

# Data Visualization

## R for Stata Users

Luiza Andrade, Leonardo Viotti & Rob Marty

April 2019



- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 ggplot - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics
- 6 ggplot - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs

# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 ggplot - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics
- 6 ggplot - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations
- 10 Appendix

In this session, you'll learn how to use R to produce insightful, meaningful and (hopefully) nice-looking graphs. In particular, you'll use a package called `ggplot2`.

Similarly to the previous session, you can find some references at the end of this presentation that include a more comprehensive discussions on data visualization.

Before we start, let's make sure we're all set:

- 1 Make sure you the packages `ggplot2` and `plotly` are installed and loaded.
- 2 Load the `whr_panel.csv` data set.

# Introduction

```
# Install packages
install.packages("plotly", # We already installed ggplot2
                 dependencies = TRUE)

# Load packages
library(ggplot2)
library(plotly)

#### Load CSV data

# Replace with where your data is
FolderPath <- file.path("YOUR/FOLDER/PATH")

whr <- read.csv(file.path(FolderPath, "whr_panel.csv"),
               header = T,
               stringsAsFactors = F)
```

In our workflow there are usually two distinct uses for plots:

- ❶ **Exploratory analysis:** Quickly visualize your data in an insightful way
  - We'll do this using R's base plot, that allows you to quickly create some basic plots
- ❷ **Publication/Reporting:** Make pretty graphs for a presentation, a project report, or to just show your boss something other than the laziest graph you could come up with
  - We'll do this using `ggplot`, a flexible and powerful tool to plot beautiful graphs
  - `ggplot`'s syntax is more complicated, but it's easier to make your graphs look good with it

# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 ggplot - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics
- 6 ggplot - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations
- 10 Appendix



# Exploratory graphs - Base plot

For exploratory plots, we're going to use Base plot. It is easy to use and can produce useful graphs with very few lines of code.

# Exploratory graphs - Base plot

## Exercise 1:

Plot the `whr_simp` data set you constructed in the previous session using the `plot()` function.

- 1 If you don't have the code from the previous session. Here it is:

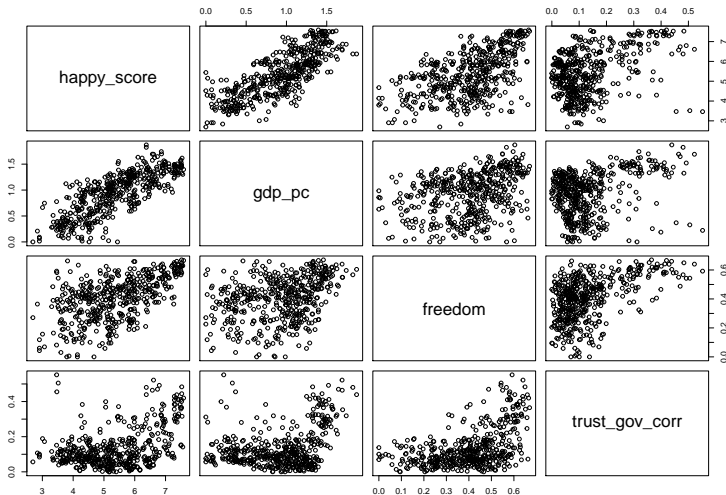
```
# Vector with covariates to be kept
covariates <- c("happy_score",
               "gdp_pc",
               "freedom",
               "trust_gov_corr")
```

```
# subset whr
whr_simp <- whr[, covariates]
```

- 2 Now, pass the name of the data set as the only argument for the plot function

# Exploratory graphs - Base plot

```
plot(whr_simp)
```

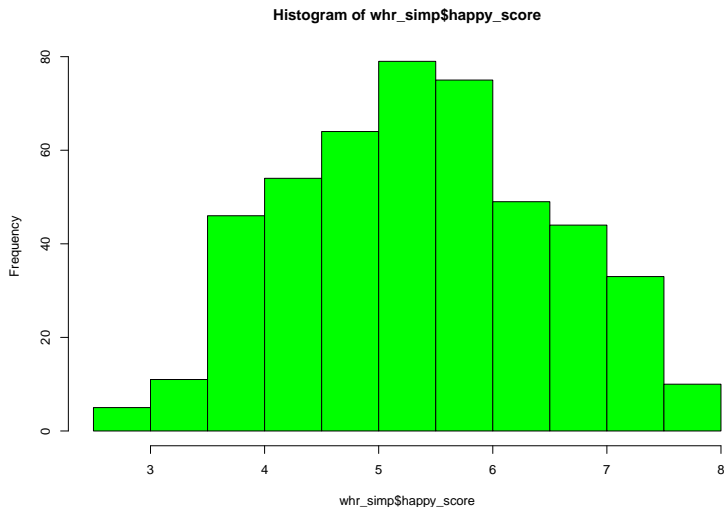


## Exercise 2

- 1 Create a histogram of variable `happy_score` in data frame `whr_simp`. Use the `hist()` function for that.
- 2 Try to set the color as "green". Use the help if you are not sure how to do it.

