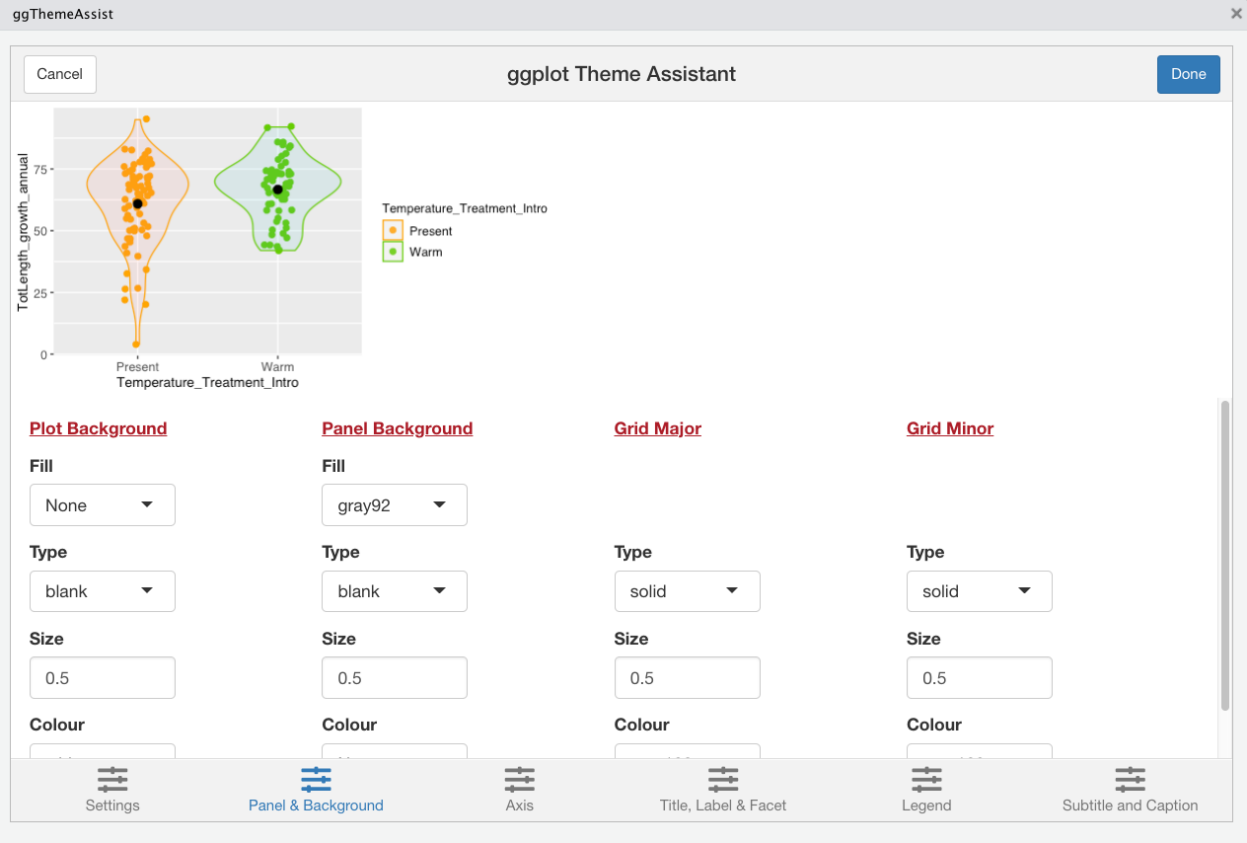
Change the ggplot theme interactively

Change many parameters in ggplot theme to make a pretty plot can be very time consuming and boring. To make this quick and easy, `ggThemeAssist` package has created an *RStudio Addin* for modifying `ggplot` theme *interactively* called **ggplot Theme Assistant**. You should install the package first.

```
# etheir install it from CRAN
install.packages("ggThemeAssist")
# or github
devtools::install_github("calligross/ggthemeassist")
library(ggThemeAssist)
```

You can access RStudio Addins on the top right of your RStudio as in the following picture.

To use this Addin, you need to highlight the code you want to modify and select the Addin.



You can easily change different parameters of panel, axis, legend and the title of the plot within different panels. After finishing the modification, you should press `Done` button on the upper right corner. This will change the highlighted code and add the relevant code to make the changes.

Challenge 02

Use the ggplot theme assistant to change the followings:

- Change the background colour of the plot
- Change the text font of all components
- Put the legend on the top of the plot vertically

Adding more features

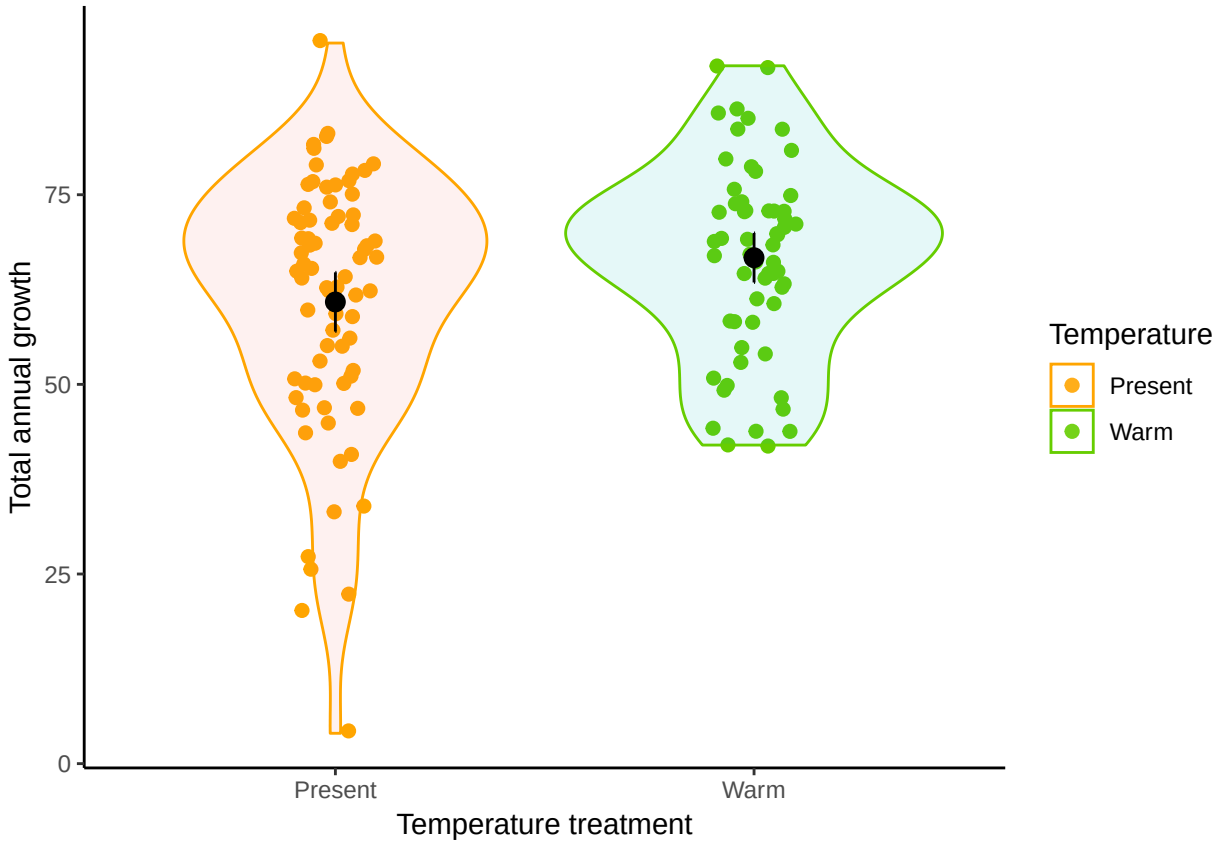
We can add the *mean* and *standard error* of the *total annual growth* to the previous plot. We need to calculate the summary statistics first. Then we add the points to the previous ggplot object.

```
library(Rmisc) ### this package has a function to extract the summary statistics from a dataframe

## first generate data frame with the summary stats
annualgrowth_stat <- summarySE(sample_data,
                               measurevar = "TotLength_growth_annual",
                               groupvars = "Temperature_Treatment_Intro",
                               na.rm = TRUE) ### we put this because we don't have complete info for so
```

```r
colnames(annualgrowth_stat) <- c("Temperature_Treatment_Intro",
                                 "n",
                                 "ave_ag",
                                 "sd_ag",
                                 "se_ag",
                                 "ci_ag")


plot01 +
  geom_point(data = annualgrowth_stat,
             aes(x = Temperature_Treatment_Intro, y = ave_ag),
             size = 3, colour='black' ) +
  geom_errorbar(data = annualgrowth_stat,
                aes(x = Temperature_Treatment_Intro,
                    ymax = ave_ag + 2 * se_ag,
                    ymin = ave_ag - 2 * se_ag,
                    width = 0),
                colour = "black",
                inherit.aes = FALSE)
```