# Exploratory graphs - Base plot

```
hist(whr_simp$happy_score, col = "green")
```



## Exercise 3

Use the `boxplot()` function to see a description of the happiness score by region.

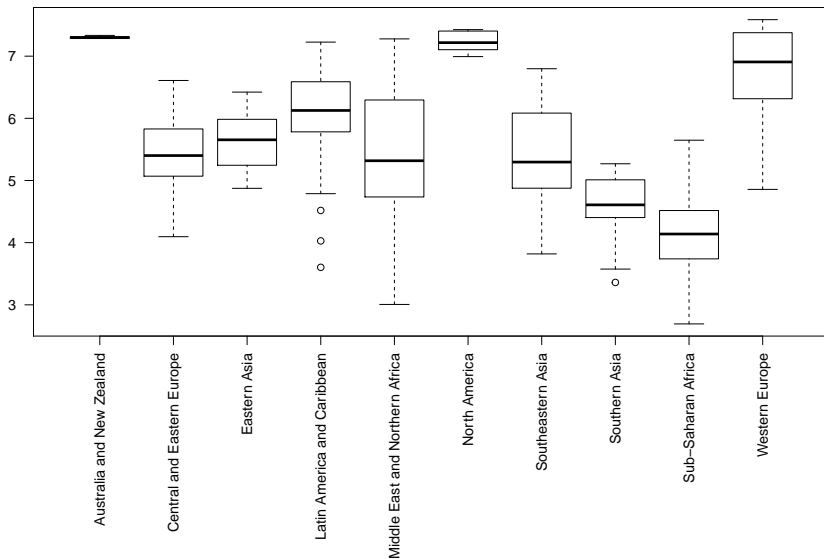
- TIP: The syntax for this function is `variable_to_be_plotted ~ variable_to_plot_over`.

# Exploratory graphs - Base plot

Since these region labels are too long, I'm using an extra argument so we can actually read the x axis.

```
boxplot(whr$happy_score ~ whr$region, las = 2)
```

# Exploratory graphs - Base plot





# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 **ggplot** - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics
- 6 ggplot - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations
- 10 Appendix

The ggplot2 package is an implementation of the theoretical framework for the construction of quantitative graphs developed in The Grammar of Graphics, a book by Leland Wilkinson.

It is a powerful and easy to use tool (once you understand its logic) that produces complex and multifaceted plots.

There are a few reasons why we use ggplot:

- It is generally easier to customize and tailor plots to your needs.
- It works great with almost any kind of plot, including maps, with little variation in syntax.
- It looks good.

Here is a list of `ggplot2` most commonly used “grammar” elements:

Element	Description
Data	The dataset(s) being used.
Aesthetics	How your data is mapped. For example what goes in the X and Y axis, size, shape and color.
Geometries	The visual elements used to represent the data (e.g. lines, dots, bars, etc.)
Themes	All non-data formatting.

## Exercise 4

First, let's create a very simple plot with the happiness score on the Y axis and freedom on the X axis.

- 1 Use the `ggplot` function to store your plot in an object called `p1_happyfree`
  - Since it's the first one, here's the code:

```
p1_happyfree <-  
  ggplot(data = whr,  
         aes(y = happy_score,  
             x = freedom))
```

- 2 Now, try to print your plot. What happens?
  - You can do it by using the `print()` function or just typing the object name in the console

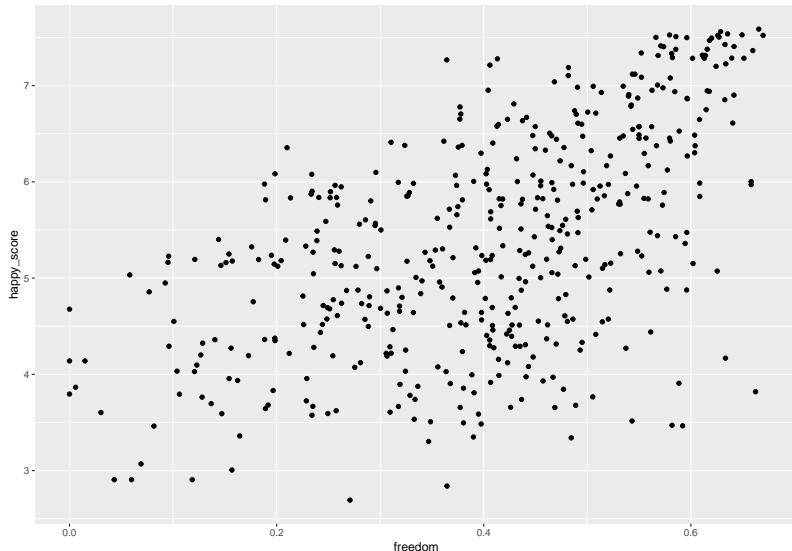
We mapped our variables, but without a geometry ggplot doesn't know how to represent the data.

Let's add a geometry

```
p1_happyfree <-  
  ggplot(whr,  
    aes(y = happy_score,  
        x = freedom)) +  
  geom_point()
```

# ggplot - Introduction

```
print(p1_happyfree)
```



# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 `ggplot` - Introduction
- 4 `ggplot` - Data structure**
- 5 `ggplot` - Aesthetics
- 6 `ggplot` - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations
- 10 Appendix



# ggplot - Data structure

As almost everything in R, `ggplot` is very flexible and can be used in different ways. It is easier to work, however, if your data is in a specific format.

- Data has to be a data frame.
- Long format, especially categorical values.
  - It's better to have one variable with all the categories than a set of dummies.
- Labelled factors.

This is very similar to Stata, with the advantage of not needing to preserve and restore anything.

## ##ggplot - Data structure

Here's an example of how an aggregated (collapsed) data set should look like:

- Since `ggplot` usually works better with data in a specific format, there are a few functions particularly useful to manipulate data
- Here, we'll use the `aggregate()` function
- `aggregate()` works similarly to `collapse` in Stata

# ggplot - Data structure

To create a line plot with ggplot, we will create a new data set containing the average of happiness score by year.

```
# Aggregate annual income by year
```

```
annualHappy<-  
  aggregate(happy_score ~ year,  
            data = whr,  
            FUN = mean)
```

```
head(annualHappy)
```

```
##   year happy_score  
## 1 2015    5.375734  
## 2 2016    5.382185  
## 3 2017    5.354019
```

## Exercise 5

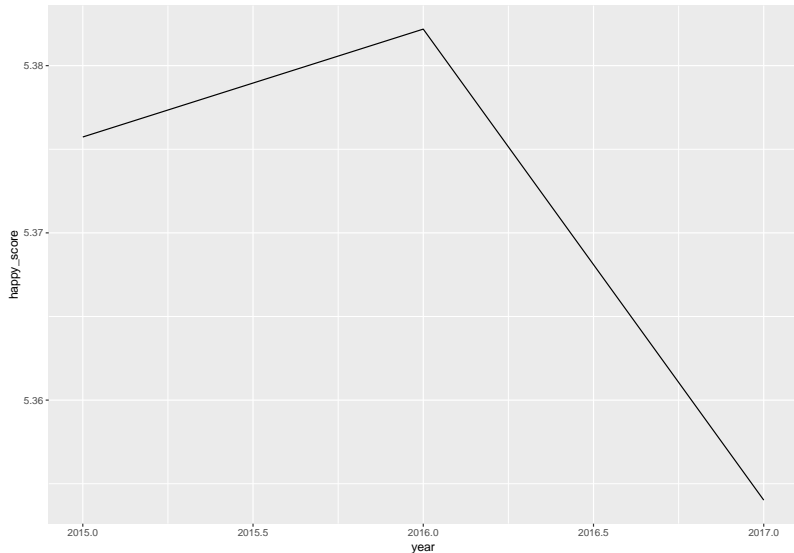
- 1 Use the `ggplot()` function with the `annualHappy` as the data argument, `happy_score` as `y` and `year` as `x` in `aes()`
- 2 This time, add `geom_line()`

# ggplot - Data structure

```
# Plot code with arguments in geom  
p2_happyyear <-  
  ggplot(data = annualHappy,  
         aes(y = happy_score,  
             x = year)) +  
  geom_line()
```

# ggplot - Data structure

```
print(p2_happyyear)
```



Ok, it worked, but that was a really boring plot. Let's do a better one.

## Exercise 6: Part 1

Use the `aggregate()` function to average happiness score, now, by two variables: `year` and `region`.

```
annualHappy_reg <-  
  aggregate(happy_score ~ year + region,  
            data = whr,  
            FUN = mean)
```

# ggplot - Data structure

```
# Aggregate data set
annualHappy_reg <-
  aggregate(happy_score ~ year + region,
            data = whr,
            FUN = mean)

annualHappy_reg[1:10,]
```

	year	region	happy_score
## 1	2015	Australia and New Zealand	7.285000
## 2	2016	Australia and New Zealand	7.323500
## 3	2017	Australia and New Zealand	7.299000
## 4	2015	Central and Eastern Europe	5.332931
## 5	2016	Central and Eastern Europe	5.370690
## 6	2017	Central and Eastern Europe	5.409931
## 7	2015	Eastern Asia	5.626167
## 8	2016	Eastern Asia	5.624167
## 9	2017	Eastern Asia	5.646667
## 10	2015	Latin America and Caribbean	6.144682



## Exercise 6: Part 2

Now, use `ggplot` as in the previous exercise to plot the aggregated data frame.

- 1 This time set the `color` and `group` arguments in the `aes()` function as the treatment variable.
- 2 Finally, add both line and point geoms.