Challenge 03

Use the ggplot object that you modified with `ggThemeAssist` and add *error bar* and *mean* point to it.

Plotting maps with ggplot / ggmap

You can plot **raster** and **vector** spatial data with `ggplot`. Vector data represent geographical phenomena by **points**, **lines** and **polygons**. Raster is another representation of spatial data that consist of equal size pixels. For importing and plotting vector data we use `sf` package.

If you don't have these packages, install them.

```
install.packages("sf")
install.packages("rworldmap") # to download the world map
```

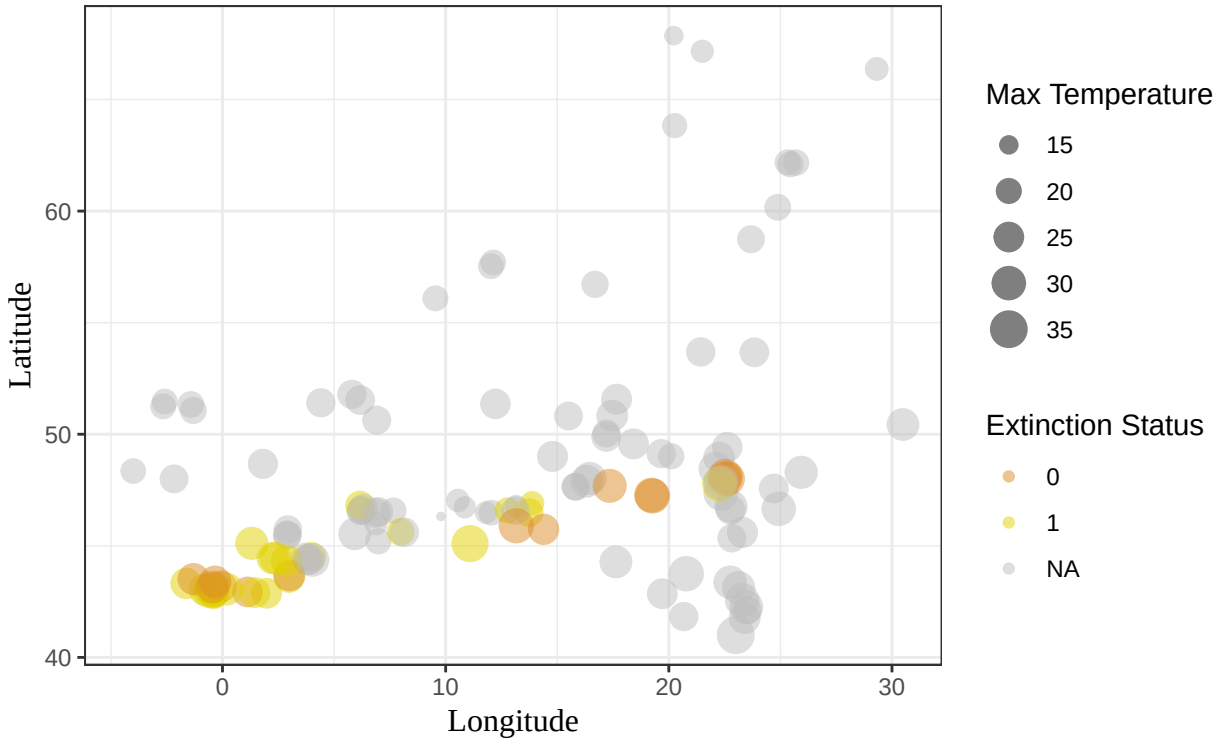
You can plot point location without converting them to spatial object in R. See the following code as an example.

```
sample_data3 <- read.csv("~/Dropbox/Data Viz/Map_of_risks.csv")

plot02 <- ggplot(data = sample_data3, aes(x = Longitude,
                                  y = Latitude,
                                  color = as.factor(Extinction.Status),
                                  size = Average.Maximum.Temperature)) + ### we can specify the size to b
  geom_point(alpha = 0.5) +
  coord_fixed() +
  theme_bw() +
  theme(axis.title = element_text(family = "Times",  size = 12, vjust = 0.75)) +
  labs(colour = "Extinction Status", size = "Max Temperature")

### try a Wes Anderson colour palette if you want, or define your own colours #####
#install.packages("wesanderson")
library(wesanderson)

plot02 + scale_colour_manual(values = wes_palette("FantasticFox1"), na.value = 'grey')
```