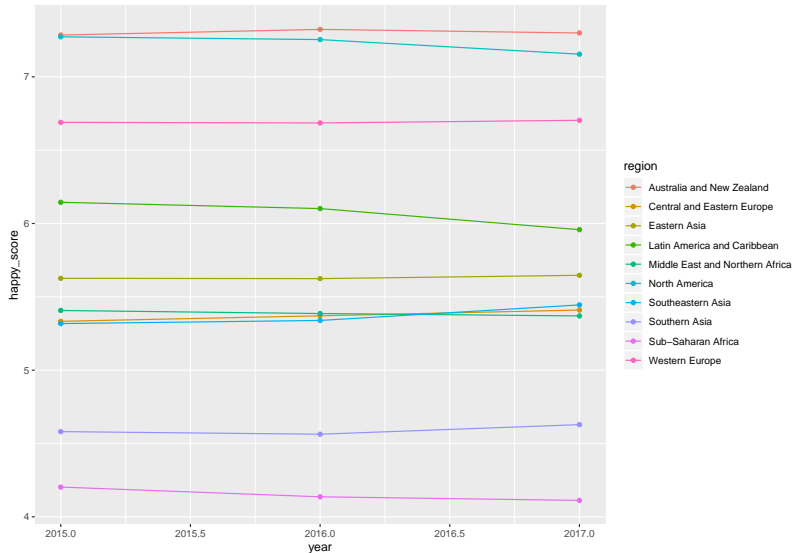
# ggplot - Data structure

```
# Aggregate data set
annualHappy_reg <-
  aggregate(happy_score ~ year + region,
            data = whr,
            FUN = mean)
```

```
# Plot aggregated data
p3_happyreg <-
  ggplot(data = annualHappy_reg,
        aes(y = happy_score,
            x = year,
            color = region,
            group = region)) +
  geom_line() +
  geom_point()

print(p3_happyreg)
```

# ggplot - Data structure



You can also use `ggplot` to combine different data frames in the same plot.

- Everything inside the `ggplot()` function will be shared by all geometries added (e.g. the line and point geometries added to the previous exercise)
- But you can also pass different arguments to different geometries.

In the following example, to showcase just 3 regions, instead of restricting the data to only the interest regions, we are adding 3 separate line geometries.

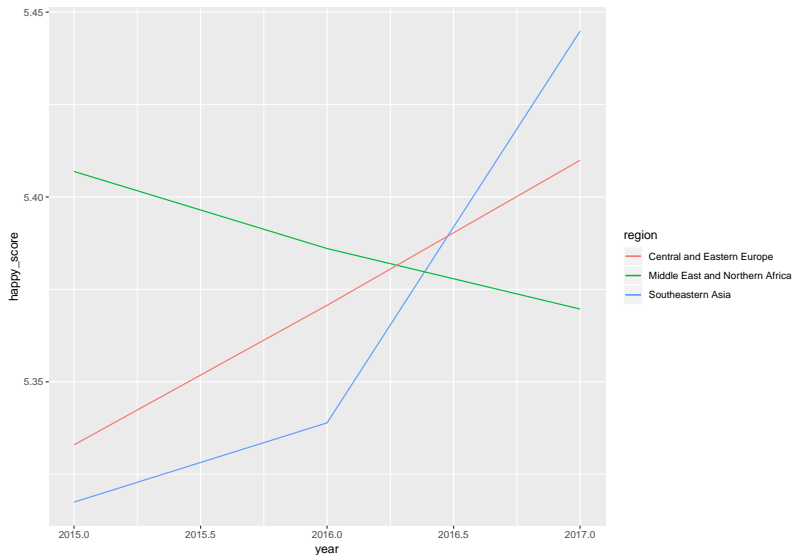
# ggplot - Data structure

```
# Separate datasets
sea <- annualHappy_reg[annualHappy_reg$region == "Southeastern Asia", ]
mena <- annualHappy_reg[annualHappy_reg$region == "Middle East and Northern Africa", ]
cee <- annualHappy_reg[annualHappy_reg$region == "Central and Eastern Europe", ]

# plot with 3 line geometries
p4_happy3reg <-
  ggplot(mapping = aes(y = happy_score,
                        x = year,
                        color = region,
                        group = region)) +
  geom_line(data = sea) + # Southeastern Asia
  geom_line(data = mena) + # Middle East and Northern Africa
  geom_line(data = cee) # Central and Eastern Europe
```

# ggplot and aggregate()

```
print(p4_happy3reg)
```



# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 ggplot - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics**
- 6 ggplot - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations
- 10 Appendix

In `ggplot`, the aesthetic element defines how data is represented in the aesthetics (e.g. color, shape and size) of geometric objects (e.g. points, lines and bars).

## Exercise 7: Part 1

- 1 Create a data frame containing only 2017 observations of `whr`.
  - TIP: This time we want to keep only observations (lines) that meet a certain condition. Use the left argument of the brackets operators. Like this:

```
whr[CONDITION,]
```

- 2 Use `ggplot()` to scatter happiness score and freedom.



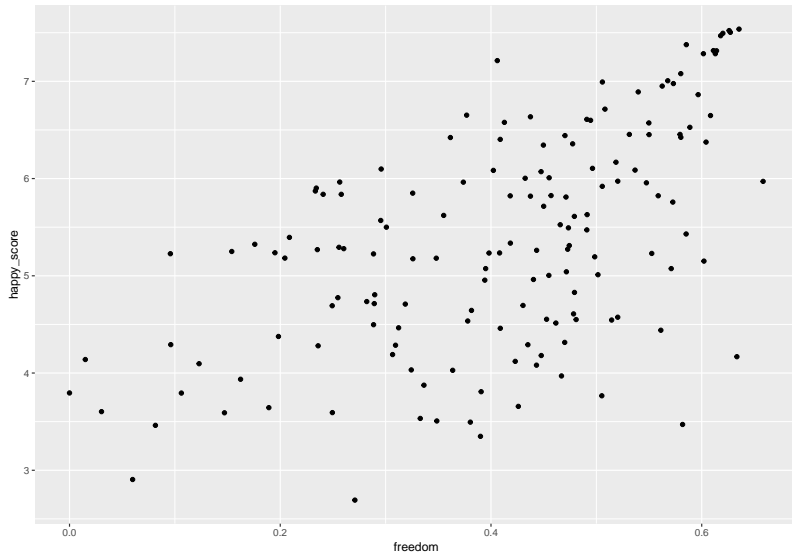
# ggplot - Aesthetics

```
# Subset whr to 2017
whr17 <- whr[whr$year == 2017, ]

# Plot
p5_happyfree17 <-
  ggplot(whr17, aes(x = freedom,
                    y = happy_score)) +
  geom_point()
```

# ggplot - Aesthetics

```
print(p5_happyfree17)
```



Now add the region to `aes()`.

## Exercise 7: Part 2

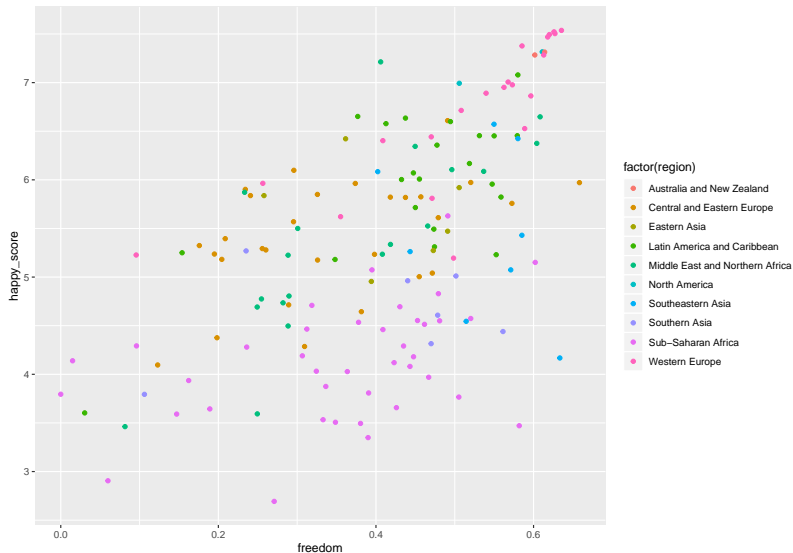
- 1 Use the `region` variable as color in `aes()`.
  - TIP: this is a categorical variable, but it is stored in string format. Use the `factor()` function. Like this:

```
color = factor(region)
```

# ggplot - Aesthetics

```
# Plot code  
p6_happyfree17reg <-  
  ggplot(whr17,  
    aes(x = freedom,  
        y = happy_score)) +  
  geom_point(aes(color = factor(region)))
```

# ggplot - Aesthetics



Now we will combine different arguments of `aes()`

## Exercise 7: Part 3

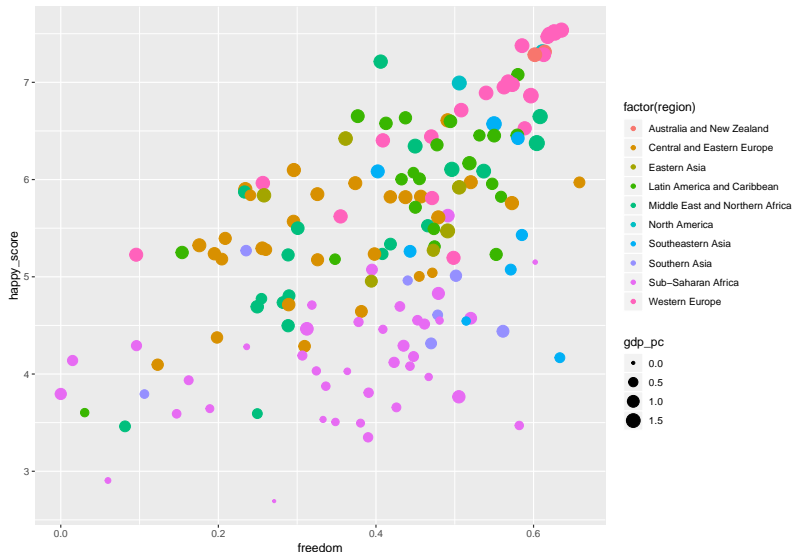
- 1 Add `gdp_pc` to the `size` argument of `aes()` in your last graph.