Now, we want to load world map, and crop it to the area we need. For this, I use `rworldmap` package to load the world map. This is stored as a `SpatialPolygonDataFrame` object. We need to convert it to **simple features** to use `geom_sf` in `ggplot` package.

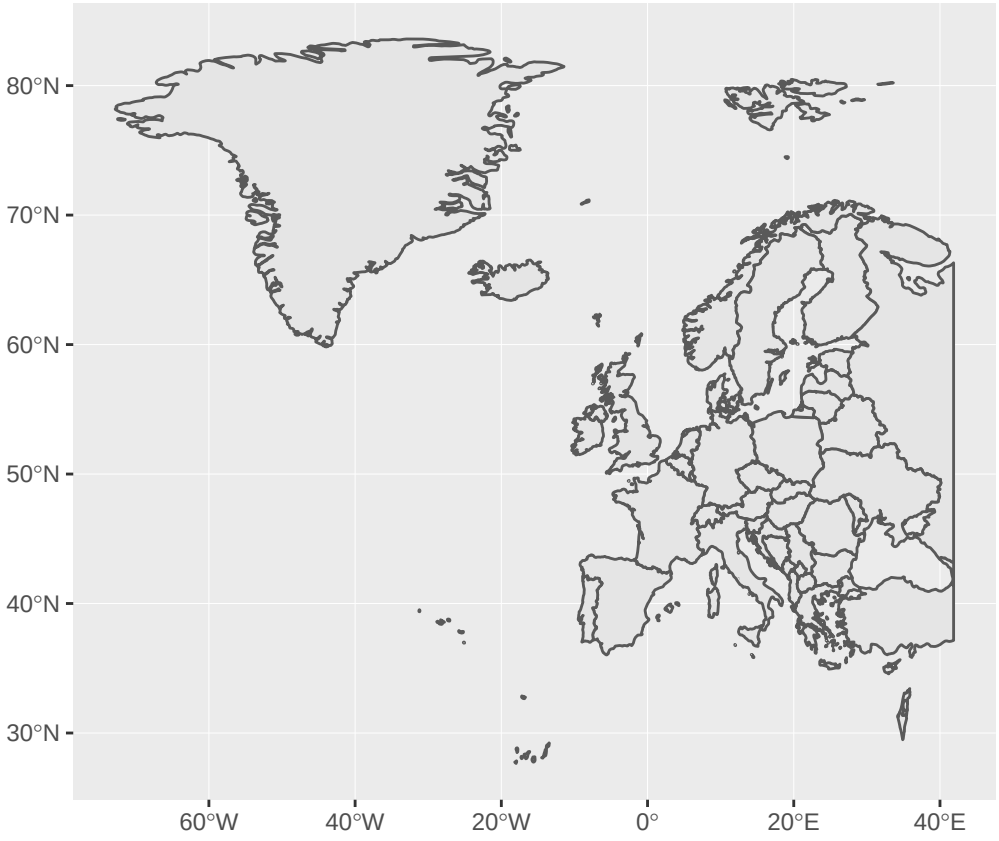
```
library(rgdal)
library(sf) ## if you have trouble, update dplyr package
library(rworldmap)

wmap <- getMap(resolution = "low") # get the world map
wmap <- wmap[which(wmap$REGION == "Europe"), ] # subset Europe
wmap_sf <- st_as_sf(wmap) # change to sf

ext <- c(xmin = -75.01603,
         xmax = 41.86231,
         ymin = 24.33489,
         ymax = 88.4606)
wmap_clip <- st_crop(wmap_sf, ext)
```