# ggplot - Aesthetics

```
# Plot code
p6_happyfree17reg <-
  ggplot(whr17,
    aes(x = freedom,
        y = happy_score)) +
  geom_point(aes(color = factor(region),
                 size = gdp_pc))
```

# ggplot - Aesthetics

```
print(p6_happyfree17reg)
```



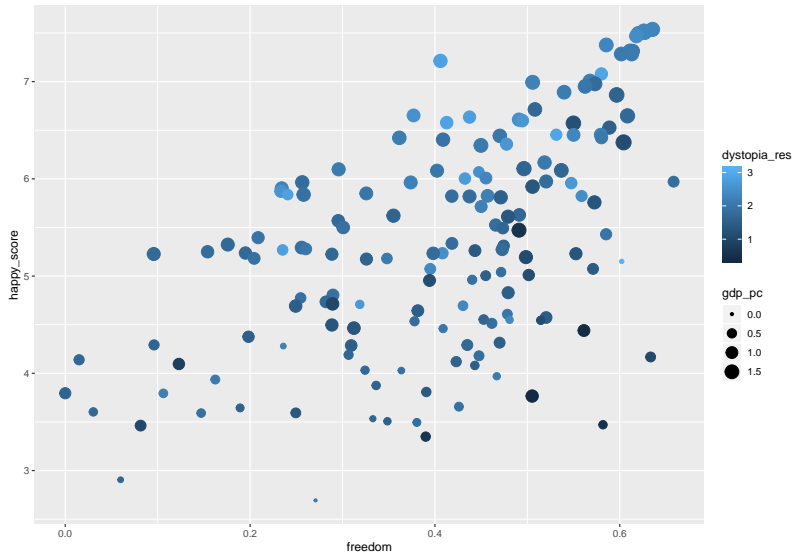


- Because we used a factor variable as the color in the aesthetic, it displayed different categories clearly
- If we had used a numeric variable instead, it would have created a gradient
- You'll not create this for time's sake, but here's how it would look:

*# Plot code*

```
ggplot(whr17, aes(x = freedom,  
                  y = happy_score,  
                  color = dystopia_res,  
                  size = gdp_pc)) +  
  geom_point()
```

# ggplot - Aesthetics



# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 ggplot - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics
- 6 ggplot - Titles and labels**
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations
- 10 Appendix

Lastly, we can add titles, axis labels and format legends. This is done by adding additional layers to the plot.

To add a title and legend to the X and Y axis, we use the functions listed below. They all take a string scalar as argument.

- `ggtitle()` - adds a title to the plot
- `xlab()` - adds a label to X axis
- `ylab()` - adds a label to Y axis

To add legend titles, we will use two functions that control the formatting of the aesthetic. As they can take many different arguments, we use `name` argument sets legend title:

- `scale_color_discrete()` - formatting of the color aesthetic.
- `scale_size_continuous()` - formatting of the size aesthetic.

## Exercise 8

- 1 Copy the code for the graph produced in the previous exercise.
- 2 Use the + symbol to add the layer.
- 3 `ggtitle()`, `xlab()` and `ylab()` take simple strings as inputs.
- 4 For `scale_color_continuous()` and `scale_size_continuous()` you need to specify `name` argument as the desired string.

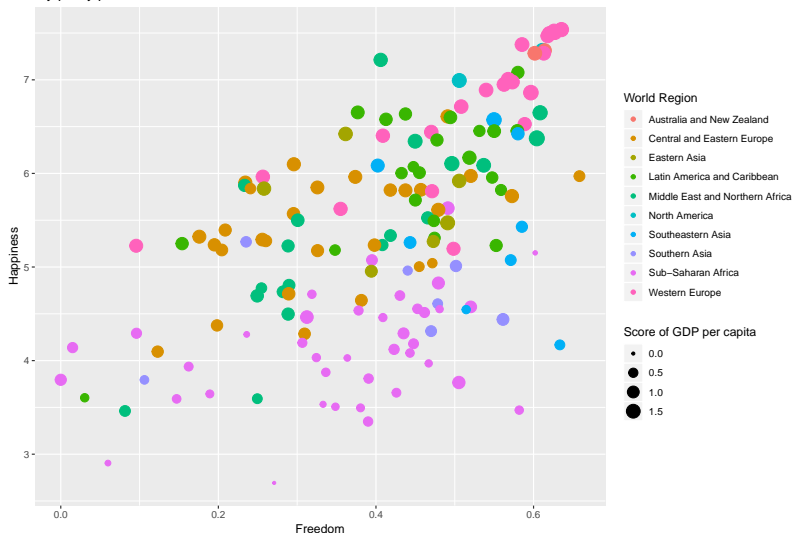
# ggplot - Titles and labels

```
# A properly labelled plot
p6_happyfree17reg <-
  p6_happyfree17reg +
    ggtitle("My pretty plot") +
    xlab("Freedom") +
    ylab("Happiness") +
    scale_color_discrete(name = "World Region") +
    scale_size_continuous(name = "Score of GDP per capita")
```