Now, we can plot the map by ggplot.

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



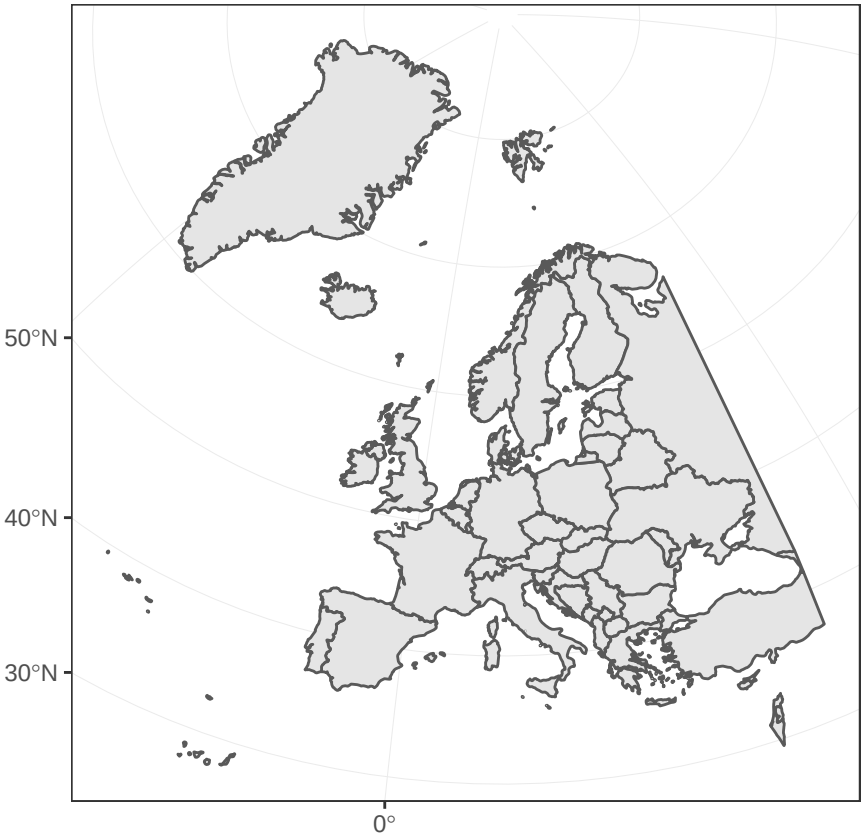
As you can see, *Greenland* is much bigger than reality and it seems as big as Europe! This is because of the **Mercator projection** (*lat and long coordinates*). Different projections might keep some aspect of the map and distort the others. *Mercator projection* preserves the angles but distorts the size of objects as the latitude increases from the Equator to the poles.
Do NOT use this projection in your publications.

We can change this to a projection than better preserve the area across Europe. The **Laea projection (EPSG:3035)** works well.

Let's try it on the map.

```
wmap_proj <- st_transform(wmap_clip, crs = st_crs(3035))

ggplot(data = wmap_proj) +
  geom_sf() +
  theme_bw()
```

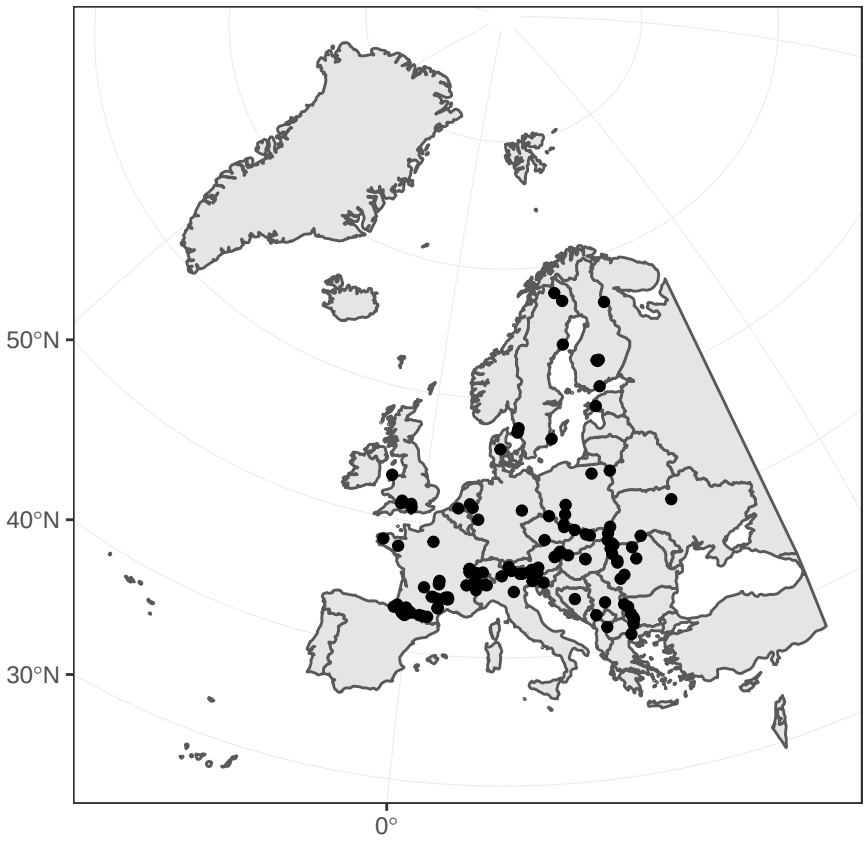


Adding the points to map

To add the points to this map, we need to convert the points to `sf` object and change the projection to match the same locations on the map.

```
thepoints <- st_as_sf(sample_data3, coords = c("Longitude", "Latitude"), crs = 4326)
thepoints <- st_transform(thepoints, crs = st_crs(3035))

ggplot(data = wmap_proj) +
  geom_sf() +
  geom_sf(data = thepoints) +
  theme_bw()
```



Challenge 04

Use the same code to re-create the above map. Change the colour of the points based on `Extinction.Status` column and make the points transparent.

Notice:

- the colour is an *aesthetic* of ggplot
- you can use `as.factor()` to change the continues colour to discrete
- use *ggplot theme assistant* to add more changes