# ggplot - Titles and labels

p6\_happyfree17reg

My pretty plot





# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 ggplot - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics
- 6 ggplot - Titles and labels
- 7 Saving a plot**
- 8 Interactive graphs
- 9 References and recommendations
- 10 Appendix

Let's save the previous plot

Exercise 9: We will use the `ggsave()` function to save a .png image

- 1 "Add" the `ggsave()` function to the `p6_happyfree17reg` object you created.
- 2 Set the `filename` argument as your outputs path with the file name (including ".png")
- 3 Specify the `width` and `height` arguments. The default unit is inches, but you can use the `units` arguments to specify differently.

# Saving a plot

```
Output <- "YOUR/PATH/HERE"

p6_happyfree17reg +
  ggsave(filename = file.path(Output, "myPrettyGraph.png"),
    width = 9,
    height = 6)
```

# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 ggplot - Introduction
- 4 ggplot - Data structure
- 5 ggplot - Aesthetics
- 6 ggplot - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs**
- 9 References and recommendations
- 10 Appendix

# Interactive graphs

There are several packages to create interactive or dynamic data visualizations with R. Here are a few:

- `leaflet` - R integration to one of the most popular open-source libraries for interactive maps.
- `highcharter` - cool interactive graphs.
- `plotly` - interactive graphs with integration to ggplot.
- `gganimate` - ggplot GIFs.
- `DT` - Interactive table

These are generally, html widgets that can be incorporated in to an html document and websites.

Now we'll use the `ggplotly()` function from the `plotly` package to create an interactive graph!

## Exercise 9

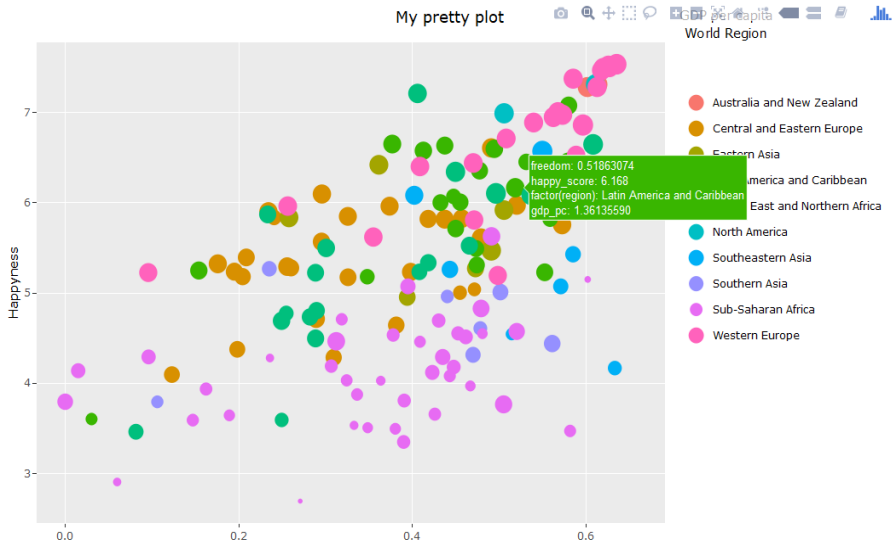
- 1 Load the `plotly` package
- 2 Pass that object with the last plot you created to the `ggplotly()` function

# Interactive graphs

```
# Load package  
library(plotly)  
  
# Use ggplotly to create an interactive plot  
ggplotly(p6_happyfree17reg)
```

# Interactive graphs

(Sorry, this is a .pdf presentation so this is just a screenshot.)





# Homework - Interactive graphs

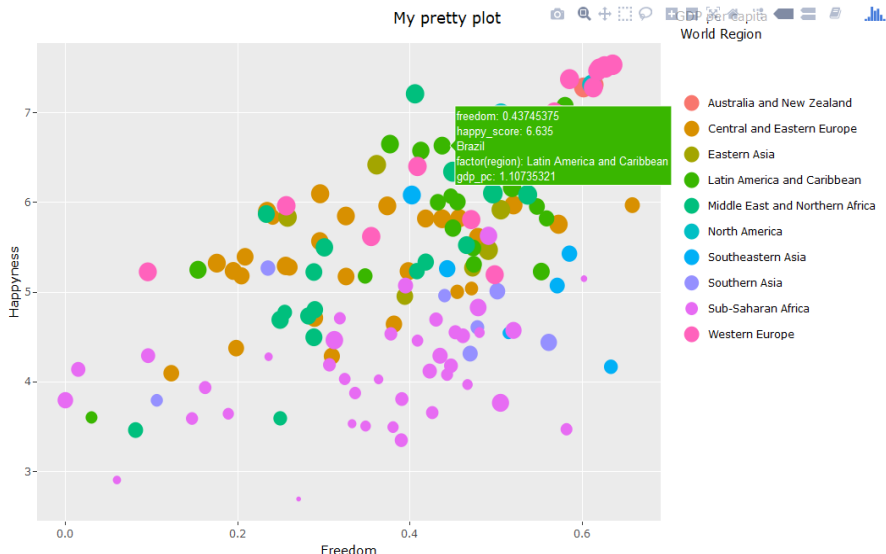
We can add pop-up information by adding another argument to the `aes()` function.

## Customize your ggplotly plot

- 1 Add a `text` argument to the `geom_point aes()` function and assign it to the `country` variable. This argument won't be recognized by `ggplot()`, but will be by `ggplotly()`.
- 2 Use themes to change the background color. You can find a list of default themes in <https://ggplot2.tidyverse.org/reference/ggtheme.html>
  - TIP: type `?ggthemes::` on the console for more options
- 3 Use the `tooltip` argument of the `ggplotly` function to customize what is shown when hovering

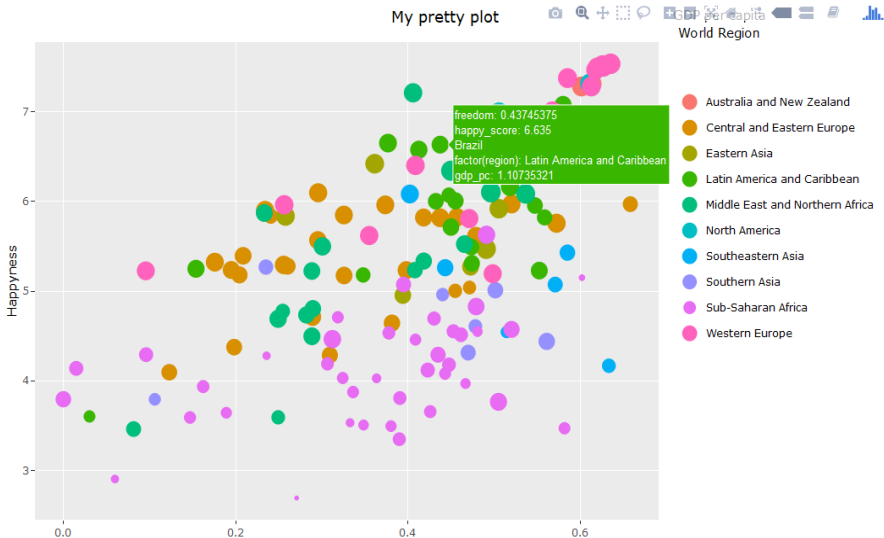
# Interactive graphs

Here's what we've done:



# Interactive graphs

(Sorry, this is a .pdf presentation so this is just a screenshot.)



# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 `ggplot` - Introduction
- 4 `ggplot` - Data structure
- 5 `ggplot` - Aesthetics
- 6 `ggplot` - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations**
- 10 Appendix

# References and recommendations

## Websites:

- Interactive stuff : <http://www.htmlwidgets.org/>
- The R Graph Gallery: <https://www.r-graph-gallery.com/>
- Ggplot official site: <http://ggplot2.tidyverse.org/>

## Online courses:

- Johns Hopkins Exploratory Data Analysis at Coursera:  
<https://www.coursera.org/learn/exploratory-data-analysis>

## Books:

- The grammar of graphics by Leland Wilkinson.
- Beautiful Evidence by Edward Tufte.
- R Graphics cook book by Winston Chang
- R for Data Science by Hadley Wickham and Garrett Grolemund

# Outline

- 1 Introduction
- 2 Exploratory graphs - Base plot
- 3 `ggplot` - Introduction
- 4 `ggplot` - Data structure
- 5 `ggplot` - Aesthetics
- 6 `ggplot` - Titles and labels
- 7 Saving a plot
- 8 Interactive graphs
- 9 References and recommendations
- 10 **Appendix**

R has several built-in graphic devices. A graphic device is the medium where you visually represent a plot. Here are the most come:

- The standard screen device or the plot pane in RStudio.
- PDF file (file device).
- PNG (file device).
- JPEG (file device).

# Differed ways of saving a plot

There are a couple of ways of saving plots in R:

- With ggplot you can use the `ggsave()` function that has a simple syntax and can do most of the dirty work for you.
- Choosing an specific graphics device (other than the default window). `pdf()`, `png()`, `jpeg()` and `bmp()` are functions that call graphic devices that save your plots in these formats. There are a few others built-in and you can check them by typing `?Devices`, and even more in CRAN.



R has four main graphic systems:

- Base plot - most basic plotting system.
- Grid graphics - is a low-level graphics system mainly used by package developers.
- Lattice graphics - easy way to plot high dimensional data or many graphs at the same time.
- Ggplot2 - a flexible and powerful tool to plot beautiful graphs with intuitive syntax